HarvardX: PH125.9 Data Science - Movielens Project

George Cedeño

3/6/2021

Contents

1 Overview

This project is related to the MovieLens Project of the HervardX: PH125.9x Data Science: Capstone course. The report begins with a general idea and objective of the project.

Then the given dataset will be prepared and setup. An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict movie ratings until a final model. Results will be explained. Finally the report ends with some concluding remarks.

1.1 Introduction

Recommendation systems use ratings that users have given to items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon, are able to collect massive datasets that can be used to predict what rating a particular user will give to a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

For this project, rated movies are analyzed. Recommendation systems are one of the most commonly used models in machine learning algorithms. In fact the success of Netflix is said to be based on its strong recommendation system. The Netflix prize (open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films), in fact, represent the high importance of algorithm for products recommendation system.

For this project I will create a movie recommendation system using the 10M version of MovieLens dataset, collected by GroupLens Research.

1.2 AIM:

The aim of this project is to train a machine learning algorithm that predicts user ratings (from 0.5 to 5 stars) using the inputs of a provided subset (edx dataset) to predict movie ratings in the provided validation set.

The value used to evaluate algorithm performance is the Root Mean Square Error, or RMSE. RMSE is a measure of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset; a lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers. Five models will be developed and their resulting RMSE will be compared in order to assess their quality. The evaluation criteria for this algorithm is to not use the validation set until the end to assess quality of the model.

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

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

Finally, the best resulting model will be used to predict the movie ratings.

1.3 Dataset

The MovieLens dataset is automatically downloaded

- [MovieLens 10M dataset] https://grouplens.org/datasets/movielens/10m/
- [MovieLens 10M dataset - zip file] http://files.grouplens.org/datasets/movielens/ml-10m.zip

```
############################################################
# Create edx set, validation set, and submission file
############################################################
# Note: this process could take a couple of minutes for loading required package: tidyverse and package
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")
```

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- read.table(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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

In order to predict in the most possible accurate way the movie rating of the users that haven't seen the movie yet, the he MovieLens dataset will be splitted into 2 subsets that will be the "edx", a training subset to train the algorithm, and "validation" a subset to test the movie ratings.

```
# The Validation subset will be 10% of the MovieLens data.
set.seed(1)
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 subset:
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)

# Training set used will be 50% of edx data to train algorithm's
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.5, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

# Add rows removed from validation set back into train set
removed <- anti_join(temp, test)
train <- rbind(train, removed)
```

As algorithm development is to be carried out on the "edx" subset only (90% of the data), the edx dataset was split again into separate training & test sets for model development. The "validation" subset (10% of the data) will be used in the end to test the final algorithm.

2 Methods and Analysis

2.1 Data Analysis

To get familiar with the dataset, we find the first rows of "train" subset as below. The subset contain the six variables "userID", "movieID", "rating", "timestamp", "title", and "genres". Each row represent a single rating of a user for a single movie.

```
##    userId movieId rating timestamp                                    title
## 4       1     292      5 838983421                             Outbreak (1995)
## 6       1     329      5 838983392             Star Trek: Generations (1994)
## 7       1     355      5 838984474                   Flintstones, The (1994)
## 8       1     356      5 838983653                       Forrest Gump (1994)
## 10      1     364      5 838983707                     Lion King, The (1994)
## 11      1     370      5 838984596 Naked Gun 33 1/3: The Final Insult (1994)
##                                   genres
## 4               Action|Drama|Sci-Fi|Thriller
## 6              Action|Adventure|Drama|Sci-Fi
## 7                    Children|Comedy|Fantasy
## 8                  Comedy|Drama|Romance|War
## 10 Adventure|Animation|Children|Drama|Musical
## 11                           Action|Comedy
```

