Rating Prediction with MovieLens 10M Dataset

Danilo Ferreira de Oliveira

04/12/2021

Contents

1 Introduction

For this project, we will be creating a movie recommendation system using the MovieLens dataset. You can find the entire latest MovieLens dataset here. You will be creating your own recommendation system using all the tools shown throughout the courses in the HarvardX: Data Science Professional Certificate series. We will use the 10M version of the MovieLens dataset to make the computation a little easier.

You will train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

Recommendation systems are more complicated machine learning challenges because each outcome has a different set of predictors. For example, different users rate a different number of movies and rate different movies. By evaluating the movies specific users like to watch, we can predict ratings they would give for movies they haven't seen, and then recommend them those with relatively high ratings.

We will firstly follow the approach made by Professor Rafael Irrizarry in Introduction to Data Science, and continue it with the addition of genre effects, and a model that uses matrix factorization.

To compare different models or to see how well we are doing compared to a baseline, we will use root mean squared error (RMSE) as our loss function. We can interpret RMSE similar to standard deviation.

ou will use the following code to generate your datasets. Develop your algorithm using the edx set. For a final test of your final algorithm, predict movie ratings in the validation set (the final hold-out test set) as if they were unknown. RMSE will be used to evaluate how close your predictions are to the true values in the validation set (the final hold-out test set).

2 Analysis

2.1 Getting the data

Before we can start gathering data, we will first load the R libraries we will need along all the report.

```
library(caret)
library(tidyverse)
library(ggplot2)
library(lubridate)
library(stringr)
library(recosystem)
library(data.table)
library(kableExtra)
library(tibble)
```

Then, we download the MovieLens data and run code provided by HarvardX to generate the datasets for training and evaluating the final model. The `edx` set will be used all along the report to train and choose the best model for rating predictions, while the `validation` set will only be used at the end of to evaluate the quality of the final model with the root mean square error.

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
```

```
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

The validation set will be 10% of the MovieLens data obtained. We need to make sure the user and movie IDs in `validation` are also in `edx`, so we apply the `semi_join()` transformations. Then, we add rows removed from the validation set back into the training set. Lastly, we clean variables to recover RAM space.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

We can now start analyzing the dataframe `edx`.

2.2 Exploratory data analysis and visualization

2.2.1 Exploring the variables

Let's take a look at the columns in our dataset.

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

```
>> # A tibble: 9,000,055 x 6
>>    userId movieId rating timestamp title                  genres
>>     <int>   <dbl>  <dbl>     <int> <chr>                  <chr>
>>  1      1     122      5 838985046 Boomerang (1992)       Comedy|Romance
>>  2      1     185      5 838983525 Net, The (1995)        Action|Crime|Thriller
>>  3      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|T~
>>  4      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-~
>>  5      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
>>  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
>>  7      1     356      5 838983653 Forrest Gump (1994)    Comedy|Drama|Romance|~
>>  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
>>  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
>> 10      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
>> # ... with 9,000,045 more rows
```

It is possible to see that columns `title` and `genres` are of character class (`<chr>`), while `movieId` and `rating` are numeric (`<dbl>`), `userId` and `timestamp` are integers (`<int>`).

We should analyze the quantity of unique values for each column variable.

```
edx %>% summarize(n_userIds = n_distinct(userId), n_movieIds = n_distinct(movieId),
                  n_titles = n_distinct(title), n_genres = n_distinct(genres)) %>%
  kbl(booktabs = TRUE, caption = "Distinct values for userId, movieId, titles and genres") %>% kable_sty
```

Table 1: Distinct values for userId, movieId, titles and genres

n_userIds	n_movieIds	n_titles	n_genres
69878	10677	10676	797

Note that there more `movieId` unique variables than there are `title` ones. We need to investigate why.

```
edx %>% group_by(title) %>% summarize(n_movieIds = n_distinct(movieId)) %>%
  filter(n_movieIds > 1) %>% kbl(booktabs = TRUE, caption = "Movies with more than one ID") %>%
  kable_styling(latex_options = c("striped","hold_position"))
```

Table 2: Movies with more than one ID

title	n_movieIds
War of the Worlds (2005)	2

So we find out that *War of the Worlds (2005)* has two distinct `movieIds`. Let's see how many reviews for each of them there are.

