A MovieLens Recommendation System

Emilia Iacob

February 7, 2023

Contents

Introduction

This is a report on the steps taken to build a recommendation system based on the MovieLens dataset available on the *grouplens* website.

This dataset consists of the ratings given by different users to various movies. Our objective will be to develop a prediction algorithm capable of predicting the rating that a certain user will give to a particular movie.

The method that we'll use to judge the accuracy of our prediction algorithm will be the residual mean squared error (RMSE) that will be applied on a test set - a dataset the we have not used when building our algorithm but for which we already know the actual movie ratings from different users. The RMSE will show us the typical error we make when predicting a movie rating. Our objective is to reach a RMSE lower than 0.899, the lower the better.

The approach to build this recommendation system takes into account several steps:
- first we start with the average rating of all movies and consider this as the baseline.
- further we take into account the fact that some movies are better than others and assume a movie effect which we add to the baseline.
- next we observe that the same movie gets rated differently by different users so we assume a user effect that we will add to the movie effect and the baseline.
- then we notice that some movies don't have many ratings which increases our chances of error in predicting the rating for those movies. Therefore, we will use regularization which assigns a weight/penalty number to minimize this error when the number of ratings for a movie is low. We use cross validation to identify the best candidate for this penalty number.
- we also observe that certain genres get rated consistently lower than other genres which indicates that there might also be a genre effect that we could take into account.
- in the end, we will combine all the effects above (i.e. baseline + movie effect + regularization + user effect + genre effect) to develop a final model and test the predictions of this model on the final holdout test dataset.

Important note for the evaluators of this project:

Unlike the R script from the *MovieLens-Capstone-Emilia.R* file which builds and evaluates the different models based on the *entire edx dataset*, **the R code below only uses a sample of the edx dataset in order to allow for the report to render in pdf**. The first attempts of passing the entire edx dataset consistently gave the error message: *"Error: cannot allocate vector of size 5.6 Gb."* Therefore, I have decreased the size of the edx dataset to a more manageable size of 1.6 Gb by only selecting the movies that have been rated more than 100 times and the users that have rated at least 50 movies.

Analysis

Step 1: Collecting and preparing the data

We will use the 10M MovieLens data set which is available here:
https://grouplens.org/datasets/movielens/10m/

and here:
http://files.grouplens.org/datasets/movielens/ml-10m.zip

Let's first call the packages that we are going to use in this analysis:

```
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")

library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(knitr)
```

Now we can start downloading the file like this:

```
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

Let's create the *ratings* and *movies* dataframes which will be joined to create the *movielens* dataframe:

```
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

Now let's create the *edx* dataset and the *final_hold-out_test* set which will be 90% and 10% respectively of the movielens dataset:

```r
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

We also need to make sure that *userId* and *movieId* in the *final_hold-out_test* set are also in the *edx* set:

```r
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

Let's add the rows removed from *final_hold-out_test* set back into *edx* set:

```r
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)
```

We can also remove the temporary data objects that we created so we can free up space in R Studio's Global Environment.

```r
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

To save the resulting files (*movielens*, *edx* and *final_holdout_test*) for future use we'll do the following:

```r
save(movielens, file = "movielens.rda")
save(edx, file = "edx.rda")
save(final_holdout_test, file = "final_holdout_test.rda")
```

To quickly load the saved *edx* and *final_holdout_test* files and go to the next steps of this analysis we can use the load function:

```r
load("edx.rda")
load("final_holdout_test.rda")
```

Step 2: Exploring the edx dataset

Let's first do some exploratory data analysis on the edx dataset to better understand the challenge that we have ahead.

First, let's see how many observations and variables we have in our edx dataset:

```
dim(edx)
```

```
## [1] 9000055       6
```

Next, let's see the type of data we have in our edx dataset:

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
```

We notice that each row represents the rating of a user for a particular movie and we have a little over 9 million rows.

Let's see the number of unique users and the number of unique movies that are rated in the edx data set:

```
edx %>% summarize (unique_users = n_distinct(userId), unique_movies = n_distinct(movieId)) %>%
  kable()
```

unique_users	unique_movies
69878	10677

If we multiply the number of unique users by the number of unique movies we get 746,087,406 ratings. However, we just saw that we have only a little over 9 milion ratings which means that not all user-movie combinations are captured in this dataset so not all movies get rated by all users and viceversa.

Let's now see the actual number of movies rated for each of the ratings in the edx data set

- either as a table:

```
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
##   85374  345679  106426  711422  333010 2121240  791624 2588430  526736 1390114
```

- or as a histogram:

```
hist(edx$rating, main="Number of movies for each rating", xlab = "Rating")
```

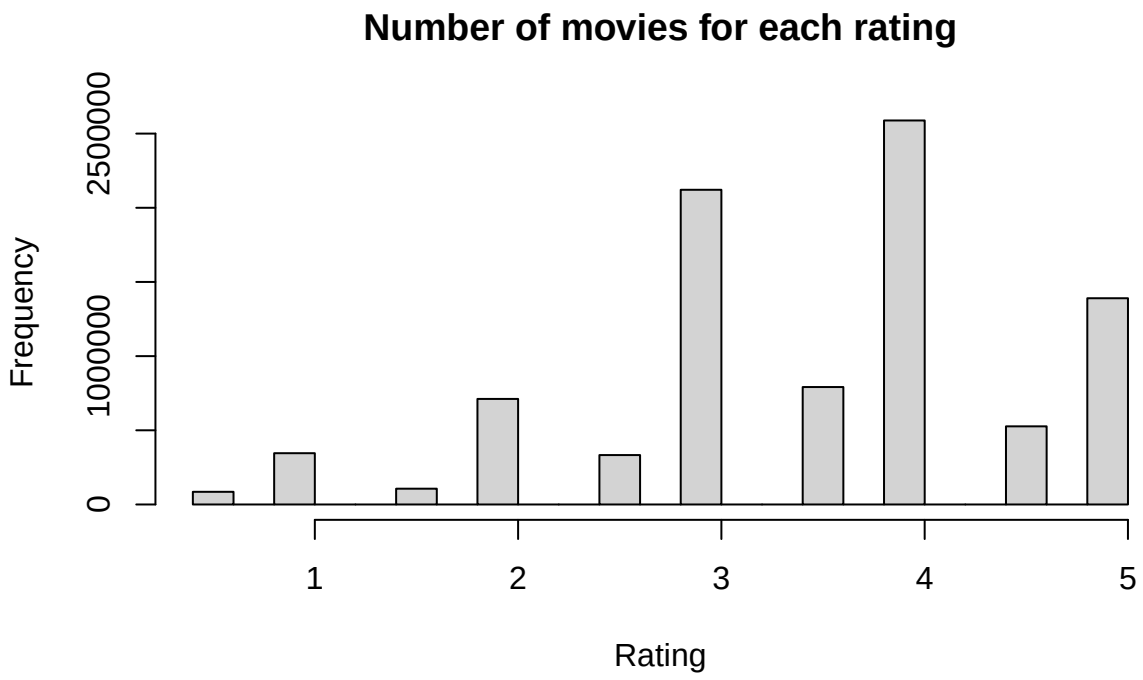

Figure 1: Distribution of movies by ratings recieved

We can see that the ratings of 3 and 4 are the most frequent.