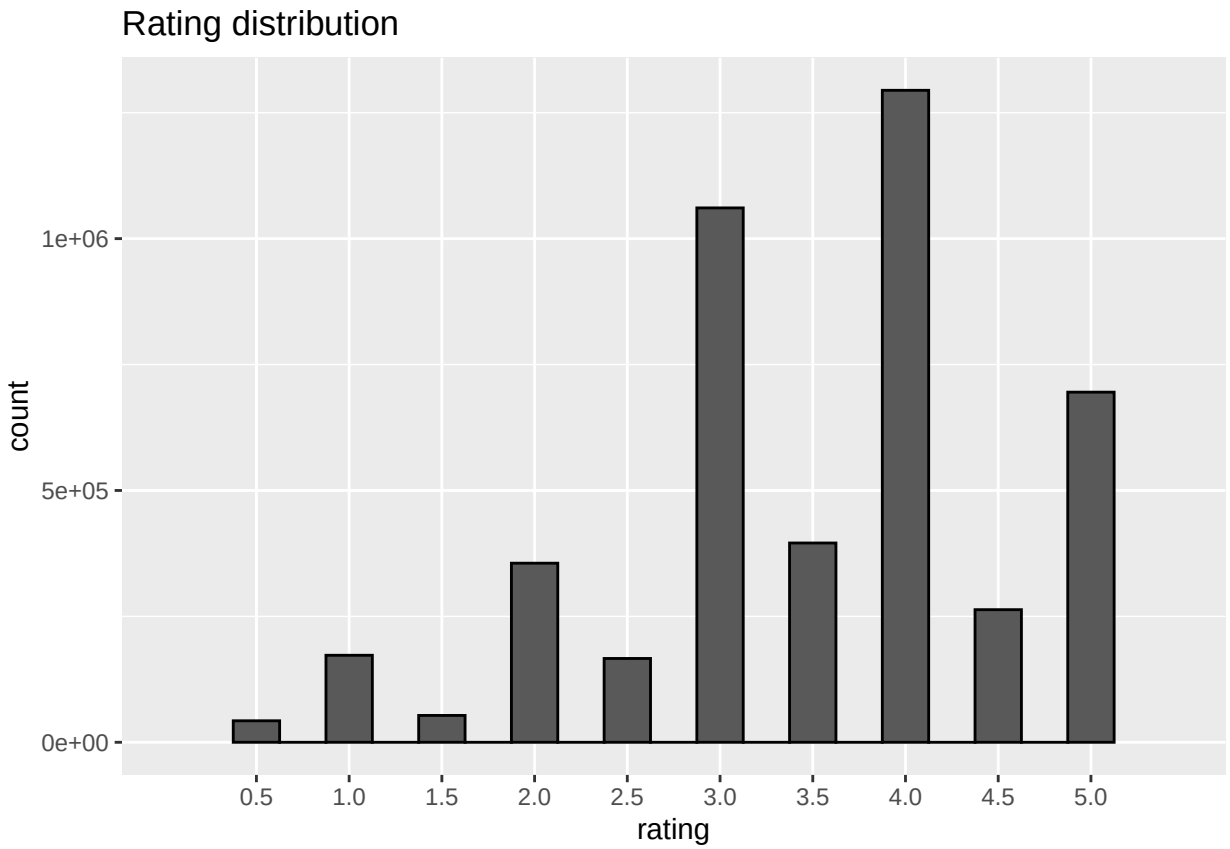
A summary of the subset confirms that there are no missing values.

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :8.229e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35735   Median : 1833   Median :4.000   Median :1.036e+09
##  Mean   :35870   Mean   : 4121   Mean   :3.513   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:4500245     Length:4500245
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The total of unique users, movies, and genres in the train subset is 69,878 unique users, 10,677 different movies, and 797 genres:

```
##   n_users n_movies n_genres
## 1   69878    10677      797
```