Table 3: Different values of movieId and genres for *War of the Worlds (2005)*

movieId	title	genres	n
34048	War of the Worlds (2005)	Action|Adventure|Sci-Fi|Thriller	2460
64997	War of the Worlds (2005)	Action	28

The second `movieId` has only 28 reviews to it. It is a low value and the validation set possibly has these two movieIds for the movie, so we won't meddle with it.

2.2.2 Ratings

We can now visualize the data. Let's see how the ratings distribution is configured.

From the plot above, we can affirm that full star ratings are more common than those with half stars, since 4, 3 and 5 are the most frequent ones.

2.2.3 Movie and user IDs

The plots above show the frequency of quantities of reviews that each user gives and quantities of reviews each movie receives. Their x-axis are both on a logarithmic scale, and the one on the left is *rightly skewed*.

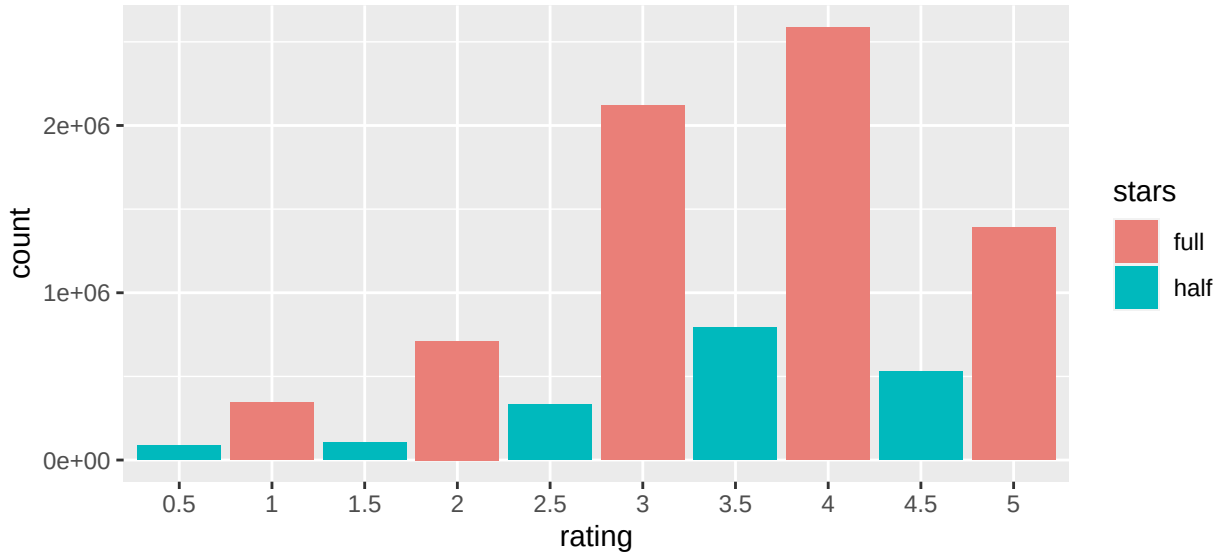

Figure 1: Distribution of ratings

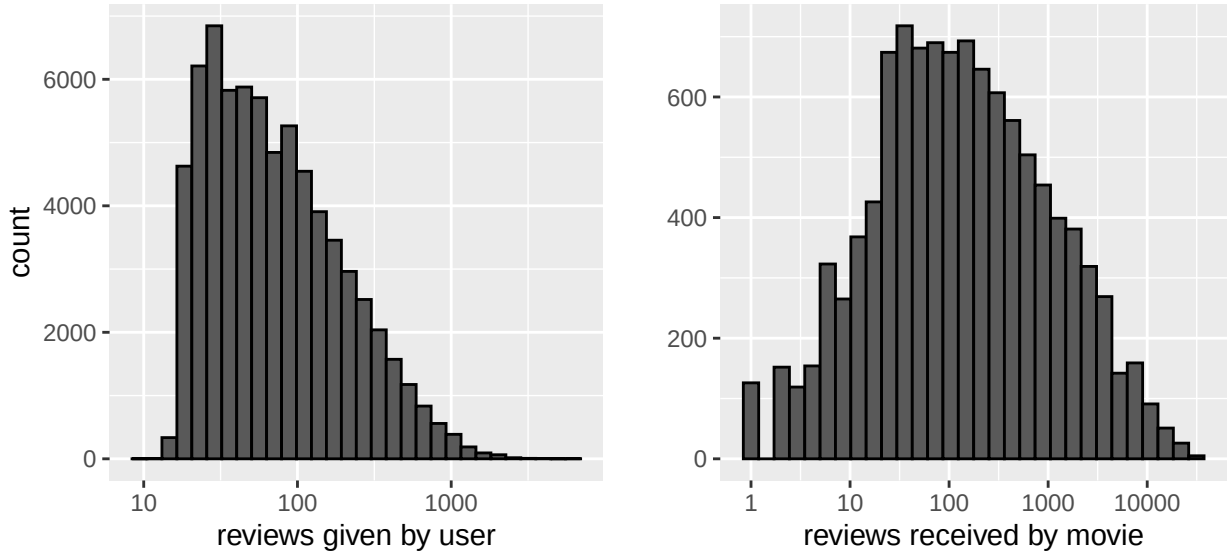

Figure 2: Frequency of reviews given by a specific user (left) and of reviews received by a specific movie (right)

If we analyze them, we can say that the majority of the users give from 20 to 200 reviews, while each movie frequently receive between 20 to a 1000 reviews, which is a pretty broad range.

Plotting the histogram for average ratings, we get the plots below for both users and movies. It is possible to see that it is rare for a user to average rating movies below 2, since the top plot shows there are few users to the left of that number, only 185 of all 69878 in the dataset, to be more exact.

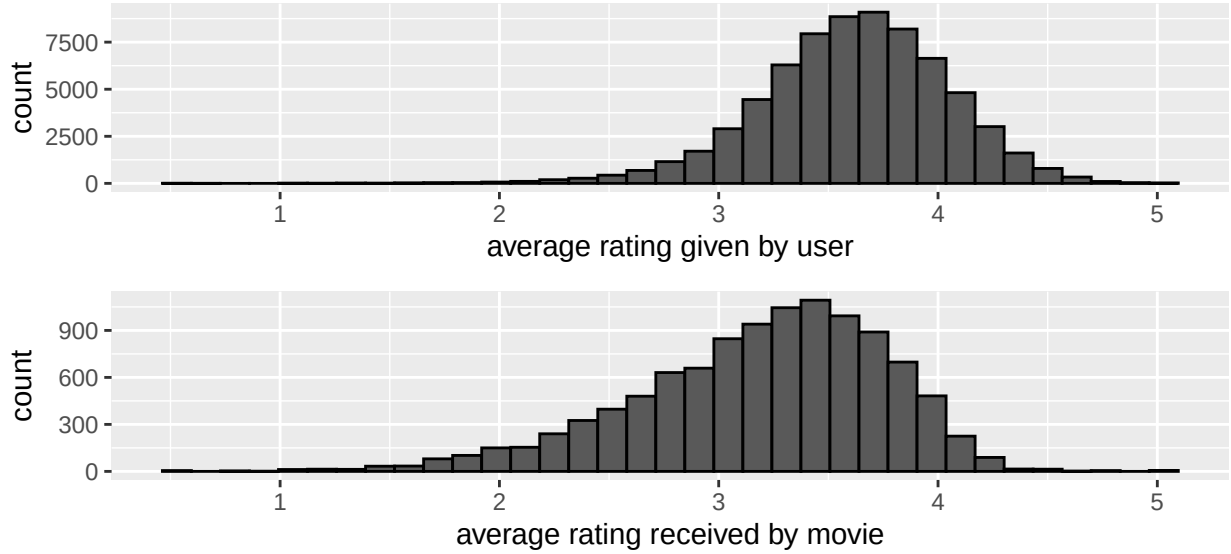

Figure 3: Frequency of average rating a user gives (top) and average rating a movie receives (bottom)