If we plot the distribution of movies that received a rating we'll notice that some movies are rated more often than others:

```
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies with Ratings")
```

If we plot the distribution of users that have rated movies we'll notice, again, that some users give ratings to more movies than other users:

```
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users with Ratings")
```

Let's see the top movies by number of ratings:

```
top_movies <- edx %>% group_by(title) %>%  summarise(total_ratings = n()) %>%
  top_n(5)
```

```
## Selecting by total_ratings
```

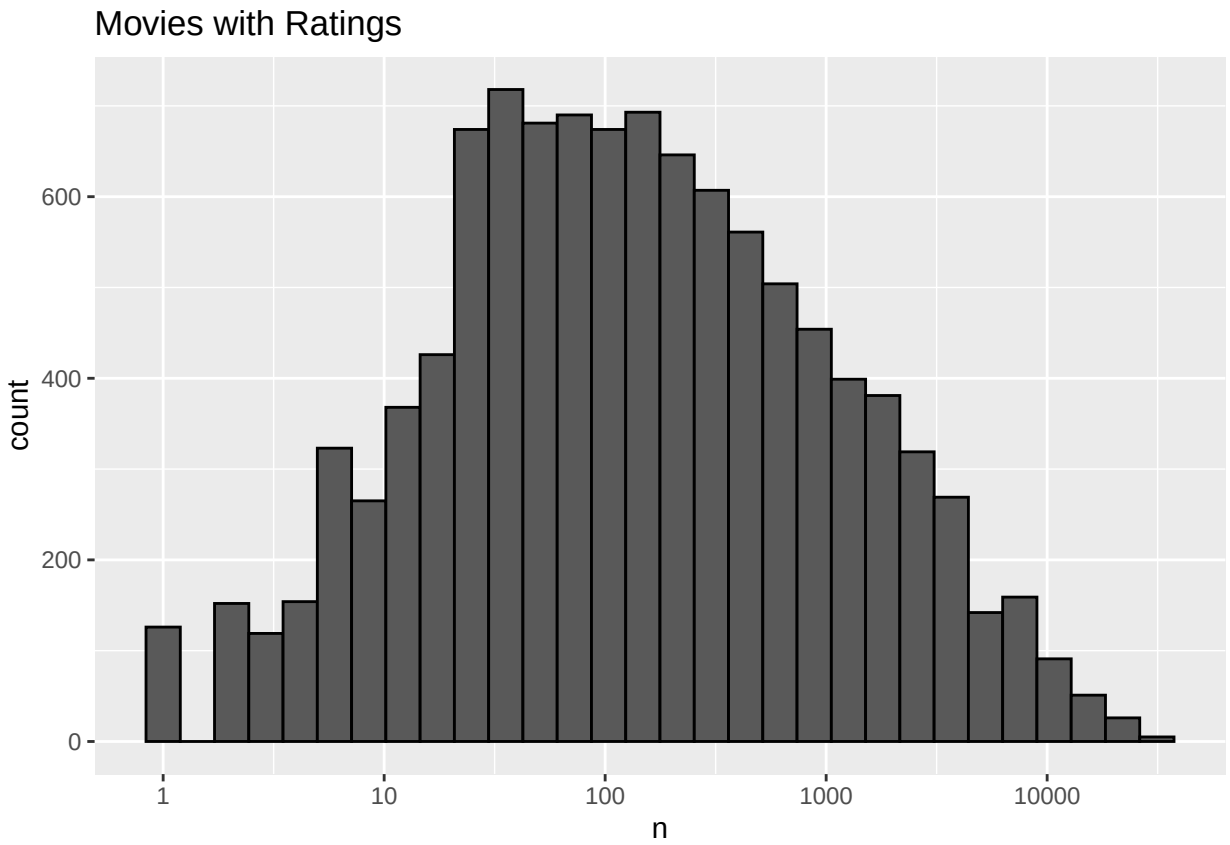

Figure 2: Distribution of movies that received a rating

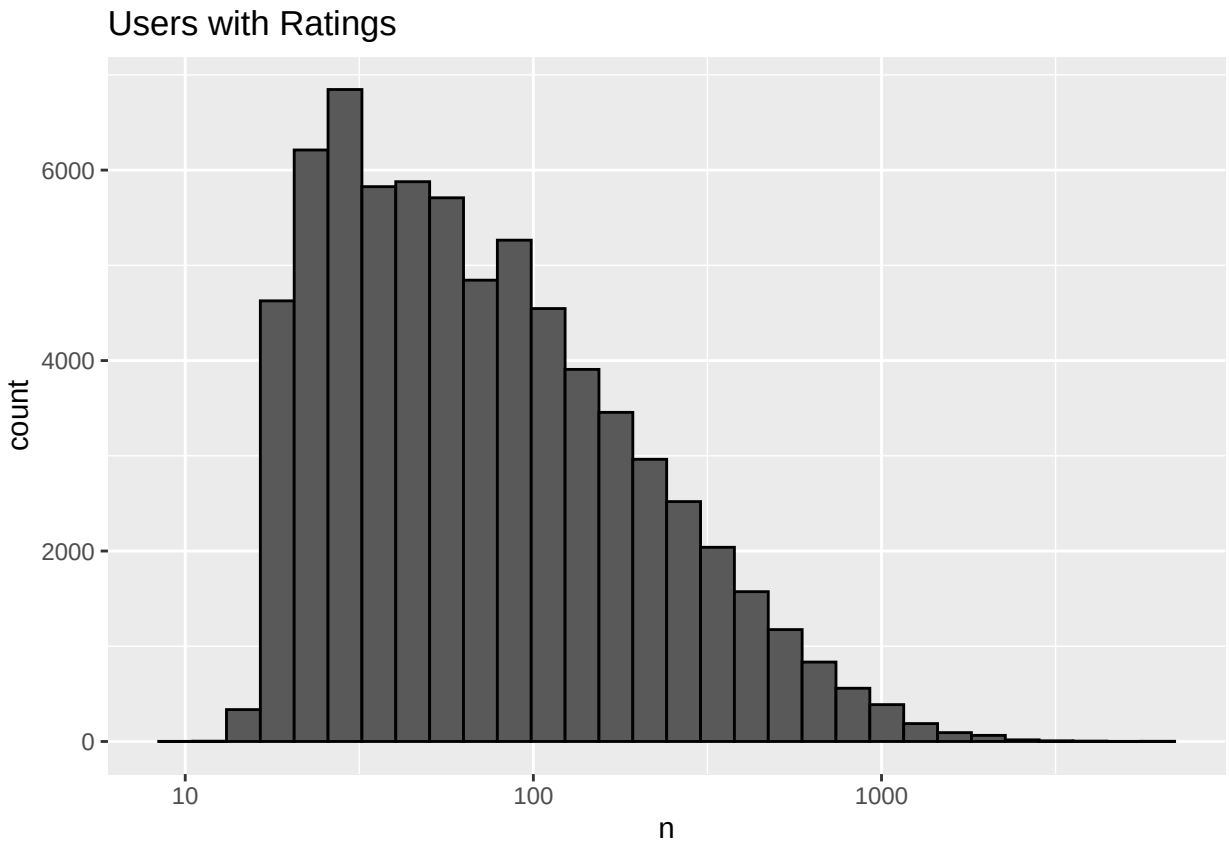

Figure 3: Distribution of users that gave a rating

```
top_movies %>% kable()
```

title	total_ratings
Forrest Gump (1994)	31079
Jurassic Park (1993)	29360
Pulp Fiction (1994)	31362
Shawshank Redemption, The (1994)	28015
Silence of the Lambs, The (1991)	30382

If we now pick a sample of users rating these top movies:

```
edx %>%
  filter(userId %in% c(12:20)) %>%
  filter(title %in% top_movies$title) %>%
  select(userId, title, rating) %>%
  spread(title, rating) %>% kable()
```

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption, The (1994)	Silence of the Lambs, The (1991)
13	NA	NA	4	NA	NA
16	NA	3	NA	NA	NA
17	NA	NA	NA	NA	5
18	NA	3	5	4.5	5
19	4	1	NA	4.0	NA

we notice that not all users rate all these top movies. This is consistent with our initial finding that not all users rate all movies. In fact, to see how sparse the ratings are among users let's build a matrix with a random sample of 100 movies and 100 users with yellow indicating the rating given by a user to that movie:

```
users <- sample(unique(edx$userId), 100)
sample_matrix <- edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users", main = "Movies rated by a sample of 100 users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```

The colors represent the rating scale from the lowest rating shown in yellow to the highest rating shown in intense red.

The challenge with this recommendation system is that each outcome has a different set of predictors. For example, to predict the rating for movie i by user u we will need to consider how user u is rating other movies similar to movie i, how movie i is rated by other users similar to user u and how often movie i is rated overall - we have already seen that some movies get rated a lot while others are barely rated.

Therefore, in predicting the rating of the movie i by user u we will need to consider the entire matrix, i.e. all movies and users.

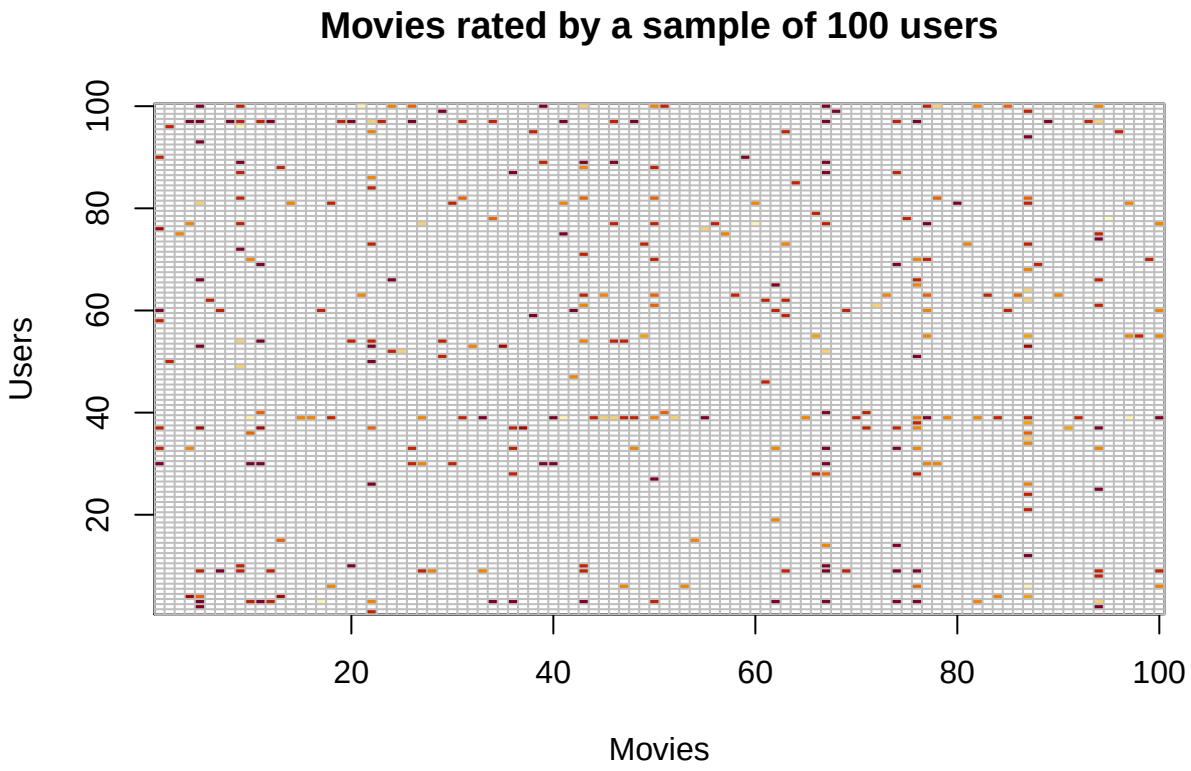

Figure 4: Sample of 100 users and distribution of their ratings

Step 3: Preparing the train dataset (edx)

In order to train and test different models we'll use the *edx* dataset and split it further into train and holdout test datasets. Before we do that, though, we'll take a more manageable sample of the *edx* dataset that we will call *edx_sample* - **this will allow us to render this report into pfd** (see the note in the Introduction section). We will make sure that all the users and movies in this sample are also in the final_holdout_test and we'll filter this sample to only include the movies that have been rated at least 100 times and the users that rated at least 50 movies.