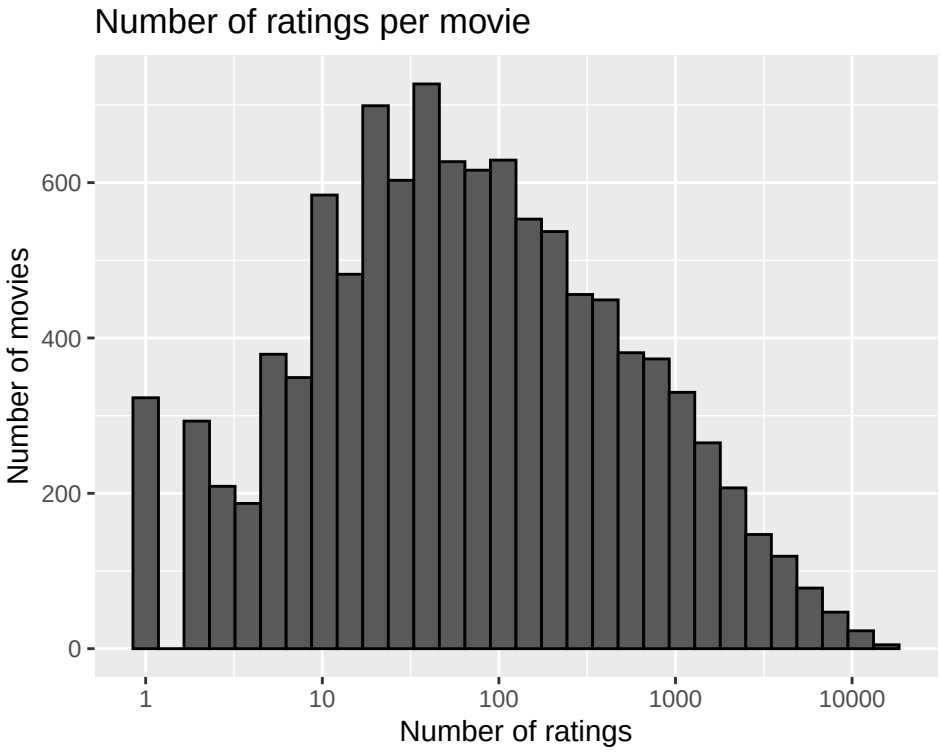
Users have a preference to rate movies rather higher than lower as shown by the distribution of ratings below. 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half rating are less common than whole star ratings.



We can observe that some movies have been rated more often than others, while some have very few ratings and sometimes only one rating. This will be important for our model as very low rating numbers might results in untrustworthy estimate for our predictions. In fact, ~300 movies have been rated only once.

Thus regularization and a penalty terms will be applied to the models in this project. Regularization is a technique used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting (the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably). Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values.

```
train %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  ggtitle("Number of ratings per movie")
```



As 20 movies that were rated only once appear to be obscure, predictions of future ratings for them will be difficult.