From the 1.0676×10^4 movies, we can see that only 61 of them average below 1.5, and 57 films average over 4.25 rating.

Let's now explore the top 10 most reviewed movies in the dataset and how many times they were reviewed.

As we can see, the most reviewed movies are normally blockbusters, and they also have really high ratings. We can infer that the most times a movie is reviewed, the better its rating is.

```
edx %>% filter(title %in% head(unique_movies$title,n=10)) %>% group_by(title) %>%
  summarize(avg=mean(rating)) %>% arrange(desc(avg))
```

```
>> # A tibble: 10 x 2
>>    title                                                      avg
>>    <chr>                                                    <dbl>
>>  1 Shawshank Redemption, The (1994)                          4.46
>>  2 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  4.22
>>  3 Silence of the Lambs, The (1991)                          4.20
>>  4 Pulp Fiction (1994)                                       4.15
>>  5 Braveheart (1995)                                         4.08
>>  6 Forrest Gump (1994)                                       4.01
>>  7 Fugitive, The (1993)                                      4.01
>>  8 Terminator 2: Judgment Day (1991)                         3.93
>>  9 Apollo 13 (1995)                                          3.89
>> 10 Jurassic Park (1993)                                      3.66
```

the title of the movies come along with the release date, inside parenthesis.

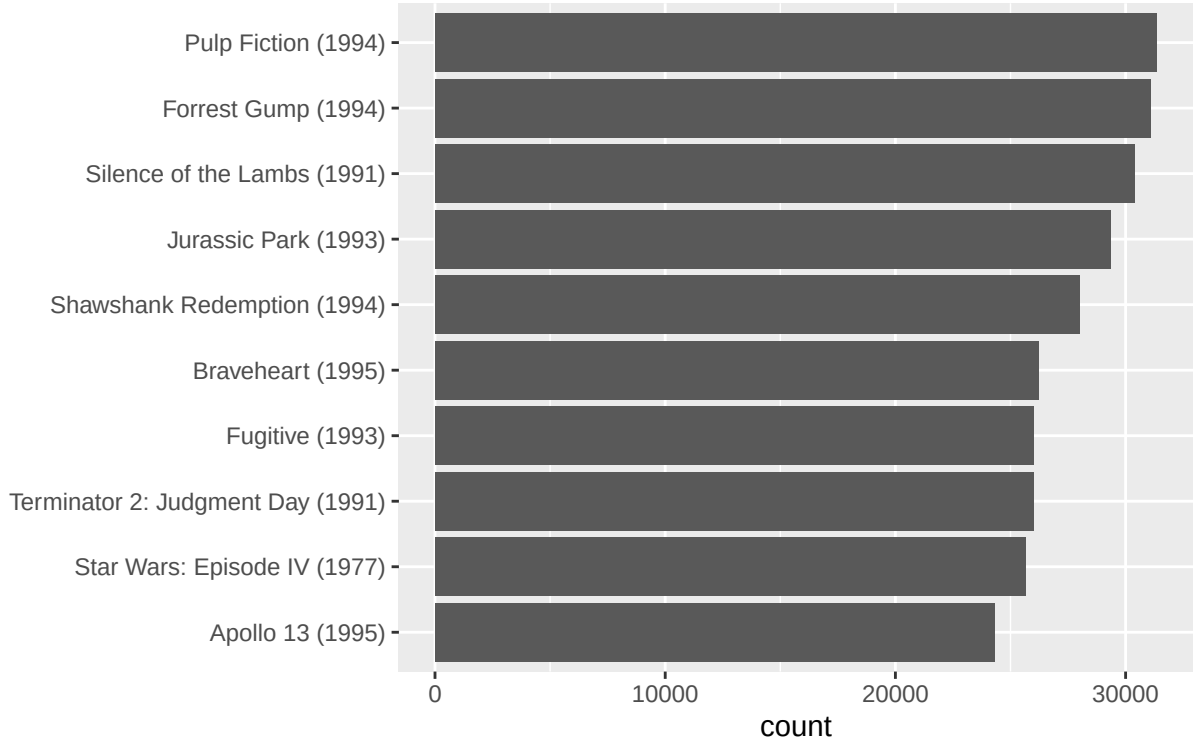

Figure 4: Number of reviews for the top 10 most reviewed movies

2.2.4 Release date

Since there isn't a column for each movie's release date, We will check it by using the `str_extract` function from the `stringr` package.

```
release_date <- edx$title %>% str_extract('\\([0-9]{4}\\)') %>%
  str_extract('[0-9]{4}') %>% as.integer()
summary(release_date)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>    1915    1987    1994    1990    1998    2008
```

We first apply this function extracting the date along with the parenthesis, and then the number from inside them. We do it like this because by extracting only the number, movies like *2001: A Space Odyssey (1968)* and *2001 Maniacs (2005)* show up when searching for year 2001, for example. To go even further, it also extracts the numbers 1000 and 9000 for *House of 1000 Corpses (2003)* and *Detroit 9000 (1973)*, respectively.

Below we can see that the earliest release date in our dataset (1915) only has one movie:

Table 4: Movie with the earliest release date in the dataset

movieId	title	genres	n
7065	Birth of a Nation, The (1915)	Drama|War	180

We now compute the number of ratings for each movie and then plot it against the year the movie came out, using a boxplot for each year. The logarithmic transformation is applied on the y-axis (number of ratings) when creating the plot.

