Movielens Project

Derrick Owusu Ofori

11/01/2021

Contents

Introduction

A recommendation system is a class of machine learning algorithm that offers useful and relevant suggestions to users. A recommendation system has applications in several fields, and in this project, we will explore how recommender systems can be applied to movies. We will employ the MovieLens 10M dataset complied by the Grouplens Research lab found here; https://grouplens.org/datasets/movielens/10m/ and http://files.grouplens.org/datasets/movielens/ml-10m.zip. This dataset contains roughly ten million observations and six variables; userId, movieId, rating, timestamp, title and genres.

In this project, we will utilize a subset of the dataset to build an algorithm that can make predictions about ratings in a validation set. We will start by importing data, performing some exploratory data analysis and then compare five models to determine the one with the best Root Mean Square Error (RMSE).

Methods

Installing packages and Importing data

```r
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(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")

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

```r
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")
```

The dataset will now be split into a train set (edx) which we will use to train, develop and select our algorithm and a test set (validation) to test the performance of the final algorithm. The test set will be 10% of the dataset.

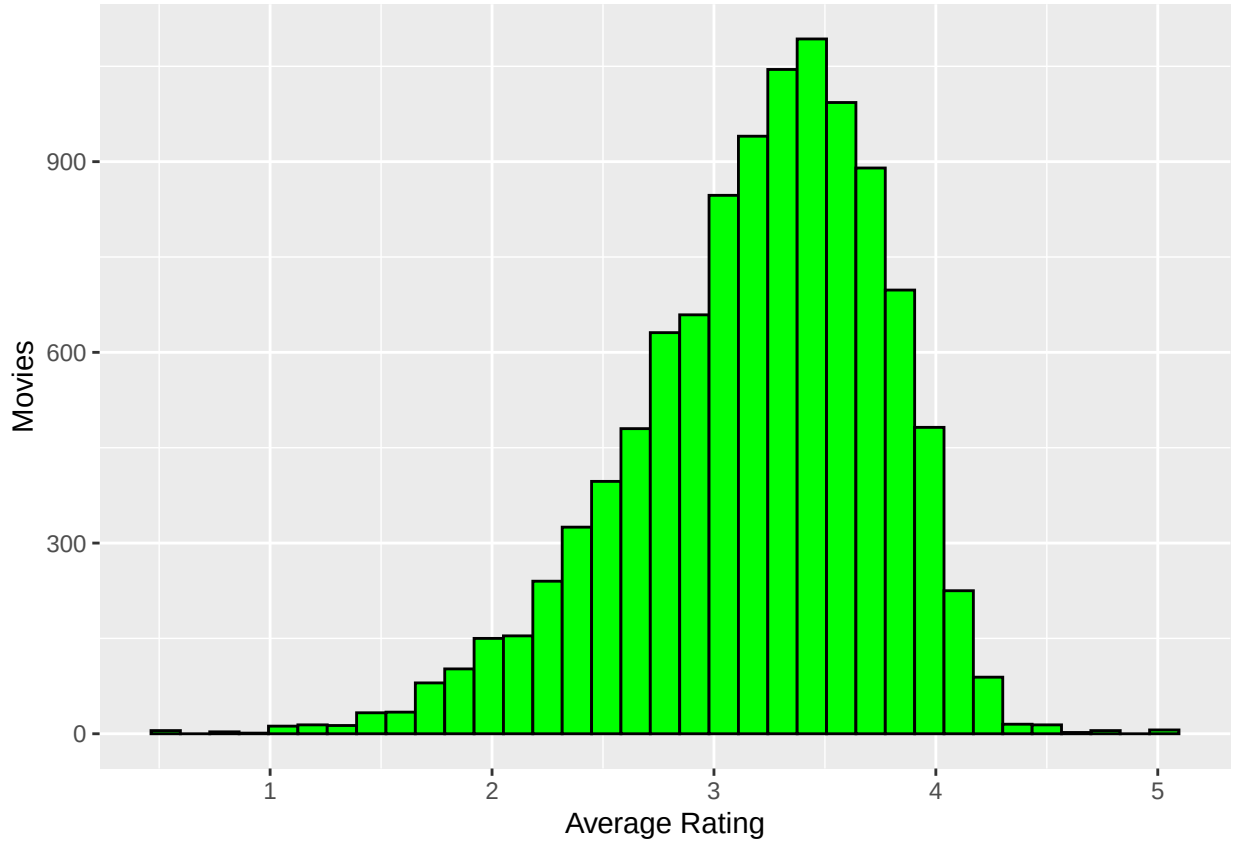
```r
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# 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)
```

Data exploration and visualization

Average movie rating

The figure below indicates that certain movies tend to have higher ratings on average while others have lower ratings on average.

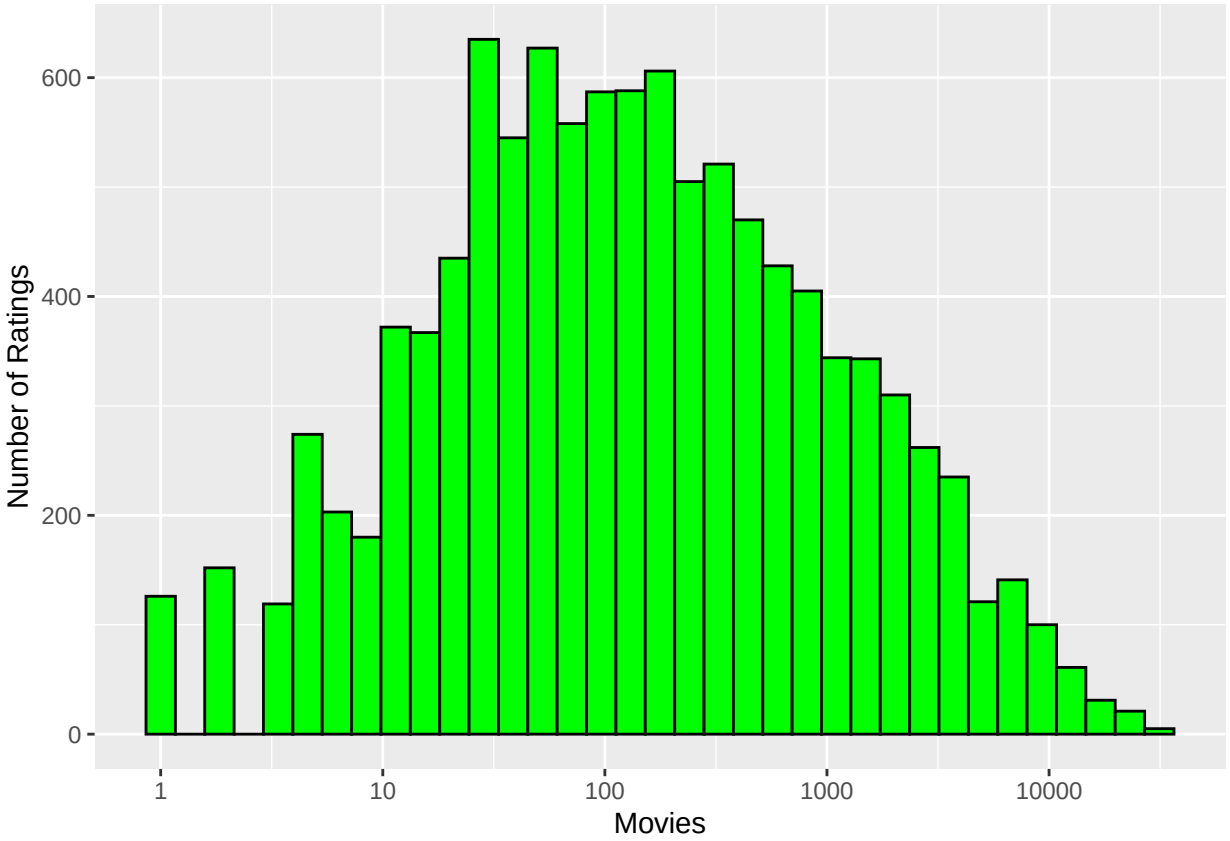
```
# Average movie rating
edx %>% group_by(movieId) %>%
  summarise(avg_rating = sum(rating)/n()) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(bins=35, color ="black", fill="green") +
  labs(x = "Average Rating", y = "Movies")
```



Number of ratings per movie

This figure illustrates that some movies have a high number of ratings while others have a low number of ratings.

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=35, color = "black", fill="green") +
  scale_x_log10() + labs(x = "Movies", y = "Number of Ratings")
```