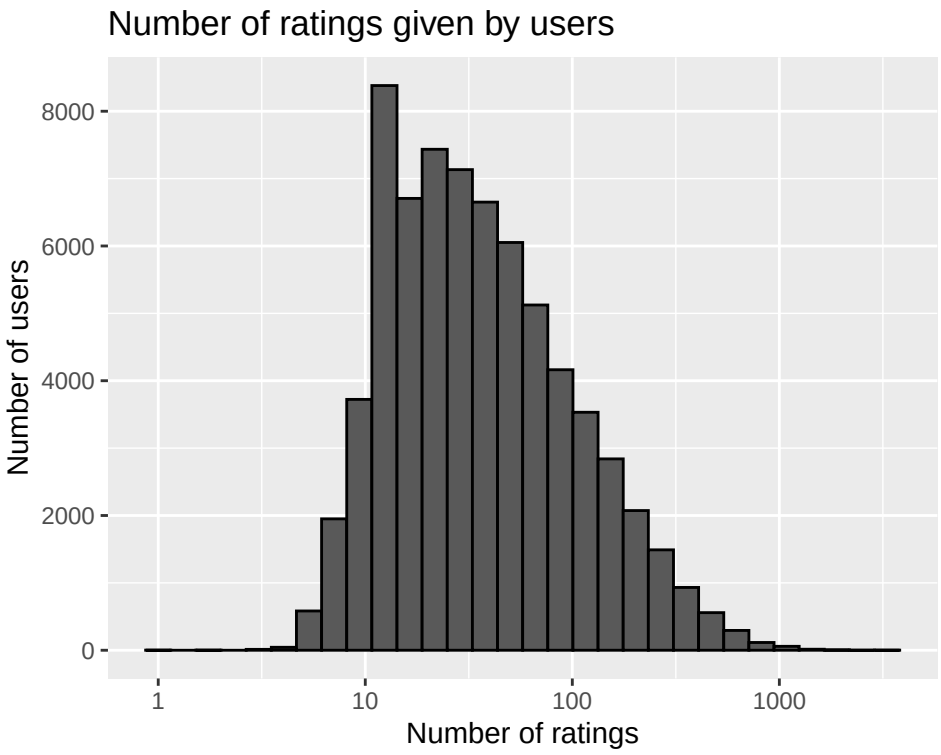
```
train %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(train, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = count) %>%
  slice(1:20) %>%
  knitr::kable()
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	3.0	1
100 Feet (2008)	2.0	1
29th Street (1991)	3.5	1
3 Bad Men (1926)	1.5	1
30 Years to Life (2001)	3.0	1
4 (2005)	2.5	1
5 Centimeters per Second (Byôsoku 5 senchimêtoru) (2007)	3.5	1
55 Days at Peking (1963)	4.5	1
7 Faces of Dr. Lao (1964)	3.5	1
7 Men from Now (1956)	4.5	1
Accused (Anklaget) (2005)	0.5	1
Ace High (Quattro dell'Ave Maria, I) (1968)	2.0	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1

title	rating	n_rating
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...) (1971)	1.5	1
Aerial, The (La Antena) (2007)	5.0	1
Africa addio (1966)	3.0	1
Alley Cats, The (1966)	4.0	1
Along Came Jones (1945)	3.0	1
Altered (2006)	2.0	1

We can observe that the majority of users have rated between 30 and 100 movies. So, a user penalty term need to be included later in our models.

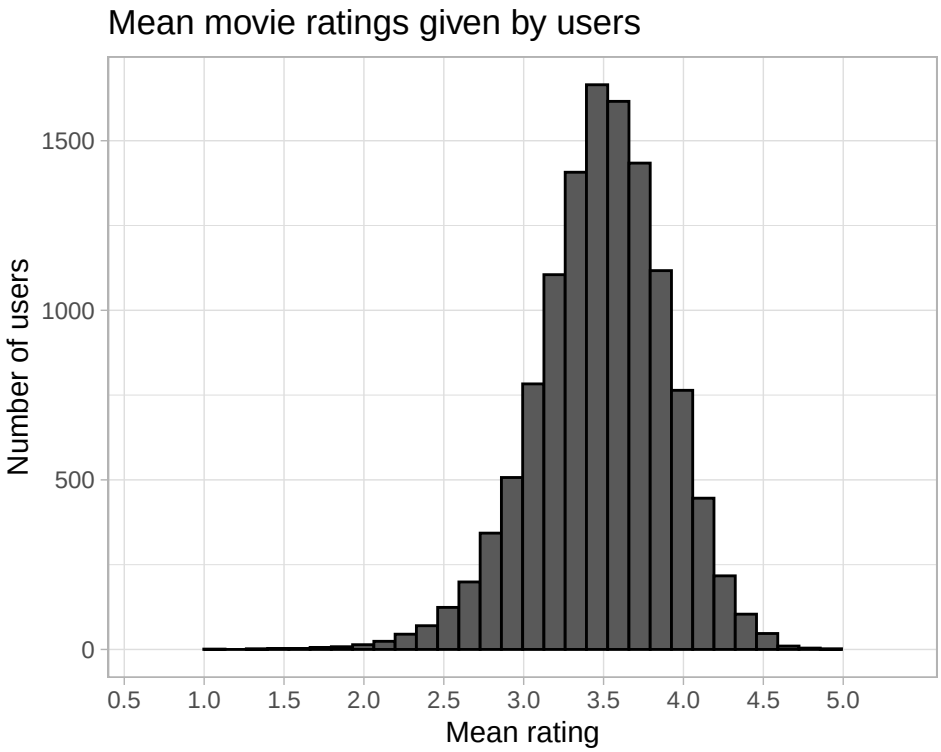
```
train %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of users") +
  ggtitle("Number of ratings given by users")
```



Furthermore, users differ vastly in how critical they are with their ratings. Some users tend to give much lower star ratings and some users tend to give higher star ratings than average. The visualization below includes only users that have rated at least 100 movies.