```
movieIds_to_keep <- edx %>% count(movieId) %>% filter(n>=100) %>% pull(movieId)
userIds_to_keep <- edx %>% count(userId) %>% filter (n>=50) %>% pull(userId)
edx_to_use <- edx %>% semi_join(final_holdout_test, by= "userId") %>%
  semi_join(final_holdout_test, by = "movieId")
edx_sample <- edx_to_use %>%
  filter(userId %in% userIds_to_keep, movieId %in% movieIds_to_keep)
```

Let's now create a train and a test set from the *edx_sample* dataset:

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
sampletest_index <- createDataPartition(y = edx_sample$rating, times=1,
                                        p = 0.2,
                                        list = FALSE)
edx_sample_train <- edx_sample[-sampletest_index, ]
temp_sample_test <- edx_sample[sampletest_index,]
```

Let's make sure that *userId* and *movieId* in the *edx_sample_test* set that we'll build next are also in the *edx_sample_train* set:

```
edx_sample_test <- temp_sample_test %>%
  semi_join(edx_sample_train, by = "movieId") %>%
  semi_join(edx_sample_train, by = "userId")
```

Let's add the rows removed from *edx_sample_test* set back into *edx_sample_train* set:

```
removed <- anti_join(temp_sample_test, edx_sample_test)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx_sample_train <- rbind(edx_sample_train, removed)
```

Let's remove the temporary objects that we have created to free up space:

```
rm(removed, sampletest_index, temp_sample_test)
```

Let's now take a look at our newly created *edx_sample_train* dataset:

```
str(edx_sample_train)
```

```
## 'data.frame':    6359391 obs. of  6 variables:
##  $ userId   : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ movieId  : int  25 28 30 32 52 57 58 111 141 194 ...
##  $ rating   : num  3 3 5 5 4 3 3 4 3 3 ...
##  $ timestamp: int  857911265 857913507 857911752 857911264 857911416 857912840 857911416 857912365 85
##  $ title    : chr  "Leaving Las Vegas (1995)" "Persuasion (1995)" "Shanghai Triad (Yao a yao yao dao
##  $ genres   : chr  "Drama|Romance" "Drama|Romance" "Crime|Drama" "Sci-Fi|Thriller" ...
```

We'll now create a matrix from the *edx_sample_train* dataset which will have *user IDs* as rows, *movie IDs* as columns and values as ratings for each user/movie combination. We'll save this matrix in the *y* object

```
y <- select(edx_sample_train, movieId, userId, rating)  %>%
  pivot_wider(names_from = movieId, values_from = rating)
rnames <- y$userId
y <- as.matrix(y[,-1])
rownames(y) <- rnames
dim(y)
```

```
## [1] 40642  5711
```

Let's see the first 5 rows and columns of this matrix with a row for all the ratings given by each userId to each of the movies:

```
y[1:5, 1:5]
```

```
##    25 28 30 32 52
## 5   3  3  5  5  4
## 7  NA NA NA  4 NA
## 8  NA NA NA NA NA
## 10  3 NA NA NA NA
## 11 NA NA NA NA NA
```

To evaluate how good the models are at predicting the ratings we will use the Root Mean Squared Error (RMSE) as the loss function. We'll build a function first that calculates this RMSE:

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

Step 4: Building and evaluating different models

First Model: average rating for all movies, regardless of user or movie

In this very simple model we predict that the rating for any movie will be the average rating accross all movies. In this case, the variation of ratings from movie to movie will be explained by a random error occurring from a user/movie combination.

Let's calculate the average rating for all movies in the edx_train dataset:

```
avg <- mean(y, na.rm = TRUE)
avg
```

```
## [1] 3.505514
```

If we predict all unknown ratings as the avg rating of all movies then the RMSE will be:

```
avg_rmse <- RMSE (edx_sample_test$rating, avg)
avg_rmse
```

```
## [1] 1.054676
```

We need to get close to 0.8649 though, so we definitely need to find better ways to improve our model.

Let's create a table with the RMSE obtained from each of our models and record the result of our first model:

```
rmse_models <- data.frame (model = "A simple average", RMSE = avg_rmse)
rmse_models %>% kable()
```

model	RMSE
A simple average	1.054676

Second Model: taking into account a movie effect

We've already noticed that some movies get rated higher than others (e.g. blockbusters).

In this model we predict that the rating of each movie is comprised of the average rating of all movies plus a movie effect: *movie rating = average rating for all movies + movie effect*

This movie effect will be the one explaining the differences in ratings between movies.

To estimate the movie effect we can use least squares by fitting a linear regression model on the edx_sample_train dataset (e.g. *fit <- lm(rating ~ as.factor(movieId), data = edx_sample_train)*) but because we have thousands of movies and thus thousands of movie effects this linear model will take a very long time to run.

Instead, by looking at the movie rating equation above, we can tell that the least squares estimate of the movie effect is actually the average of the difference between the rating of each movie and the average rating for all movies: *movie effect estimate = avg (movie rating - average rating for all movies)*

```
movie_effect <- colMeans (y - avg, na.rm = TRUE)
```

Let's create a histogram to see the distribution of these movie effects (m_effect):

```
qplot(movie_effect, bins=10, color = I("gray"), main = "Movie effect distribution")
```

We can see that the movie effect varies from -2.5 to 1.5 which means that if we add this movie effect to the average rating for all movies we obtain the rating for each movie. For example, if a movie effect is 1.5 it means that the movie rating will be maximum (1.5 + 3.5 = 5).