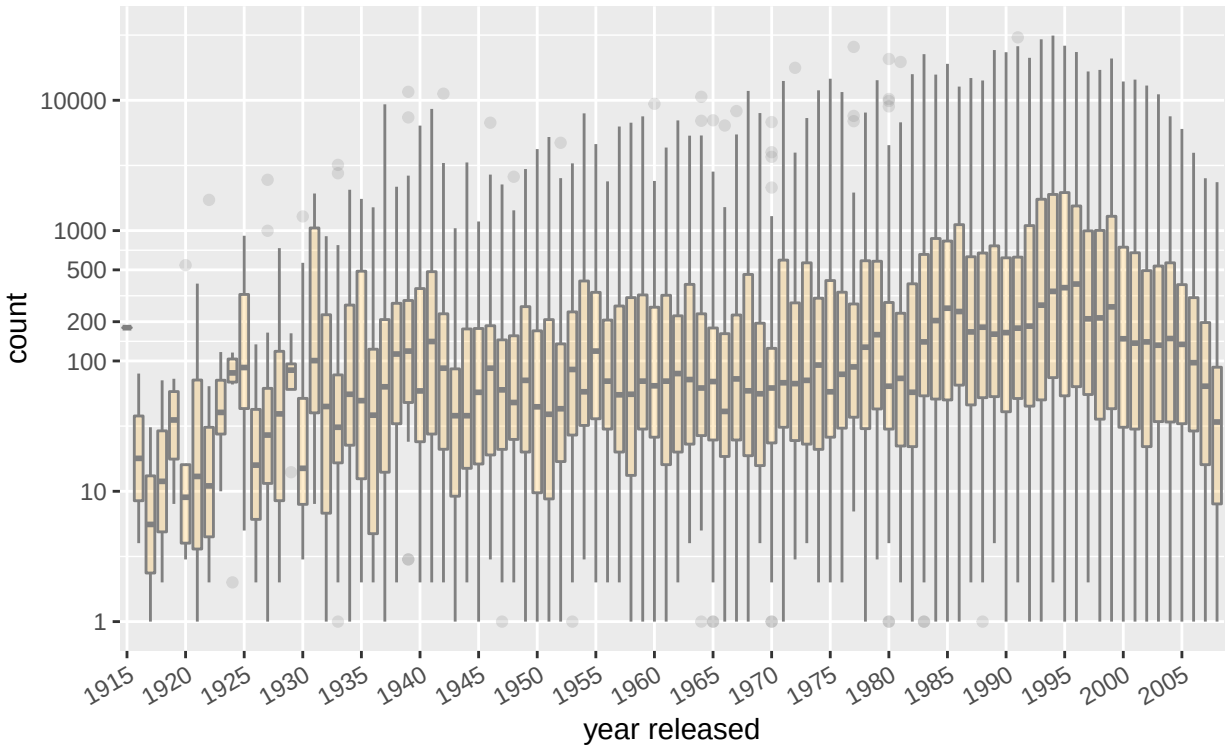

Figure 5: Number of reviews the movies received versus the year they were released

We see that, on average, movies that came out in the mid 90's get more ratings, reaching values over 30,000 reviews. We also see that with newer movies, starting near 1998, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it.

Now we compute the average of ratings for each movie and plot it against the release year with boxplots, similarly to before.

It is possible to see that most movies released until 1973 normally range between 3.0 and 4.0 ratings. From then on, the range gets bigger and values get lower. The median value goes from averaging over 3.5 to near 3.25. There are even some movies from around the 2000's that average 0.5 ratings, but they are outliers and probably only received very few ratings.

2.2.5 Genres and genre combinations

The `genres` variable actually represents combinations of a series of unique genres. We will count the number of reviews for all different combinations that appear in our dataset and arrange them in descending order. The table below shows, as an example, the top 7 most reviewed genre combinations.

Let's check the genre effect on ratings. We will filter for genre combinations that were reviewed over a thousand times, put them in ascending order of average rating and show 15 of them, equally distant inside the arranged dataframe. The indexes to select the genres in the arranged set are represented by the variable `gcomb_15`.

```
gcomb_15 <- c(1,seq(31,415,32),444)
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n > 1000) %>% arrange(desc(avg)) %>%  .[gcomb_15,] %>%
  mutate(genres = reorder(genres, avg)) %>%
```