```
train %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
```

```
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_light()
```



2.2 Modelling Approach

We write now the loss-function, previously anticipated, that compute the RMSE, defined as follows:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations.

The RMSE is our measure of model accuracy.

We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If its result is larger than 1, it means that our typical error is larger than one star, which is not a good result.

The written function to compute the RMSE for vectors of ratings and their corresponding predictions is:

```r
set.seed(1)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The lower the better, as said previously.

2.2.1 I. Average movie rating model

The first basic model predicts the same rating for all movies, so we compute the dataset's mean rating. The expected rating of the underlying data set is between 3 and 4. We start by building the simplest possible recommender system by predicting the same rating for all movies regardless of user who give it. A model based approach assumes the same rating for all movie with all differences explained by random variation :

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimizes the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings: The expected rating of the underlying data set is between 3 and 4.

```r
set.seed(1)
mu_hat <- mean(train$rating)
mu_hat
```

```
## [1] 3.512541
```

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

```r
set.seed(1)
naive_rmse <- RMSE(test$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060509
```

Here, we represent results table with the first RMSE:

```r
set.seed(1)
rmse_results <- data.frame(model="Naive Mean-Baseline Model", RMSE=naive_rmse)
rmse_results %>% knitr::kable()
```

model	RMSE
Naive Mean-Baseline Model	1.060509

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

In order to do better than simply predicting the average rating, we incorporate some of insights we gained during the exploratory data analysis.

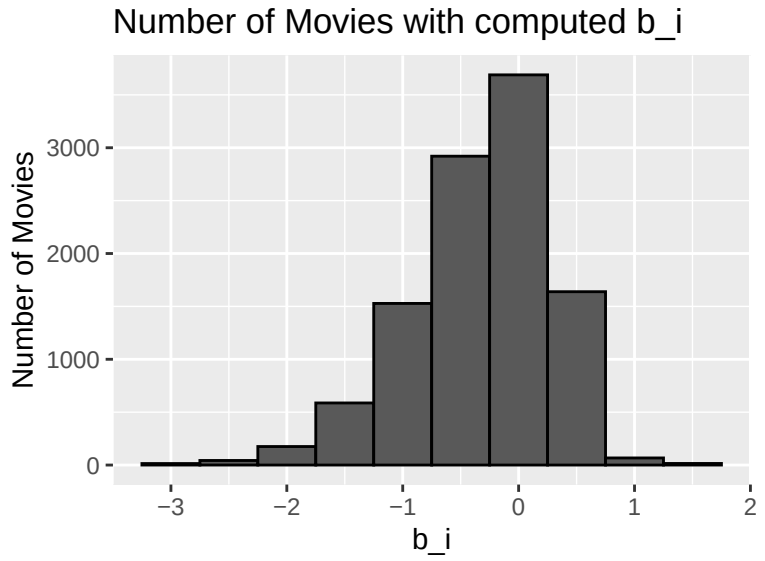
2.2.2 II. Movie effect model

To improve above model we focus on the fact that, from experience, we know that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

The histogram is left skewed, implying that more movies have negative effects

```
set.seed(1)
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
movie_avgs %>% qplot(b_i, geom="histogram", bins = 10, data = ., color = I("black"),
                     ylab = "Number of Movies", main = "Number of Movies with computed b_i")
```



This is called the penalty term movie effect.

