

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

Data Science: Capstone Project for Harvardx Professional Data Science Certificate
(Movielens Project: PH125.9x)

Delpagodage Ama Nayanahari Jayaweera

25 November 2023

Contents

1	Chapter 1	2
1.1	Overview	2
1.2	Introduction	2
1.3	Executive Summary	2
1.4	Dataset	2
2	Chapter 2	4
2.1	Methods and Analysis	4
2.1.1	Data Analysis	4
2.2	Modelling Approach	8
2.2.1	Average movie rating model	9
2.2.2	Movie effect model	10
2.2.3	Movie and user effect model	11
2.2.4	Regularized movie and user effect model	12
3	Chapter 3	15
3.1	Results	15
4	Chapter 4	15
4.1	Discussion	15
5	Chapter 4	16
5.1	Conclusion	16
5.2	Limitations	16
5.3	Future Work	16
6	Reference	17

1 Chapter 1

1.1 Overview

This project is related to the MovieLens Project of the HarvardX: PH125.9x Data Science: Capstone course. The present report begins by providing a general overview of the project and describing its objectives. The given dataset is then prepared and set up for analysis.

An exploratory data analysis is conducted to develop a machine learning algorithm capable of predicting movie ratings. The process involves refining the model until a final version is achieved. The results are explained in detail. Finally, the report concludes with some insightful remarks.

1.2 Introduction

Recommendation systems utilize user ratings to provide personalized suggestions based on their preferences. Companies like Amazon, which offer diverse products and allow users to rate them, accumulate extensive datasets. These datasets enable the prediction of user ratings for specific items, and items with predicted high ratings are recommended to users, enhancing their overall experience.

This approach is not limited to products and extends to various domains, such as movies, as demonstrated in our project. Recommendation systems represent a widely employed model in machine learning algorithms. Netflix, for instance, is recognized for its success, largely attributed to a robust recommender system. The significance of such algorithms in product recommendation systems is underscored by events like the Netflix Prize—an open competition seeking the best collaborative filtering algorithm to predict user ratings for films solely based on previous ratings, without additional information about users or films.

Our project focuses on the development of a movie recommendation system using the 10M version of the MovieLens dataset, curated by GroupLens Research.

1.3 Executive Summary

The objective of this project is to train a machine learning algorithm capable of predicting user ratings (ranging from 0.5 to 5 stars). The algorithm will utilize a provided subset (edx dataset supplied by the staff) to predict movie ratings in a given validation set.

Algorithm performance will be assessed using the Root Mean Square Error (RMSE), a widely used metric that measures the differences between model-predicted values and observed values. RMSE serves as an accuracy metric, with lower values indicating better performance. It is particularly sensitive to outliers, as larger errors have a more significant impact on the RMSE.

Four models will be developed and compared based on their resulting RMSE to evaluate their quality. The benchmark for this algorithm is an RMSE expected to be lower than 0.86490.

The RMSE computation function, for vectors of ratings and their corresponding predictors is as follows:

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

Finally, the most effective model will be utilized for forecasting movie ratings.

1.4 Dataset

The MovieLens dataset is automatically downloaded

1. [MovieLens 10M dataset] <https://grouplens.org/datasets/movielens/10m/>
2. [MovieLens 10M dataset - zip file] <https://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 caret
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")
dl <- tempfile()
download.file("https://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(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

To enhance the accuracy of predicting movie ratings for users who haven't seen the movie yet, the MovieLens dataset will be divided into two subsets: “edx,” serving as the training subset to train the algorithm, and “validation,” a subset for testing 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)
```

The algorithm development will exclusively take place on the “edx” subset, with the “validation” subset reserved for testing the final algorithm.

2 Chapter 2

2.1 Methods and Analysis

2.1.1 Data Analysis

To acquaint ourselves with the dataset, examine the initial rows of the “edx” subset as shown below. This subset comprises six variables: “userID”, “movieID”, “rating”, “timestamp”, “title”, and “genres.” Each row corresponds to a singular rating provided by a user for a specific movie.