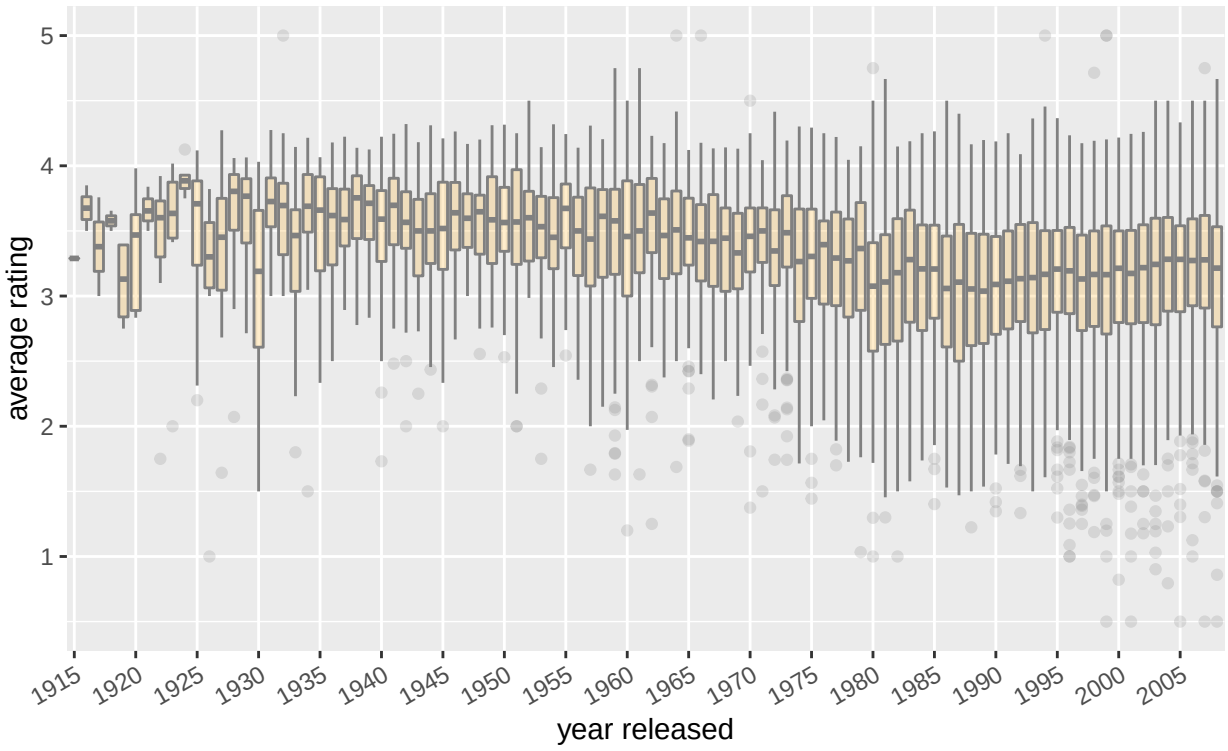

Figure 6: Average rating the movies received versus the year they were released

Table 5: Top genre combinations rated

rank	genres	count
1	Drama	733296
2	Comedy	700889
3	Comedy|Romance	365468
4	Comedy|Drama	323637
5	Comedy|Drama|Romance	261425
6	Drama|Romance	259355
7	Action|Adventure|Sci-Fi	219938

```
  ggplot(aes(x = genres, y = avg,
             ymin = avg - 2*se, ymax = avg + 2*se)) + geom_point() +
  geom_errorbar() + theme(axis.text.x=element_text(angle = 30, hjust = 1)) +
  labs(x='', y= 'average rating')
```

From the plot above, it is clear the effect genre combinations have on the rating, since they range from approximately 2.0 to over 4.5. Let's now see the effect of each genre separately.

There are 20 different genres in this dataset, and they are the following:

We can count how many reviews each genre have had, by applying the following function:

```
all_genres_count <- sapply(all_genres, function(g) {
  sum(str_detect(edx$genres, g))
```