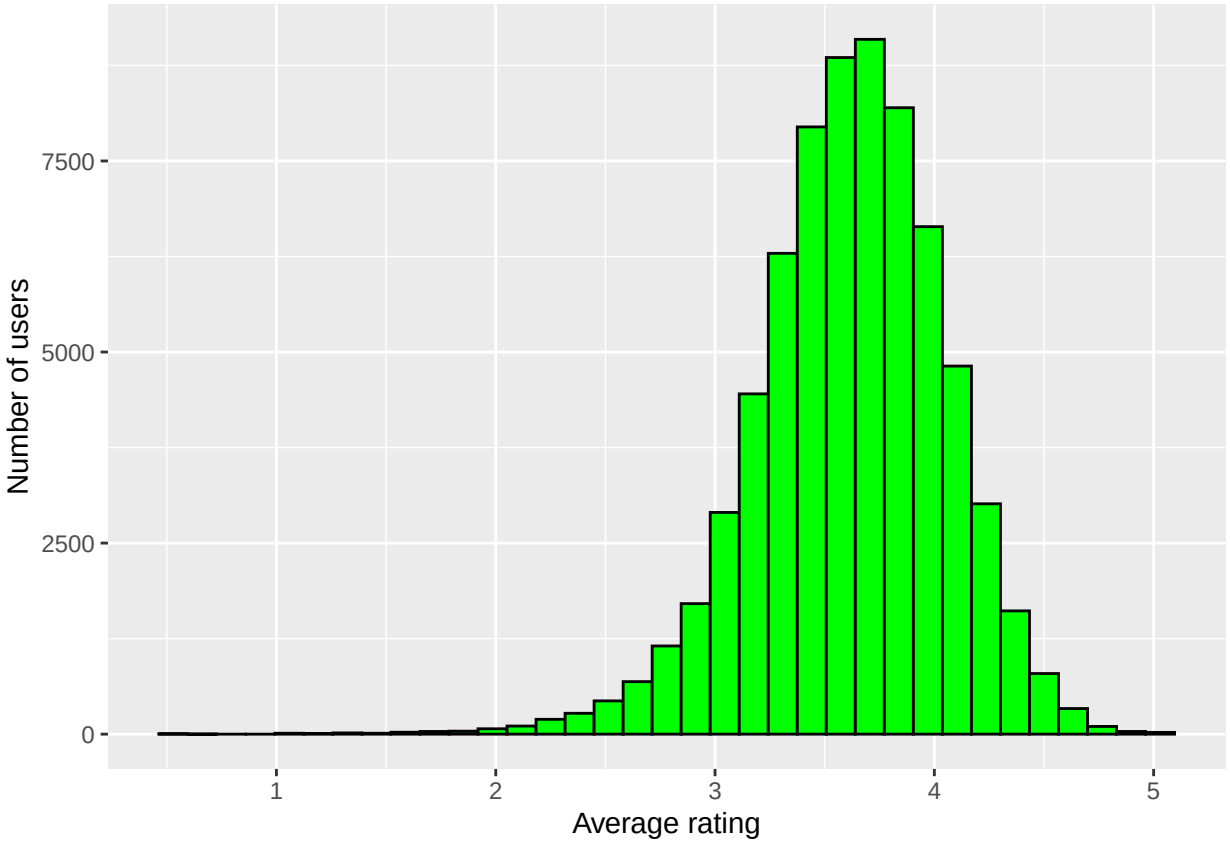


These patterns indicate a movie effect which will be accounted for during the final algorithm.

Average rating per user

We see from this figure that certain users give higher ratings on average than others.