```
##  userID movieId rating timestamp title genres
## 1      1      122      5 838985046 <NA> <NA>
## 2      1      185      5 838983525 <NA> <NA>
## 3      1      231      5 838983392 <NA> <NA>
## 4      1      292      5 838983421 <NA> <NA>
## 5      1      316      5 838983392 <NA> <NA>
## 6      1      329      5 838983392 <NA> <NA>
```

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

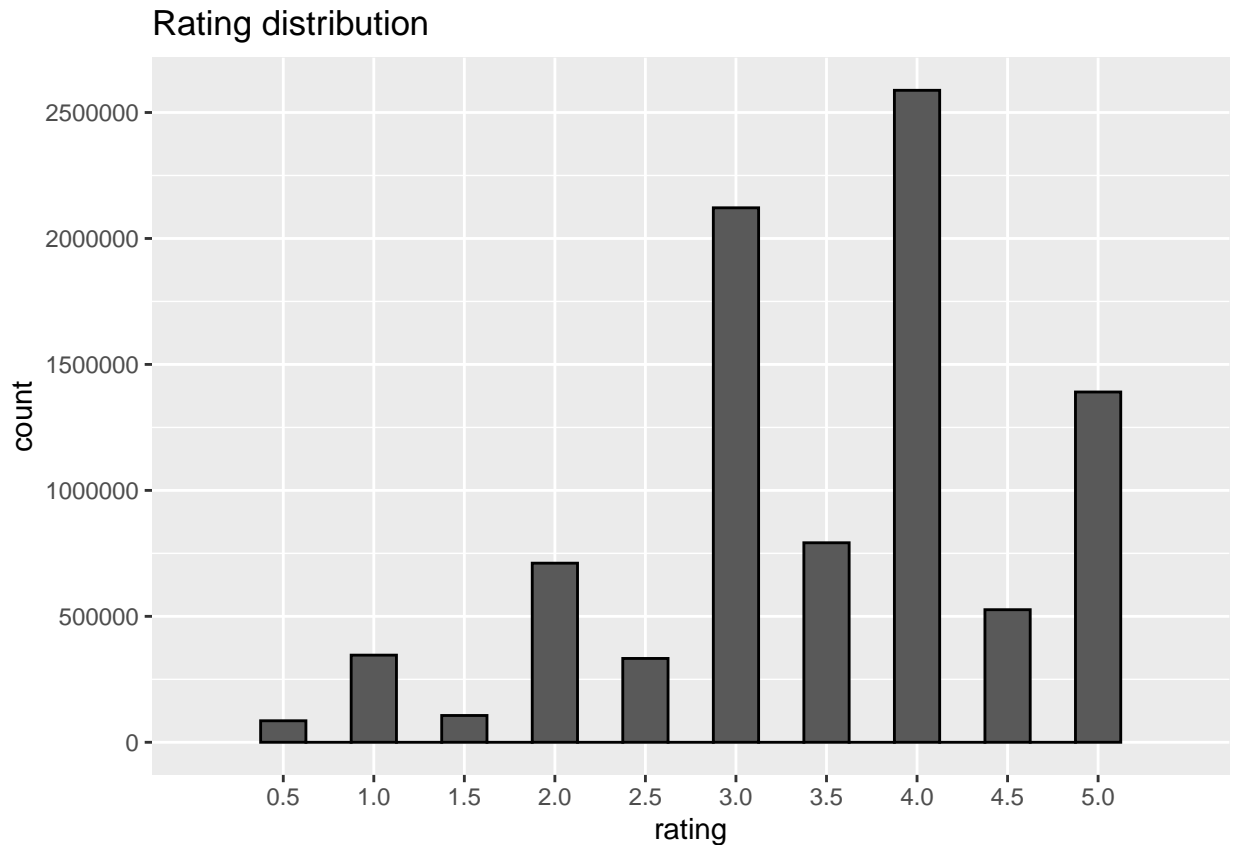
```
##      userID      movieId      rating      timestamp
## Min.   :      1  Min.   :      1  Min.   :0.500  Min.   :7.897e+08
## 1st Qu.:18122  1st Qu.:   648  1st Qu.:3.000  1st Qu.:9.468e+08
## Median :35743  Median :  1834  Median :4.000  Median :1.035e+09
## Mean   :35869  Mean   :  4120  Mean   :3.512  Mean   :1.033e+09
## 3rd Qu.:53602  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:9000061  Length:9000061
## Class :character Class :character
## Mode  :character Mode  :character
##
##
##
```

The total of unique movies and users in the edx subset is about 70.000 unique users and about 10.700 different movies:

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

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.

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean 'limits = factor(...)' or 'scale_*_continuous()'?
```