Let's now build a data frame to show the movie effects for each movie:

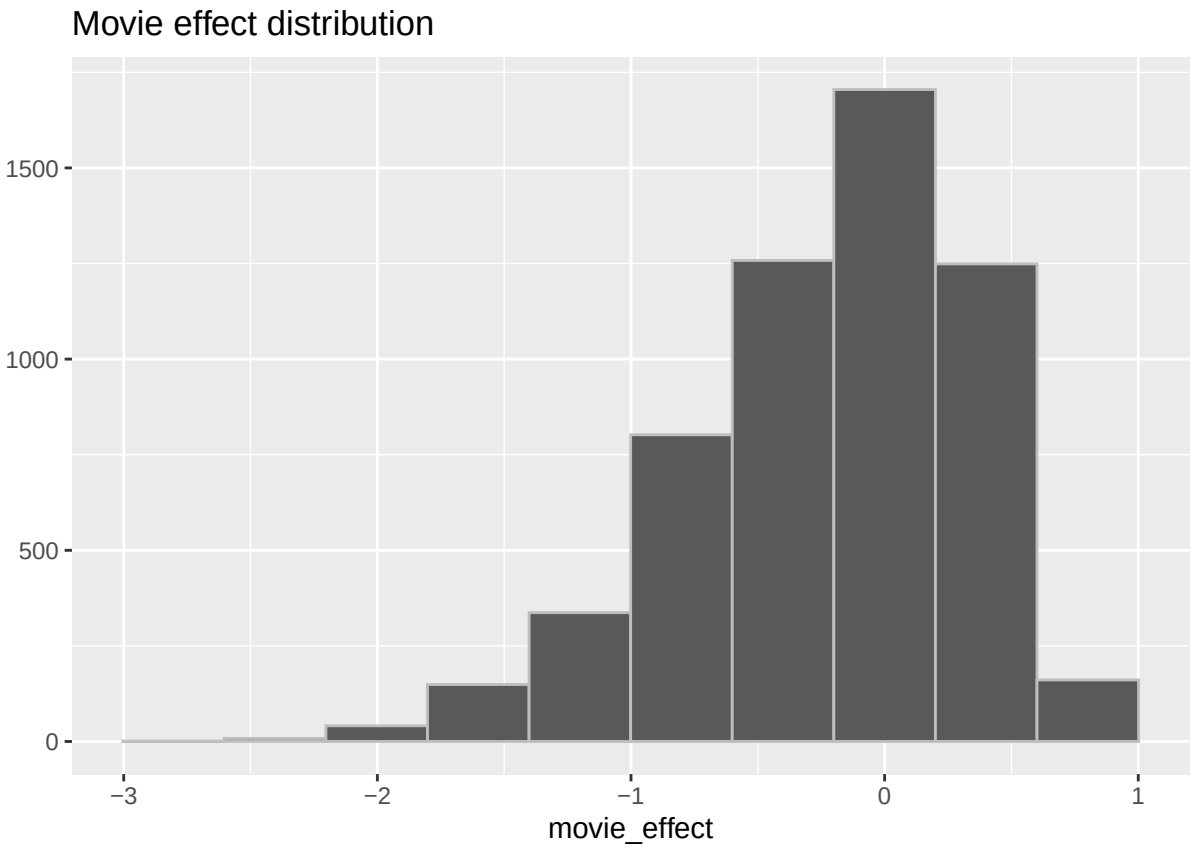

Figure 5: Movie effect distribution

```
fit_movie_effect <- data.frame (movieId = as.integer(colnames(y)),
                                average = avg, movie_effect = movie_effect)
head(fit_movie_effect)
```

```
##    movieId  average movie_effect
## 25      25 3.505514   0.18469208
## 28      28 3.505514   0.54030109
## 30      30 3.505514   0.13815340
## 32      32 3.505514   0.40525799
## 52      52 3.505514   0.03203304
## 57      57 3.505514  -0.20338229
```

and lets join this movie effect dataframe with the *edx_sample_test* dataset so we can calculate our prediction and compare it easily with the actual rating in the *edx_sample_test* dataset:

```
movie_rmse <- left_join(edx_sample_test, fit_movie_effect, by = "movieId") %>%
  mutate(prediction = average + movie_effect) %>%
  summarize(rmse = RMSE(rating,prediction))
movie_rmse
```

```
##        rmse
## 1 0.9352921
```

Let's add this to our rmse_models data frame for an easy comparison:

```
rmse_models <- rbind(rmse_models, list("A movie effect model",movie_rmse ))
rmse_models %>% kable()
```

model	RMSE
A simple average	1.054676
A movie effect model	0.9352921

Third Model: taking into account the movie plus a user effect

We've already seen that the same movie gets rated differently by different users which means that the user preferences need to be taken into consideration as well.

Let's improve our model by also adding the user effect to the movie effect. Let's first calculate the average rating accross all movies for each user in the *edx_sample_train* dataset:

```
user_avg_rating <- rowMeans(y, na.rm = TRUE)
```

And let's plot this user effect in a histogram:

This user effect allows us to predict that a user who usually gives movies lower ratings is also expected to give a lower rating to a highly rated movie.

We can use least squares again to estimate the user effect by employing a linear model (we can use, for example, *fit <- lm(rating ~ as.factor(movieId) + as.factor(userId), data = edx_sample_train)*) but this will take a long time because of the size of our dataset.

Instead, we can estimate an approximation of the user effect by calculating the average of the difference between the movie rating, the average rating of all movies and the movie effect, like this:

```
user_effect <- rowMeans(sweep(y-avg, 2, movie_effect), na.rm = TRUE)
```

Let's now build a data frame to show the user effects for each user:

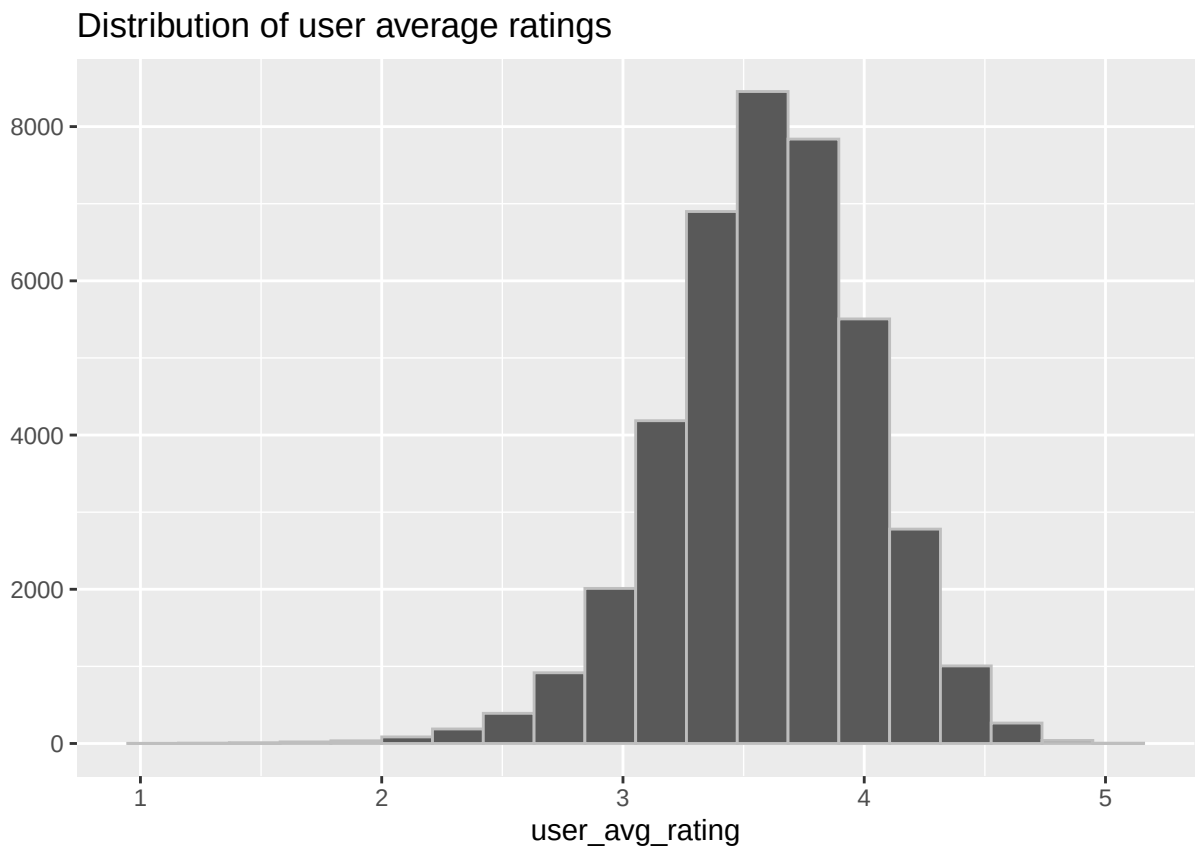

Figure 6: Distribution of user average ratings

```
fit_user_effect <- data.frame(userId = as.integer(rownames(y)),
                              user_effect = user_effect)
```

and let's join this user effect dataframe with the *edx_sample_test* dataset and the movie effect dataframe so we can calculate our prediction and compare it easily with the actual rating in the *edx_sample_test* dataset:

```
user_rmse <- left_join(edx_sample_test, fit_movie_effect, by = "movieId") %>%
  left_join(fit_user_effect, by = "userId") %>%
  mutate(prediction = average + movie_effect + user_effect) %>%
  summarize(rmse = RMSE(rating,prediction))
user_rmse
```

```
##        rmse
## 1 0.8568664
```

Let's add the user_rmse to our *rmse_models* dataframe:

```
rmse_models <- rbind(rmse_models, list("A movie plus user effect model", user_rmse))
rmse_models %>% kable()
```

model	RMSE
A simple average	1.054676
A movie effect model	0.9352921
A movie plus user effect model	0.8568664

Fourth Model: applying regularization to the movie effect model

We know already that some movies get more rated than others. For the movies that get lots of rating, when calculating the movie effect, averaging these ratings will give us a fairly good estimate of the movie effect.

However, when a movie has only one or two ratings the movie effect will only reflect those few ratings which means that it's more likely to have larger estimates of the movie effect which will cause larger errors. Large errors in the movie effect will increase our RMSE.

Therefore, we need to give a weight or a penalty to each movie effect depending on how many ratings a movie has. The more ratings a movie has the more weight we will give to the movie effect and the lower the penalty will be. Regularization allows us to penalize large estimates that are built based on small sample sizes.

We will apply the penalty to each movie effect by dividing the movie effect to the sum between the number of ratings for that movie and the penalty number. The penalty number will be chosen by using cross validation:

```
penalties <- seq(from = 0, to= 10, by = 0.1)
count_ratings <- colSums(!is.na(y))
fit_movie_effect$count_ratings <- colSums(!is.na(y))
sum_movie_effect <- colSums(y-avg, na.rm = TRUE)

rmse_penalty <- sapply(penalties, function(penalty){
  fit_movie_effect$movie_effect <- sum_movie_effect/(count_ratings + penalty)
  left_join(edx_sample_test, fit_movie_effect, by = "movieId") %>%
    mutate(prediction = average + movie_effect) %>%
    summarize(rmse = RMSE(rating,prediction)) %>%
    pull(rmse)
  })
```

Let's plot these penalties to see which of these values minimizes the RMSE:

The penalty number that minimizes the RMSE is:

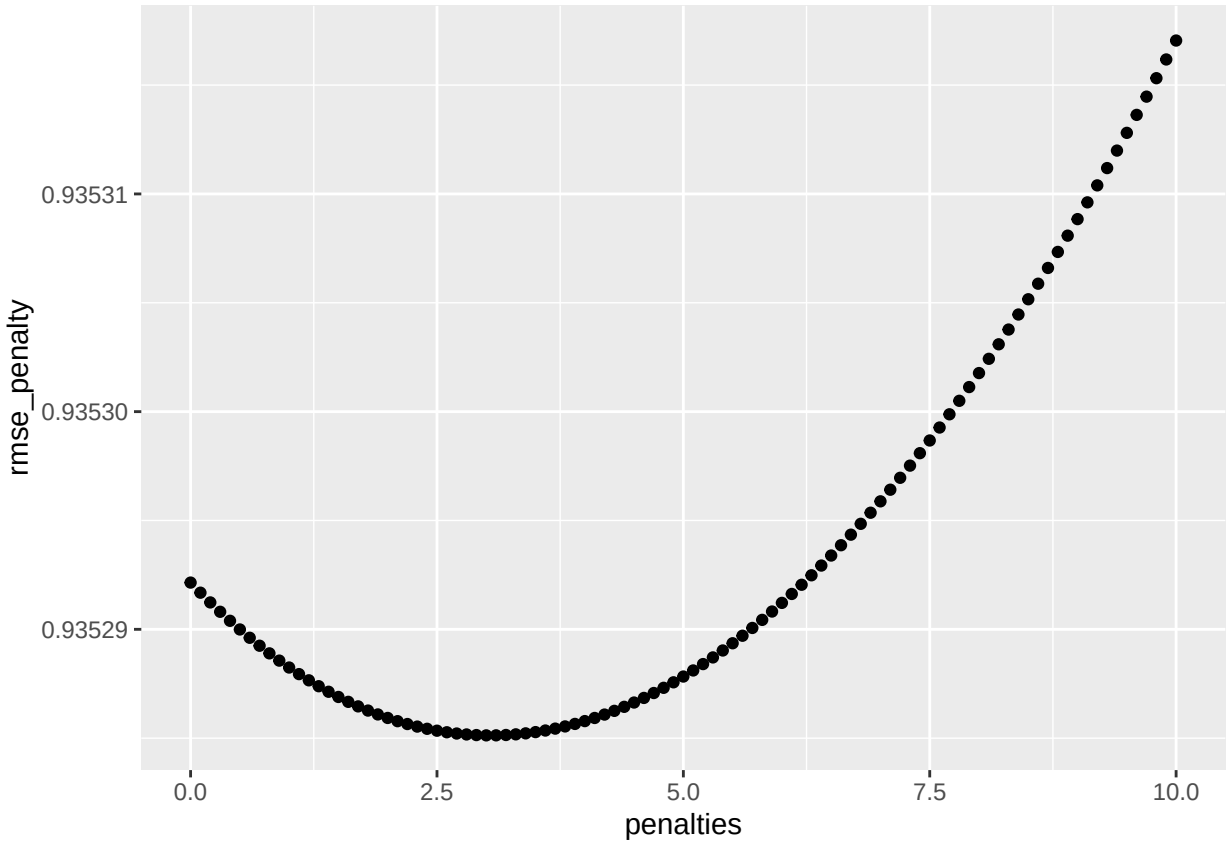

Figure 7: Vizualizing the RMSE for each penalty number

```
penalty <- penalties[which.min(rmse_penalty)]
penalty
```

```
## [1] 3
```

So we found that the penalty that will minimize the rmse for our movie effect is 3. Let's apply this penalty number to the movie effect model and create a regularized movie effect column:

```
fit_movie_effect$movie_effect_reg <- sum_movie_effect/(count_ratings + penalty)
```

Let's see if using this new regularized movie effect improves our rmse:

```
movie_rmse_reg <- left_join(edx_sample_test, fit_movie_effect, by = "movieId") %>%
  mutate(prediction = average + movie_effect_reg) %>%
  summarize(rmse = RMSE(rating,prediction))
movie_rmse_reg
```

```
##        rmse
## 1 0.9352851
```

Let's add this new rmse for movie effect with regularization to the rmse_models:

```
rmse_models <- rbind(rmse_models,
                     list("A movie effect model with regularization", movie_rmse_reg))
rmse_models %>% kable()
```

model	RMSE
A simple average	1.054676
A movie effect model	0.9352921
A movie plus user effect model	0.8568664
A movie effect model with regularization	0.9352851

We can see that there is not much improvement applying regularization to just the movie effect model. In the next section we'll apply regularization to the movie plus the user effect model to see if the new model performs better.

Fifth model: applying regularization to the movie and user effect model

As seen in the previous section, there wasn't much improvement when we applied regularization to just the movie effect model. In this section we'll now apply regularization to the movie plus user effect model.

Let's estimate again the user effect, this time using the regularized movie effect:

```
user_rmse_reg <- left_join(edx_sample_test, fit_movie_effect, by = "movieId") %>%
  left_join(fit_user_effect, by = "userId") %>%
  mutate(prediction = average + movie_effect_reg + user_effect) %>%
  summarize(rmse = RMSE(rating,prediction))
user_rmse_reg
```

```
##        rmse
## 1 0.8568621
```

Let's add this improved rmse to our *rmse_models* dataframe for easy comparison:

```
rmse_models <- rbind(rmse_models,
                     list("A movie + user effect model with regularization", user_rmse_reg))
rmse_models %>% kable()
```

model	RMSE
A simple average	1.054676
A movie effect model	0.9352921
A movie plus user effect model	0.8568664
A movie effect model with regularization	0.9352851
A movie + user effect model with regularization	0.8568621

Sixth Model: adding a genres effect

If we take a look at the movie genre and analyze the ratings by genre we can see that genre plays a role in ratings as well. In the plot below, when we take a look at the movies with more than 1000 ratings in the edx data set we notice that some genres get rated consistently lower, on average, compared to other genres - see Fig.8 below.

Let's create a model in which we explain the differences in ratings between movies through a genre effect. In this model we predict that the rating of each movie is comprised of the average rating of all movies plus a genre effect: *movie rating = average rating for all movies + genre effect*

To estimate the genre effect we can use least squares by fitting a linear regression model on the edx_train dataset (e.g. *fit_genres <- lm(rating ~ as.factor(genres), data = edx_sample_train)*) but because we have thousands of movies this linear model will take a very long time to run.

Instead, by looking at the movie rating equation above, we can tell that the least squares estimate of the genre effect is actually the average of the difference between the rating of each genre and the average rating for all movies: *genre effect estimate = avg (movie rating - average rating for all movies)*

```
genre_effect <- edx_sample_train %>%
  group_by(genres) %>%
  summarize(genre_effect = mean(rating)-avg) %>%
  mutate(genres = genres)
head(genre_effect, n=5)
```

```
## # A tibble: 5 x 2
##   genres                                                genre_effect
##   <chr>                                                        <dbl>
## 1 Action                                                      -0.585
## 2 Action|Adventure                                             0.146
## 3 Action|Adventure|Animation|Children|Comedy                    0.454
## 4 Action|Adventure|Animation|Children|Comedy|Fantasy           -0.506
## 5 Action|Adventure|Animation|Children|Comedy|Sci-Fi            -0.431
```

Let's now add this genre effect to the user and movie with regularization model (i.e.fifth model that we just built) and test it against the *edx_sample_test* dataset.

```
genres_rmse <- left_join(edx_sample_test, fit_movie_effect, by = "movieId") %>%
  left_join(fit_user_effect, by = "userId") %>%
  left_join(genre_effect, by = "genres") %>%
  mutate(prediction = avg + movie_effect_reg + user_effect + genre_effect) %>%
  summarize(rmse = RMSE(rating,prediction))
genres_rmse
```

```
##       rmse
## 1 0.9051419
```

Unfortunately, as we can see, the genre effect didn't manage to further minimize the error on the test dataset.