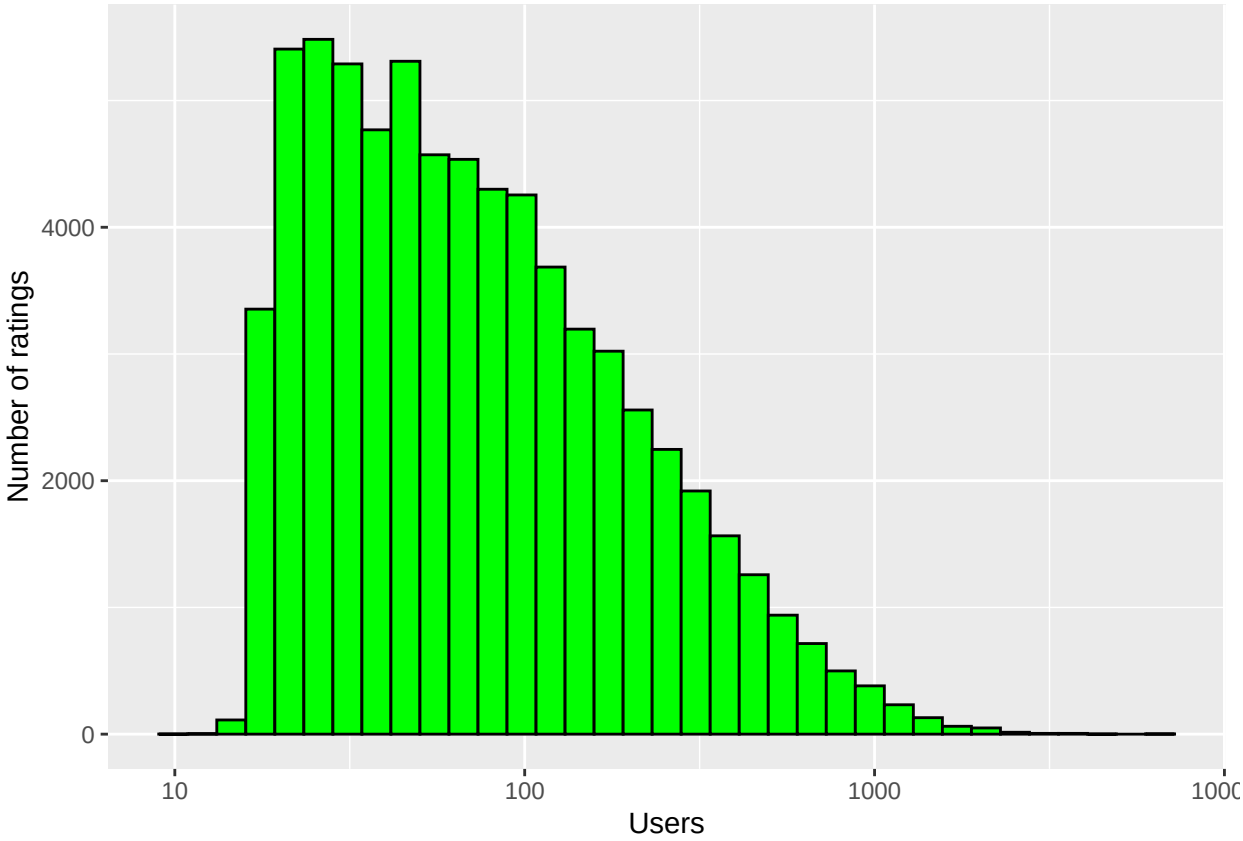
```
edx %>% group_by(userId) %>%
  summarise(avg_rating = sum(rating)/n()) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(bins=35, color = "black", fill="green") +
  labs(x = "Average rating", y = "Number of users")
```



Number of ratings per user

Similar to the number of ratings per movie, some users also tend to rate movies a lot more times than others.

```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=35, color = "black", fill="green") +
  scale_x_log10() +
  labs(x = "Users", y = "Number of ratings")
```



These figures demonstrate a user effect which we will adjust for in developing our model.

Modeling approach

The criteria to evaluate model performance will be the Root Mean Square Error (RMSE). The RMSE is the square root of the mean of the square of all the errors. It can be calculated by squaring the residuals, finding the average of the residuals and taking the square root of the result. It measures the difference between the predicted and actual values, with a lower RMSE value being better than a higher value.

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

1. Average Movie Rating Model

Our first model will be a simple recommendation system. A model-based approach will be utilized, which assumes the same rating for all movies and all users, with all differences explained by random variation.

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

Here $\epsilon_{u,i}$ represents an independent error sample from the same distribution centered at 0 and μ represents the true rating for all movies. The estimate that minimizes the residual mean squared error is the least-squares estimate of $Y_{u,i}$ which is the average of all the ratings.

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

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

With a mean rating of approximately 3.51, we will move forth and compute the naive RMSE

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

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

Now we will tabulate the first RMSE results for later comparison with other models.

```
rmse_results <- data_frame(Method  = "Average Movie Rating",
                           RMSE = naive_rmse)
```

2. Movie Effect Model

Some movies tend to be rated higher than others as we will see in the diagram below, thus we can improve our previous model by including a term, b_i to represent the average rating for each movie i.