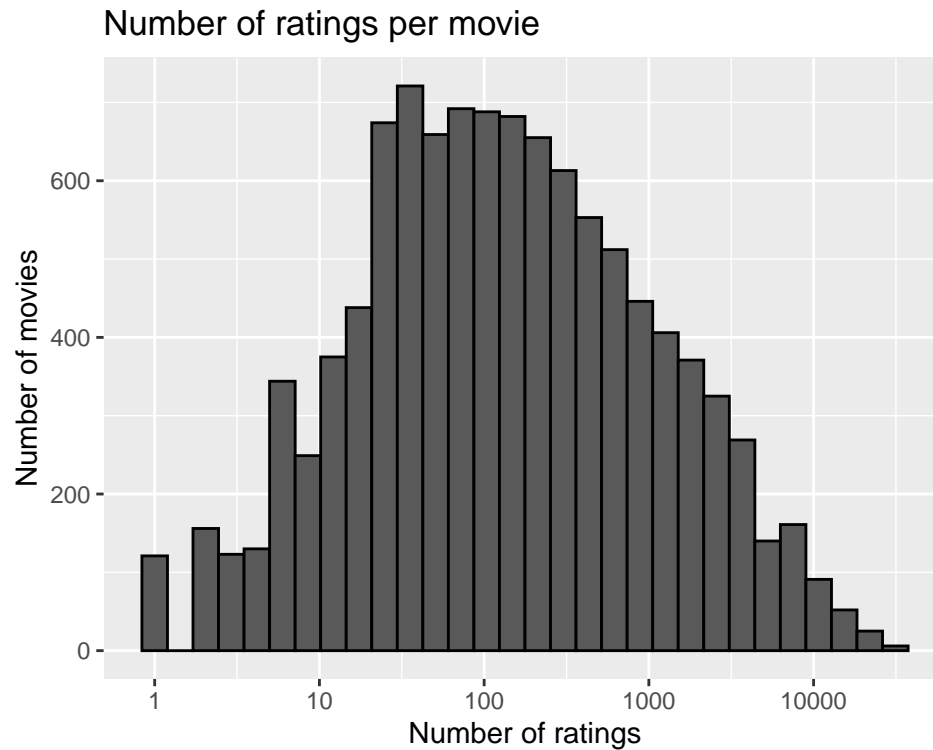


We can observe that some movies have been rated much more often than others, while some have very few ratings and sometimes only one rating. This will be important for our model as very low rating numbers might result in an untrustworthy estimate for our predictions. In fact, 125 movies have been rated only once.