Let's add the genres_rmse to our rmse_modes data frame for an easy comparison:

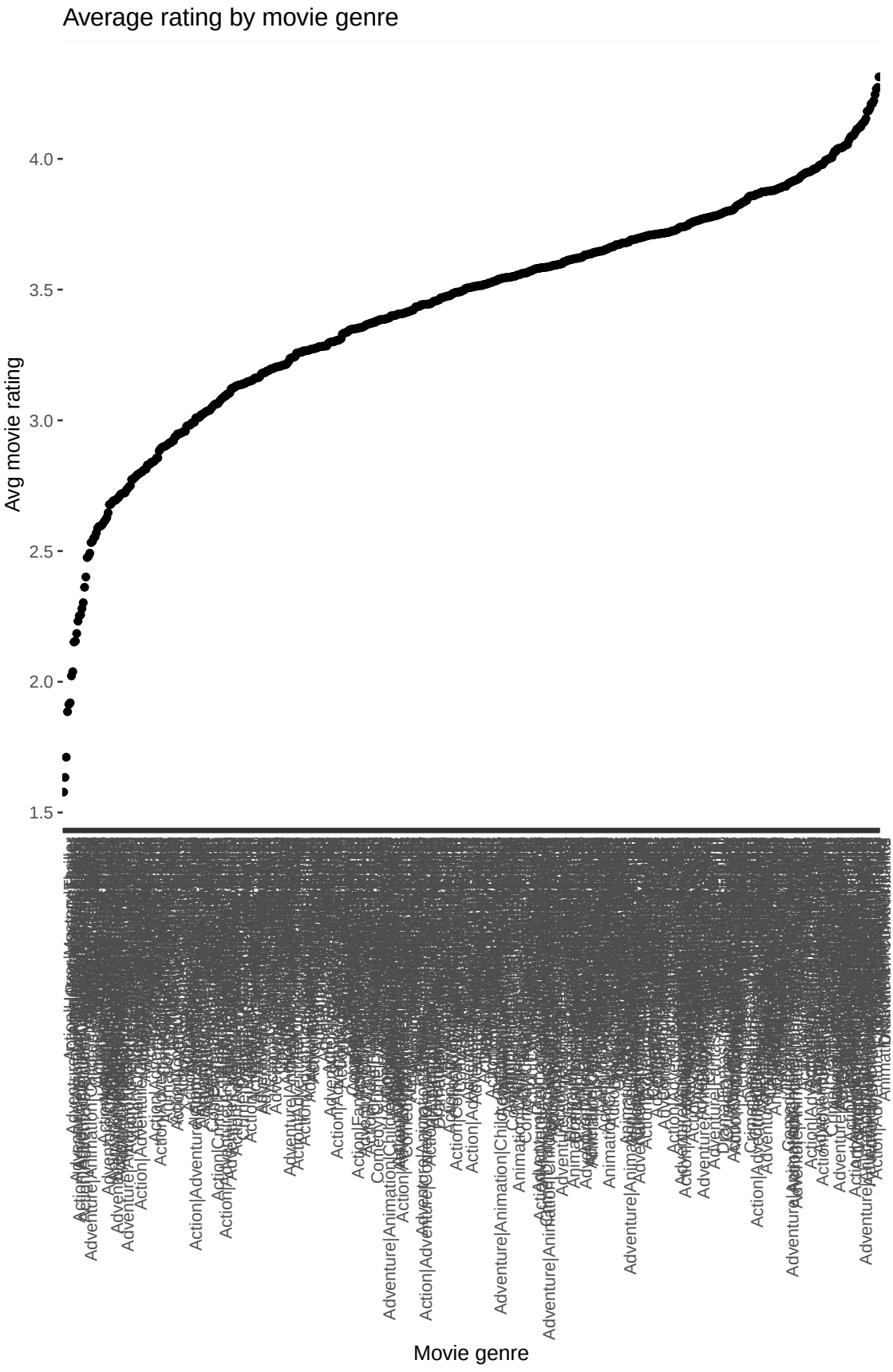

Figure 8: A sample of average ratings by movie genre

```
rmse_models <- rbind(rmse_models, list("A movie + genre + user effect model with regularization",
                                       genres_rmse))
rmse_models %>% kable()
```

model	RMSE
A simple average	1.054676
A movie effect model	0.9352921
A movie plus user effect model	0.8568664
A movie effect model with regularization	0.9352851
A movie + user effect model with regularization	0.8568621
A movie + genre + user effect model with regularization	0.9051419

We can now see that our best model is the one that takes into account both movie and user effect with regularization as the RMSE for this model is the lowest of all the models we developed so far: 0.856862066833871

Results: testing our final model on the final holdout test data set

Let's now test our best model that we trained so far on the *final_holdout_test* set that we kept aside. Please keep in mind that because in this particular report (not in the actual R script) we have used a sample of the edx dataset, we need to make sure that the movieIds and the userIds from the final_holdout_test also match our edx_sample - this way we can calculate the rmse_final:

```
rmse_final <- inner_join(final_holdout_test, fit_movie_effect, by = "movieId") %>%
  inner_join(fit_user_effect, by = "userId") %>%
  mutate(prediction = avg + movie_effect_reg + user_effect) %>%
  summarize(rmse = RMSE(rating,prediction))
rmse_final
```

```
##        rmse
## 1 0.8562612
```

rmse final
0.8562612

Conclusion and next steps

We saw that building a model that takes into account a regularized movie effect plus a user effect gives us the lowest RMSE.

In general, we know that movies tend to be rated higher closer to the launch date of the movie since the fans of a particular genre are usually the ones rating first that movie. The longer the time passes the fewer the chances for the users to be influenced by the attention a movie gets in the media and online.

Perhaps an improved model could also take into account the time factor and look at the time lapsed from the launch of a movie to the time of the rating.

However, the best next step would be to use the recommenderlab package which has been designed specifically for building recommendation models. This package contains several recommendation algorithms to choose from. In our case, the most relevant, in my opinion, would be to use the collaborative filtering recommender systems, starting with the item-based collaborative filtering model first. This model contains a similarity function that calculates a similarity matrix between movies based on their genre and the ratings received. We could also look at applying the user-based collaborative filtering model in which the algorithm calculates the similarity between users.

When starting applying the item-based collaborative filtering on the edx_sample_train dataset and building the model it took 2 hours on my machine and still didn't finish processing. The next step will be to drastically reduce the train dataset to a small sample to help better estimate the amount of time it would take on my machine to build the prediction model on the entire edx_sample_train.