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

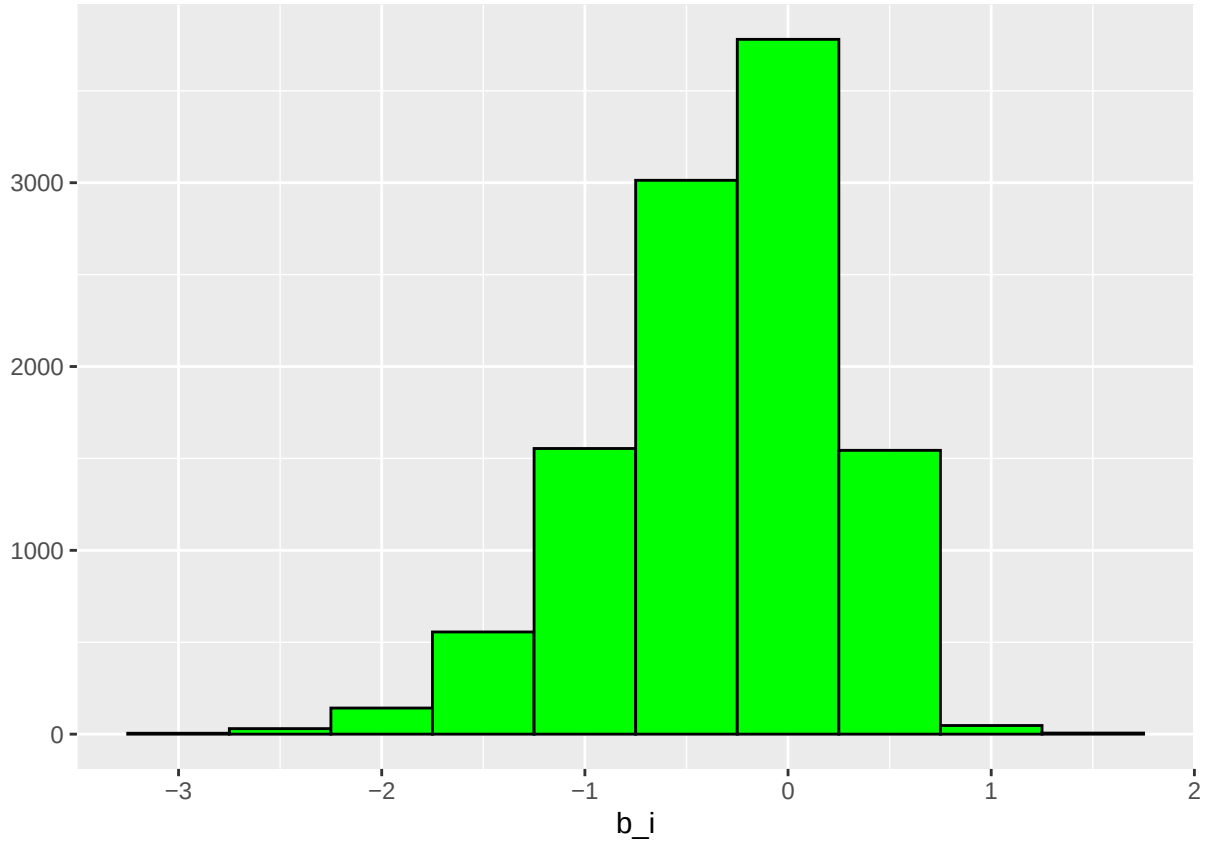
The least squares estimate is the average of the

$$Y_{u,i}$$

minus the overall mean for each movie i. This can be computed with:

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

```
movie_avgs %>% qplot(b_i, geom ="histogram",
                     bins = 10, data = .,
                     color = I("black"),
                     fill=I("green"))
```



As mentioned earlier, the estimates seem to vary substantially, this is because some movies are good and others are bad resulting in different average ratings.

Now let's make predictions with the inclusion of the movie effect.

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

We will now tabulate this result with previous RMSE results for comparison

```
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method ="Movie effect model",
                                     RMSE = model_1_rmse ))
```

3. Movie and user effect model

Next, we have to consider how users rate movies. As illustrated in the histogram below, we can see variability across users, with some users rating most of the movies they watch highly while others do not. Therefore our model will need to be augmented with a new term b_u which is the user-specific effect. This makes our new model:

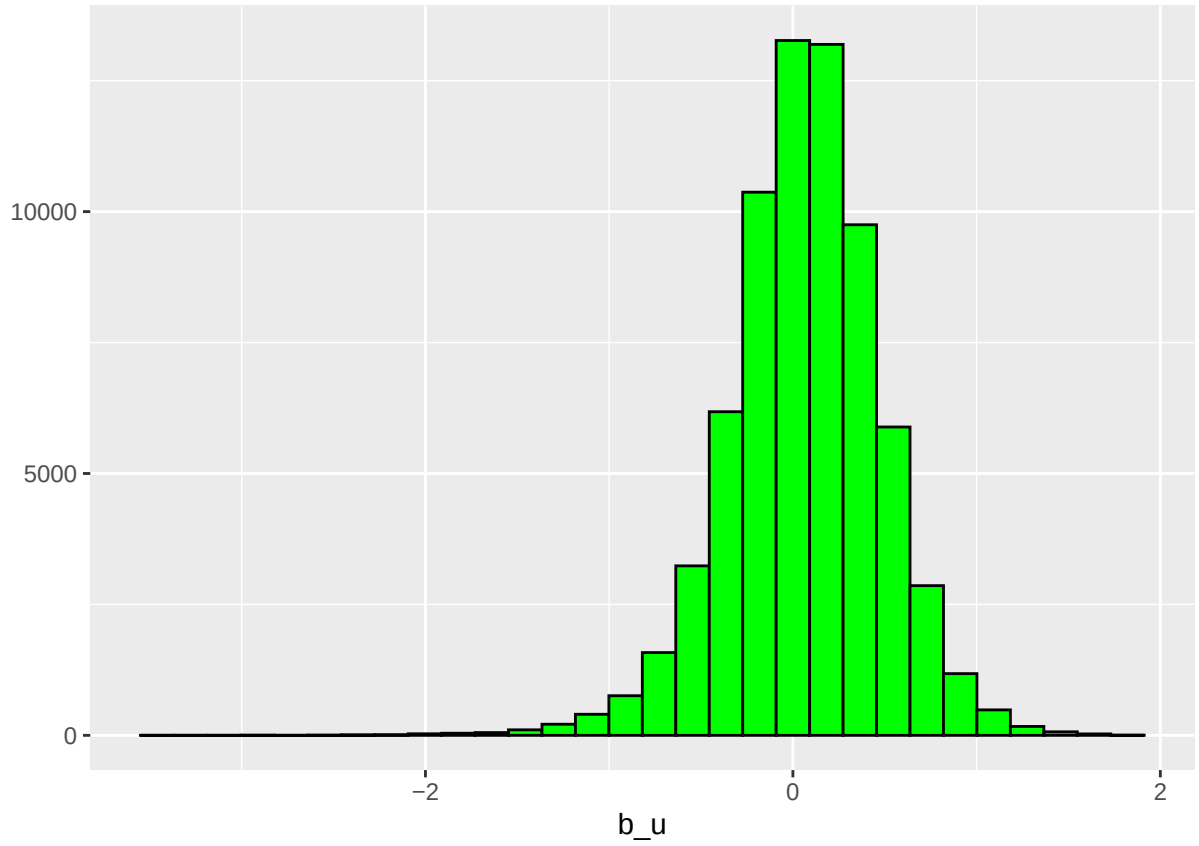
$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

We will compute 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))
```

```
user_avgs%>% qplot(b_u, geom ="histogram",
                   bins = 30, data = .,
                   color = I("black"),
                   fill=I("green"))
```



We will now make predictions with the user effect captured.

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

We will save this result with previous RMSE results for comparison