Thus, regularization and a penalty term will be applied to the models in this project. Regularizations are techniques 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.

```
edx %>%
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.

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

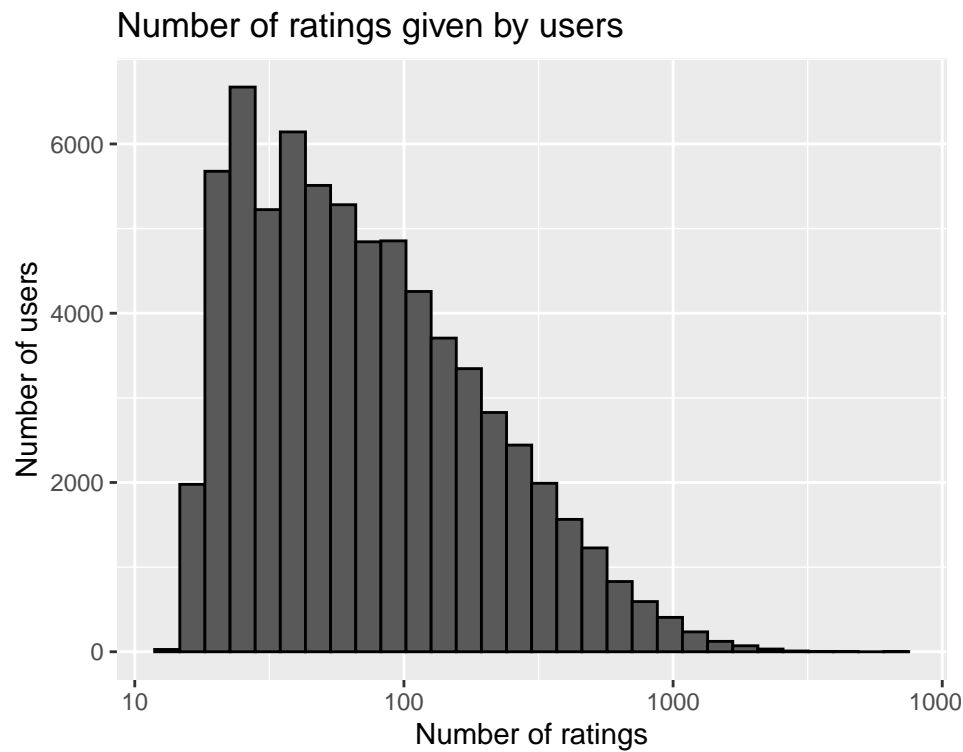
'summarise()' has grouped output by 'title'. You can override using the
'.groups' argument.

title	rating	n_rating
NA	5.0	1
NA	3.0	1
NA	3.0	1
NA	1.0	1
NA	3.0	1
NA	1.0	1
NA	1.0	1
NA	1.0	1
NA	2.0	1
NA	1.5	1
NA	2.0	1

title	rating	n_rating
NA	1.5	1
NA	1.0	1
NA	3.0	1
NA	3.0	1
NA	3.0	1
NA	2.5	1
NA	2.5	1
NA	3.0	1
NA	4.5	1
NA	2.5	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.

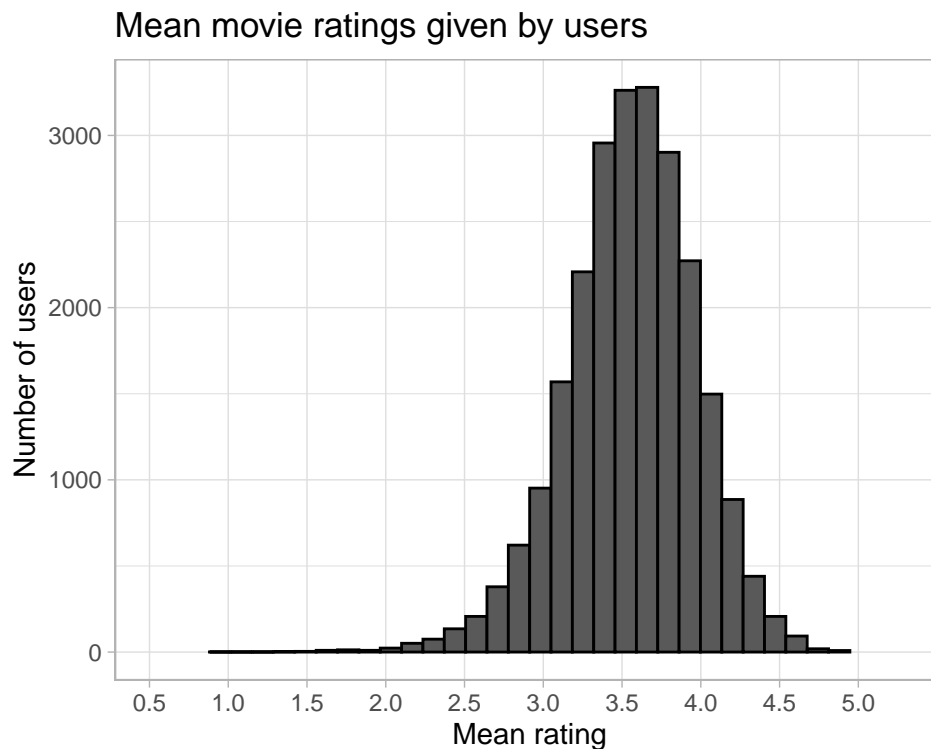
```
edx %>%
count(userId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Number of ratings") +
ylab("Number of users") +
ggtitle("Number of ratings given by users")
```



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.

```
edx %>%
  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()
```

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean 'limits = factor(...)' or 'scale*_continuous()'?
```



2.2 Modelling Approach

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

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

Letting NN represent the count of user/movie combinations and summing across all such combinations, the RMSE serves as our metric for assessing model accuracy. We can liken the RMSE to a standard deviation, signifying the usual error in our predictions of movie ratings. If the outcome exceeds 1, it indicates that our typical error surpasses one star, which is deemed undesirable. The formula for computing the RMSE for vectors of ratings and their corresponding predictions is expressed as follows:


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

The lower the better, as said previously.