Our prediction improve once we predict using this model.

```
set.seed (1)
predicted_ratings <- mu_hat + test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          data.frame(model="Movie effect model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

model	RMSE
Naive Mean-Baseline Model	1.0605093
Movie effect model	0.9443772

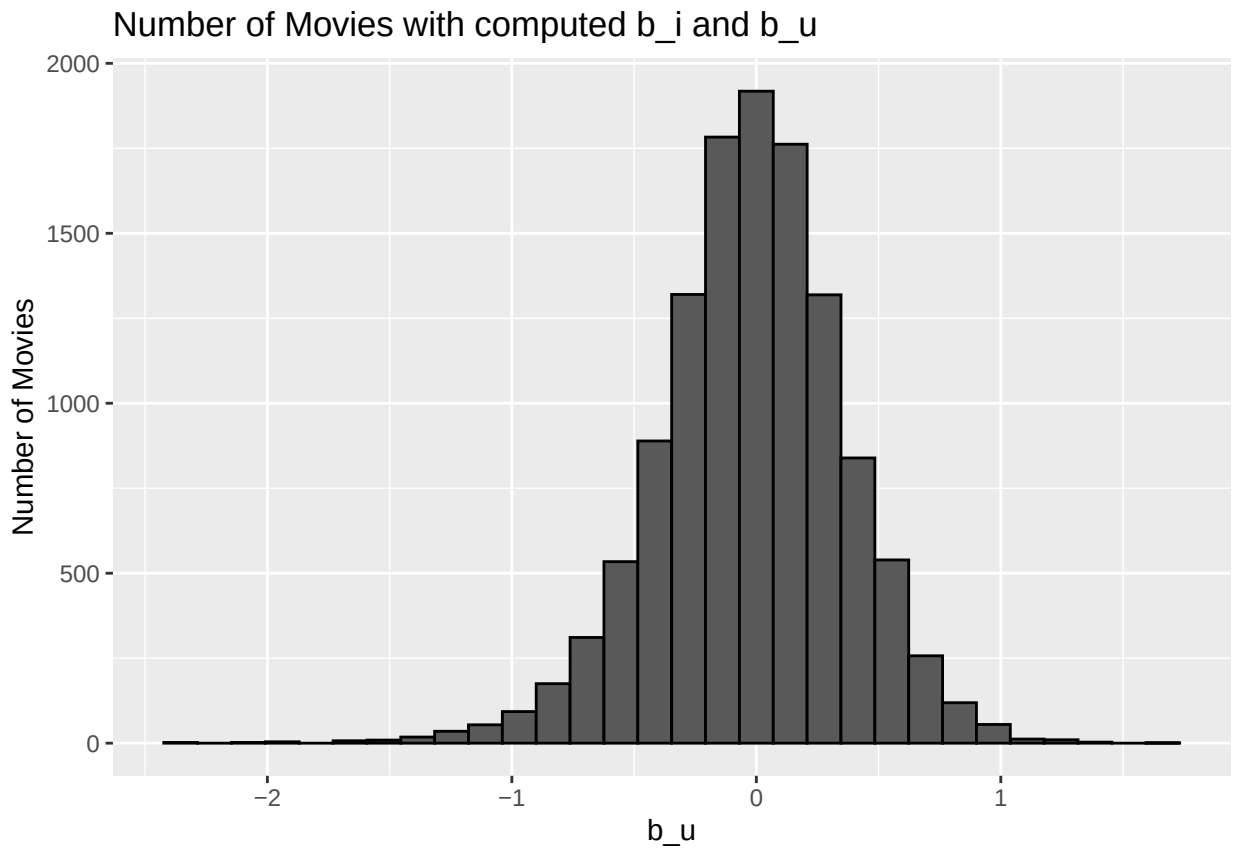
So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ , we predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average.

We can see an improvement but this model does not consider the individual user rating effect.

2.2.3 III. Movie and user effect model

We compute the average rating for user μ, for those that have rated over 100 movies, said penalty term user effect. In fact users affect the ratings positively or negatively.

```
set.seed(1)
user_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
user_avgs %>% qplot(b_u, geom="histogram", bin=30, data=., color=I("black"),
                    ylab = "Number of Movies", main = "Number of Movies with computed b_i and b_u")
```



There is substantial variability across users as well: some users are very cranky and other love every movie. This implies that further improvement to our model my be:

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

where b_u is a user-specific effect. If a cranky user (negative b_u rates a great movie (positive b_i), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

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

```
set.seed(1)
user_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

We can now construct predictors and see RMSE improves:

```
set.seed(1)
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          data.frame(model="Movie and user effect model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

model	RMSE
Naive Mean-Baseline Model	1.0605093
Movie effect model	0.9443772
Movie and user effect model	0.8698469

Our rating predictions further reduced the RMSE. But we made stil mistakes on our first model (using only movies). The supposes "best " and "worst "movie were rated by few users, in most cases just one user. These movies were mostly obscure ones. This is because with a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely. Large errors can increase our RMSE.

2.2.4 IV. Movie, user, and Genre effect model

We computed the average popularity for movies μ and then include the penalty term for genre effect. Genres can also affect ratings positively or negatively based on user preference.