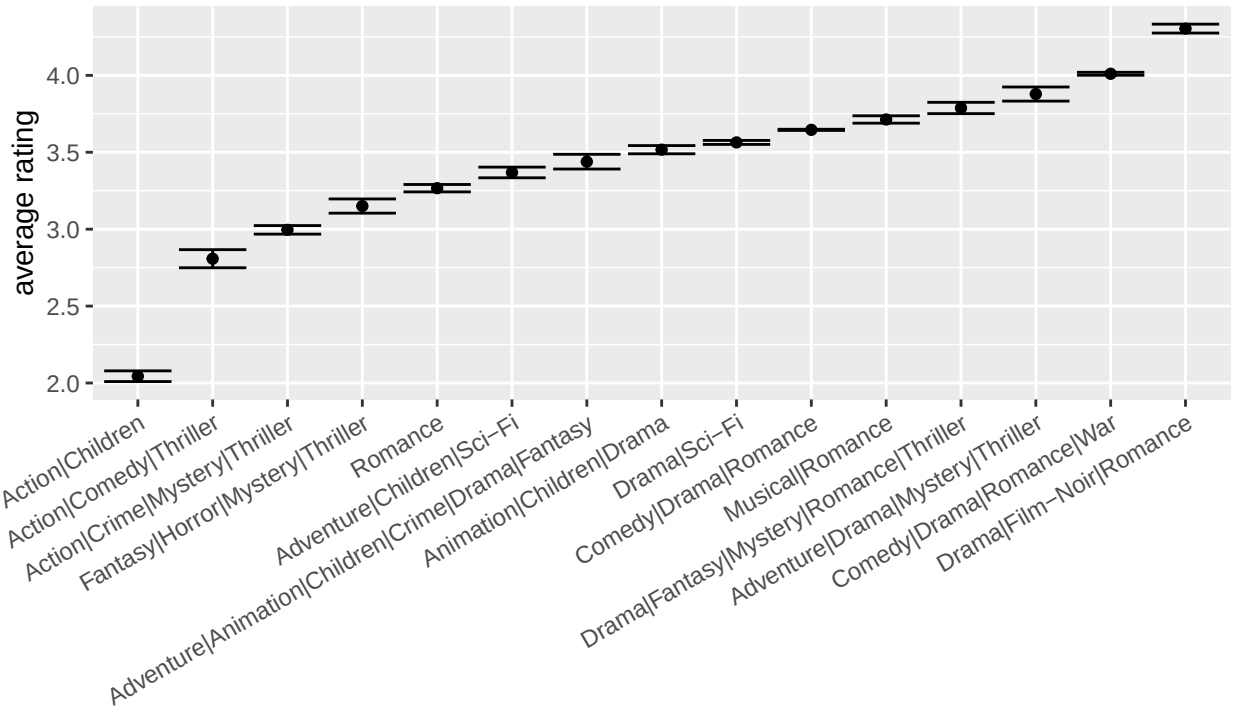

Figure 7: Distribution of average ratings for a few genre combinations possible

Table 6: All unique genres present in the dataset

Comedy	Drama	War	Film-Noir
Romance	Sci-Fi	Animation	Horror
Action	Adventure	Musical	Documentary
Crime	Children	Western	IMAX
Thriller	Fantasy	Mystery	(no genres listed)

```
}) %>% data.frame(review_count=.) %>%
  arrange(desc(.))
```

We can see the results on the table below. The genres range from near 4 million to only 7 reviews.

We need to investigate the `(no genres listed)` genre, which actually represents no genre at all. We can see the movies that belong to it and their respective averages and standard errors:

```
edx[which(edx$genres=='(no genres listed)'),] %>%
  summarize(movieId=movieId[1], userId=userId[1], title=title[1], genres=genres[1],
            n = n(), avg = mean(rating), se = sd(rating)/sqrt(n()))
```

```
>>   movieId userId                title             genres n      avg        se
>> 1    8606   7701 Pull My Daisy (1958) (no genres listed) 7 3.642857 0.4185332
```

There is only one movie in the dataset with no genres listed, and it has only 7 reviews, giving its average rating a high standard error (0.4185). For this reason, we won't be plotting this genre's average rating along with the others.

Table 7: Number of reviews per genre