2.2.1 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 minimize 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.

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

```
## [1] 3.512464
```

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] 1.060651
```

Here, we represent results table with the first RMSE:

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

```
## Warning: 'data_frame()' was deprecated in tibble 1.1.0.
## i Please use 'tibble()' instead.
```

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

method	RMSE
Average movie rating model	1.060651

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.

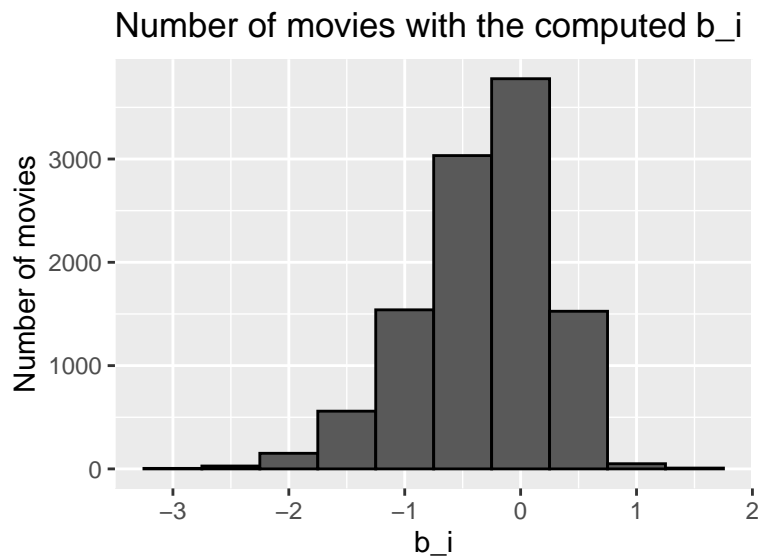
2.2.2 Movie effect model

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

```
movie_avgs <- edx %>%  
  group_by(movieId) %>%  
  summarize(b_i = mean(rating - mu))  
movie_avgs %>% qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("black"),  
  ylab = "Number of movies", main = "Number of movies with the computed b_i")
```



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	1.0606506

method	RMSE
Movie effect model	0.9437046

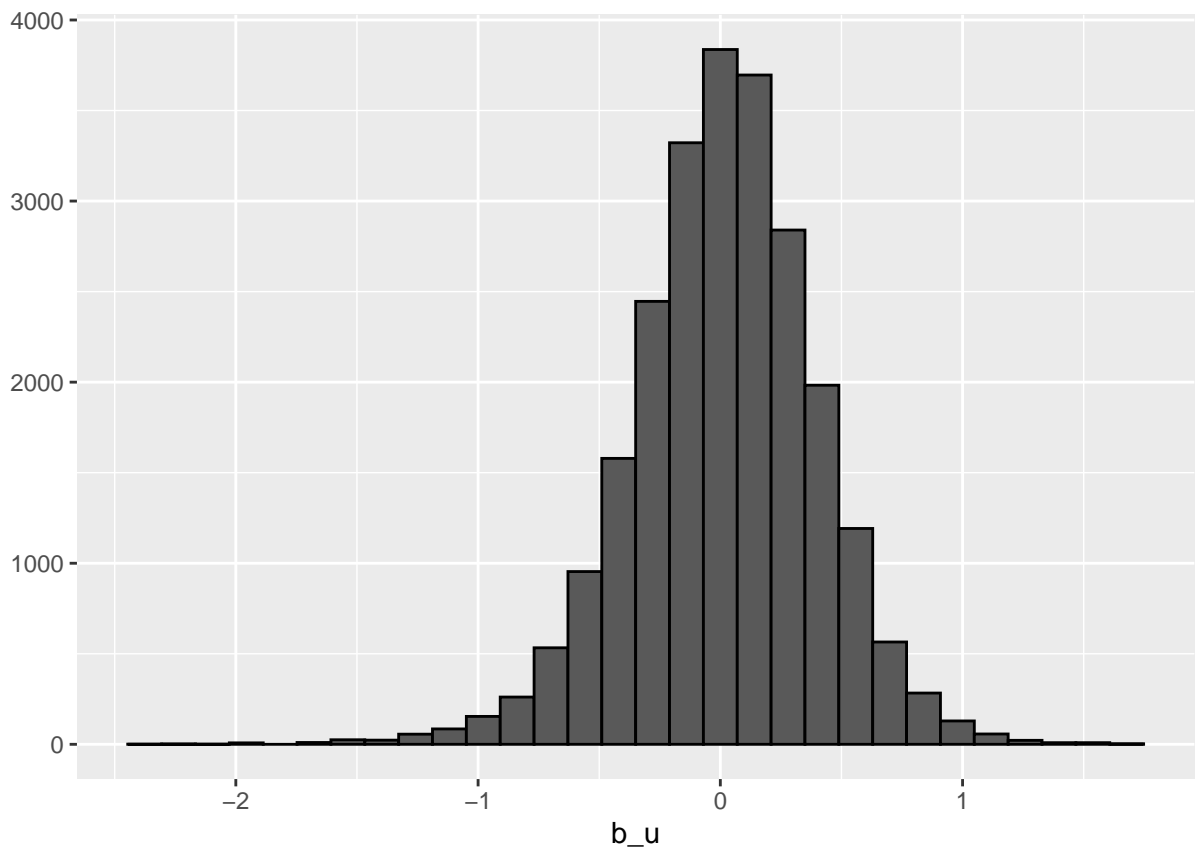
So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_i to μ . If an individual movie is on average rated worse than the average rating of all movies μ , we predict that it will be rated lower than μ by b_i , the difference of the individual movie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

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

2.2.3 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.