```
set.seed(1)
genre_pop <- train %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))
```

There is substantial variability across genres as well: not every user is interested in every genre. This implies that further improvement to our model my be:

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

where b_g is a genre-specific effect. If a sad user (negative b_u) wants to see happy movies all month (positive b_g) but ends up watching a tear-jerker movie that was recommended simply due to its rating (positive b_i), the effects are conditional (negative b_i) and we may inaccurately predict that this user gave this great movie a 5 rather than a 3.

We compute an approximation by computing μ, b_i, b_u and estimating b_g, as the average of

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

We can now construct predictors and see RMSE improves:

```
set.seed(1)
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_pop, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  pull(pred)
model_3_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          data.frame(model="Movie, user, and genre effect model",
                                     RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

model	RMSE
Naive Mean-Baseline Model	1.0605093
Movie effect model	0.9443772
Movie and user effect model	0.8698469
Movie, user, and genre effect model	0.8695020

Our rating predictions further reduced the RMSE. But we made mistakes on our second model (using only movies and user effects). Genres can affect ratings and the moods of users. If a really upset user saw a low-rated movie with a sweet ending, the rating may increase... sparking more interest amongst users that enjoy happy movies. Understanding which genre's users gravitate towards may reduce uncertainty. Therefore, since a users genre-preference varies, larger estimates of b_u, negative or positive, are more likely. Again, large errors can increase our RMSE.

Until now, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one number, one prediction, not an interval. For this we introduce the concept of regularization, that permits to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i to the sum of squares equation that we minimize. So having many large b_i, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

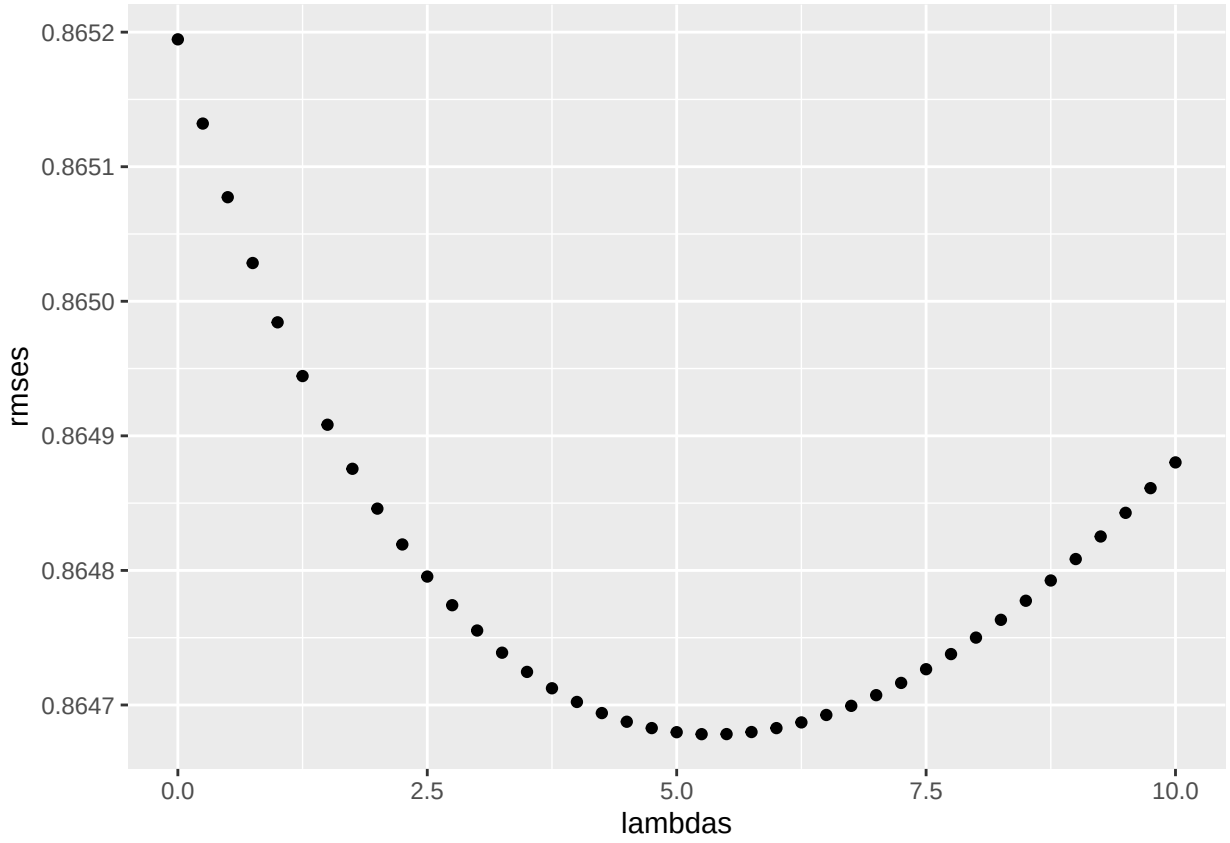
2.2.5 V. Regularized movie and user effect model