```
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))
```

4. Regularizing movie and user effect

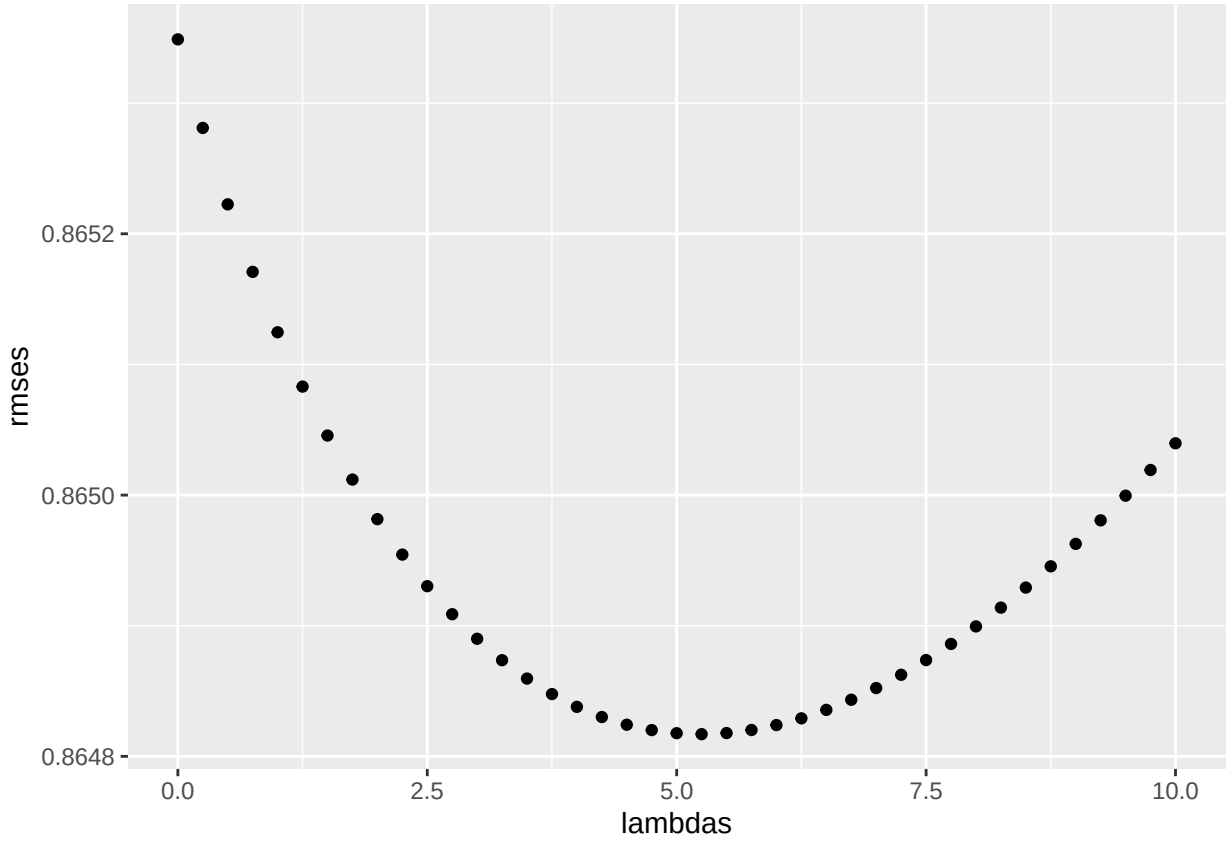
In our previous models, a grievous error was overlooked. Some obscure movies were rated by a few number of people leading to more uncertainty, and resulting in larger estimates for b_i and b_u. These are noisy estimates that result in large errors and should not be trusted. Normally, we compute standard errors and construct 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 require the concept of regularization. Regularization permits us to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i and b_u to the sum of squares equation that we minimize. To do this, we require a tuning parameter (lambda) that will minimize the RMSE. This will therefore minimize the estimates of b_i and b_u if the number of ratings is small.

For the penalty term, we will use numbers from 0 to 10 with increments of 0.25

```
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) %>%
    .$pred

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

Next we will plot rmses vs lambdas to select the optimal lambda

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




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

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

Here the optimal lambda is 5.25

Now we will tabulate and store the regularized result.

```
rmse_results <- bind_rows(rmse_results,
data_frame(Method ="Regularized movie and user effect model",
RMSE = min(rmses)))
```

5. Matrix factorization

Matrix factorization is a widely used concept in machine learning. It is similar to factor analysis, single value composition, and principal component analysis. It works by decomposing the user-item interaction matrix into a product of two lower dimensionality rectangular matrices. We will apply this approach to our recommender system by utilizing the recosystem library.

First, both the training and validation set will need to be organized into three columns: user (userId), item (movieId) and the value (rating).

```
edx_fc <- edx %>% select(movieId, userId, rating)
validation_fc <- validation %>% select(movieId, userId, rating)
```

We will transform them into a matrix format.

```
edx_fc <- as.matrix(edx_fc)
validation_fc <- as.matrix(validation_fc)
```

Next, we will write these datasets onto the hard disk as tables and assign them to a train set (train_fc) and a validation set(valid_fc).

```
write.table(edx_fc, file = "trainingset.txt", sep = " ", row.names = FALSE,
            col.names = FALSE)

write.table(validation_fc, file = "validationset.txt", sep = " ",
            row.names = FALSE, col.names = FALSE)
```

A supported data format will be utilized by calling the data_file() function.

```
set.seed(76)
train_fc <- data_file("trainingset.txt")

valid_fc <- data_file("validationset.txt")
```