```
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("black"))
```



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 may 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

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

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

We can now construct predictors and see RMSE improves:

```
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	1.0606506
Movie effect model	0.9437046
Movie and user effect model	0.8655329

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.

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.

2.2.4 Regularized movie and user effect model

So estimates of b_i and b_u are caused by movies with very few ratings and in some users that only rated a very small number of movies. Hence this can strongly influence the prediction.

The use of the regularization permits to penalize these aspects. We should find the value of λ (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i and b_u in case of small number of ratings.

```
lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+1))

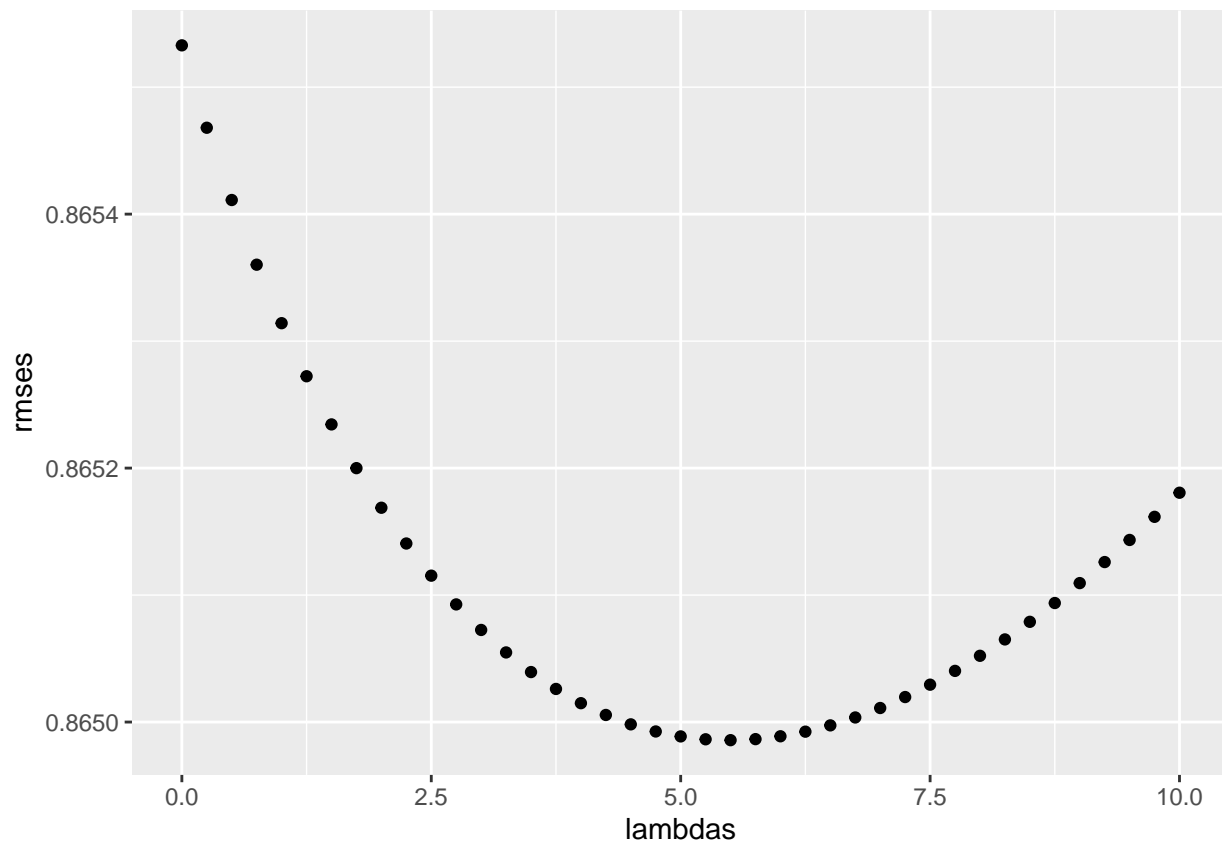
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+1))

  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))
})
```