So estimates of b_i, b_u, and b_g are caused by movies with very few ratings, some users that only rated a very small number of movies, and genre-specific preferences. Hence these can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i, b_u, and b_g in case of small number of ratings.

```
set.seed(1)
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  # Calculate the average of all ratings in 90% of data
  mu <- mean(edx$rating)
  # Add movie effect
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  # Add user effect
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  # Add genre effect
  b_g <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  # Compute the predicted ratings on validation dataset (10% of data)
  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)
  # Predict the RMSE on the validation set (10% of data)
  return(RMSE(predicted_ratings, validation$rating))
})
```

We plot RMSE vs lambdas to select the optimal lambda

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



For the full model, the optimal lambda is:

```
set.seed(1)
lambda <- lambdas[which.min(rmses)]
lambda
```

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

For the full model, the optimal lambda is: 5.25

The new results will be:

```
set.seed(1)
rmse_results <- bind_rows(rmse_results,
                          data_frame(model="Regularized movie and user effect model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

model	RMSE
Naive Mean-Baseline Model	1.0605093
Movie effect model	0.9443772
Movie and user effect model	0.8698469
Movie, user, and genre effect model	0.8695020
Regularized movie and user effect model	0.8646783

3 Results

The RMSE values of all the represented models are the following:

model	RMSE
Naive Mean-Baseline Model	1.0605093
Movie effect model	0.9443772
Movie and user effect model	0.8698469
Movie, user, and genre effect model	0.8695020
Regularized movie and user effect model	0.8646783

We therefore found the lowest value of RMSE that is 0.8644501.

4 Discussion

So we can confirm that the final model for our project is the following:

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

This model works well if the average user doesn't rate a particularly good/popular movie with a large positive b_i, by disliking a particular movie.

5 Conclusion

We can affirm to have built a machine learning algorithm to predict movie ratings with MovieLens dataset. The regularized model including the effect of user is characterized by the lower RMSE value and is hence the optimal model to use for the present project. The optimal model characterised by the lowest RMSE value (0.8644501). Other machine learning models may improve the results further, but hardware limitations, as the RAM, are a constraint.

6 Appendix - Enviroment

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

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

```
version
```

```
##                _
## platform       x86_64-apple-darwin17.0
## arch           x86_64
## os             darwin17.0
## system         x86_64, darwin17.0
## status
## major          4
## minor          0.2
## year           2020
## month          06
## day            22
## svn rev        78730
## language       R
## version.string R version 4.0.2 (2020-06-22)
## nickname       Taking Off Again
```