We will now build a recommender object (r) using the Reco() in the recosystem package. We utilize the $tune() approach to find the optimum tuning parameter.

```
r = Reco()

opts <- r$tune(train_fc, opts = list(dim = c(5, 10, 15), lrate =
                                       c(0.1,0.2), costp_l1 = 0,
                                       costq_l1 = 0, nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 15
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.1
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.01
##
## $min$lrate
## [1] 0.2
##
## $min$loss_fun
## [1] 0.8062513
```

```
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1    5        0     0.01        0     0.01   0.1 0.8401233
## 2   10        0     0.01        0     0.01   0.1 0.8247991
## 3   15        0     0.01        0     0.01   0.1 0.8092381
## 4    5        0     0.10        0     0.01   0.1 0.8581199
## 5   10        0     0.10        0     0.01   0.1 0.8304450
## 6   15        0     0.10        0     0.01   0.1 0.8115444
## 7    5        0     0.01        0     0.10   0.1 0.8435519
## 8   10        0     0.01        0     0.10   0.1 0.8274432
## 9   15        0     0.01        0     0.10   0.1 0.8085934
## 10   5        0     0.10        0     0.10   0.1 0.8707336
## 11  10        0     0.10        0     0.10   0.1 0.8382013
## 12  15        0     0.10        0     0.10   0.1 0.8326031
## 13   5        0     0.01        0     0.01   0.2 0.8225618
## 14  10        0     0.01        0     0.01   0.2 0.8152214
## 15  15        0     0.01        0     0.01   0.2 0.8629989
## 16   5        0     0.10        0     0.01   0.2 0.8302261
## 17  10        0     0.10        0     0.01   0.2 0.8203236
## 18  15        0     0.10        0     0.01   0.2 0.8062513
## 19   5        0     0.01        0     0.10   0.2 0.8236409
## 20  10        0     0.01        0     0.10   0.2 0.8162198
## 21  15        0     0.01        0     0.10   0.2 0.8117151
## 22   5        0     0.10        0     0.10   0.2 0.8477272
## 23  10        0     0.10        0     0.10   0.2 0.8410712
## 24  15        0     0.10        0     0.10   0.2 0.8277425
```

The recommender model will now be trained with $train().

```
r$train(train_fc, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9520   1.0740e+07
##    1       0.8633   9.1871e+06
##    2       0.8249   8.5748e+06
##    3       0.8039   8.2711e+06
##    4       0.7909   8.0899e+06
##    5       0.7820   7.9669e+06
##    6       0.7760   7.8885e+06
##    7       0.7717   7.8301e+06
##    8       0.7684   7.7848e+06
##    9       0.7657   7.7472e+06
##   10       0.7635   7.7226e+06
##   11       0.7617   7.6987e+06
##   12       0.7600   7.6796e+06
##   13       0.7586   7.6626e+06
##   14       0.7572   7.6477e+06
##   15       0.7559   7.6344e+06
##   16       0.7547   7.6234e+06
##   17       0.7536   7.6107e+06
##   18       0.7523   7.5986e+06
##   19       0.7512   7.5909e+06
```

Next, we will write the predictions to a temp file on the hard drive, make predictions with the validation set and calculate the RMSE.

```
saved_pred = tempfile()

r$predict(valid_fc, out_file(saved_pred))
```

```
## prediction output generated at C:\Users\user\AppData\Local\Temp\Rtmpob6eqn\file2bb44d85725f
```

```
actual_ratings <- read.table("validationset.txt", header = FALSE, sep = " ")$V3
predicted_ratings <- scan(saved_pred)

rmse_fc <- RMSE(actual_ratings, predicted_ratings)
rmse_fc
```

```
## [1] 0.7939213
```

Finally, we will save and tabulate the RMSE for comparison with previous models.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method ="Matrix factorization", RMSE = rmse_fc ))
```

Results

As illustrated below, we utilized five models for our recommender system with differing RMSE. In our first model (Average Movie Rating), we used the mean rating to make predictions. This resulted in the highest RMSE of approximately **1.0612**. Next, we adjusted the model to account for the movie effect which produced an RMSE of about **0.9439**. We further augmented the model by capturing the user effect which gave us a better RMSE of approximately **0.8653**. Afterwards, we regularized the movie and user effect model to get rid of noisy estimates affecting our model. This led to a slightly better RMSE of about **0.8648**. Finally, by applying matrix factorization we achieved the best RMSE of **0.7939213**.

Method	RMSE
Average Movie Rating	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Regularized movie and user effect model	0.8648170
Matrix factorization	0.7939213

Conclusion

Summary

The goal of this project was to develop a movie recommender system with the MovieLens 10M dataset. After some exploratory data analysis, we realized patterns in our movie and user data that influenced the creation of our models. These were the average movie rating model, the movie effect model, the movie and user effect model, the regularized movie and user effect model and finally matrix factorization which produced

the lowest RMSE of about **0.7939**.

Limitation

There were severe CPU and memory limitations, curtailing the utilization of powerful techniques which may have resulted in better model performance.

Future work

For my future work, I would like to explore how the genre effect affects model performance. The ensemble method which combines several models would also be looked into and its impact on model performance measured.