We plot RMSE vs lambdas to select the optimal lambda

```
qplot(lambdas, rmsees)
```



For the full model, the optimal lambda is:

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

```
## [1] 5.5
```

For the full model, the optimal lambda is: 5.25

The new results will be:

```
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	1.0606506
Movie effect model	0.9437046
Movie and user effect model	0.8655329
Regularized movie and user effect model	0.8649857

3 Chapter 3

3.1 Results

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

method	RMSE
Average movie rating model	1.0606506
Movie effect model	0.9437046
Movie and user effect model	0.8655329
Regularized movie and user effect model	0.8649857

We therefore found the lowest value of RMSE that is 0.8649857.

4 Chapter 4

4.1 Discussion

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

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

In this model,

$$Y_{u,i}$$

represents the predicted rating for user

$$u$$

on item

$$i$$

. The components of the model are explained as follows: 1.

$$\mu$$

is the overall average rating across all users and items.

2.

$$b_i$$

is the deviation in the rating for item i from the overall average, capturing its individual effect.

3.

$$b_u$$

is the deviation in the rating behavior of user u from the overall average, accounting for user-specific preferences.

4.

$$\epsilon_{u,i}$$

denotes the error term, representing any random or unaccounted factors affecting the rating.

This model is effective under the assumption that the average user tends not to rate a particularly good or popular movie with an excessively large positive

$$b_i$$

value, indicating a strong liking for a specific movie. The model accommodates the nuances of user preferences and item characteristics, striving to provide accurate predictions for user-item ratings based on deviations from the overall average.

5 Chapter 4

5.1 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 (0.8649857) lower than the initial evaluation criteria (0.86490) given by the goal of the present project. Improvements in the RMSE could be achieved by adding other effect (genre, year, age,..). Other different machine learning models could also improve the results further, but hardware limitations, as the RAM, are a constraint.

In conclusion, this project aimed to develop a machine learning algorithm for predicting movie ratings using the MovieLens dataset. The final model, a regularized movie and user effect model, achieved the lowest Root Mean Square Error (RMSE) of 0.8649857, surpassing the initial evaluation criteria of 0.86490. This model considers both movie and user effects, providing a more accurate prediction of movie ratings.

5.2 Limitations

Despite the success of the model, there are limitations to consider. The dataset's performance was influenced by movies with few ratings, leading to larger estimates that could impact the RMSE. Additionally, hardware limitations, such as RAM constraints, restricted the exploration of more complex machine learning models.

5.3 Future Work

Future work could involve further enhancing the model by incorporating additional features such as genre, year, and age. Exploring different machine learning models beyond the scope of this project may also yield improvements. Addressing hardware limitations to enable the exploration of more sophisticated algorithms and larger datasets could contribute to refining the predictive accuracy of movie ratings. Additionally, evaluating the model's performance on diverse datasets or extending the analysis to different domains could provide valuable insights. Continuous refinement and optimization are crucial for enhancing the robustness and applicability of the movie rating prediction algorithm.

6 Reference