all_genres_count[1:10]		all_genres_count[11:20]	
Genre	Count	Genre	Count
Drama	3910127	Horror	691485
Comedy	3540930	Mystery	568332
Action	2560545	War	511147
Thriller	2325899	Animation	467168
Adventure	1908892	Musical	433080
Romance	1712100	Western	189394
Sci-Fi	1341183	Film-Noir	118541
Crime	1327715	Documentary	93066
Fantasy	925637	IMAX	8181
Children	737994	(no genres listed)	7

The plot below shows the average ratings for each unique genre, as they range from under 3.3 to over 4.0.

We now observe the top genres reviewed in this dataset, and how many reviews each one of them have. We accounted for the genres that were reviewed over a million times, resulting in 8 distinct possibilities. The reviews count for all genre has already been shown, but we will plot the count for the top 8 genres to better visualize it, and also compare its shape with the movie count plot for the top 8.

We now plot how many different movies where reviewed for each of the top 8 genres.

The general shape and proportion of the bars pretty much look the same for both reviews and movie counts.

2.2.6 Rating date

Since `timestamp` doesn't really have any value for us as it is, we create from it another column that gives us the date and time a movie was rated.

```
edx <- mutate(edx, date = as_datetime(timestamp))
edx %>% as_tibble()
```

```
>> # A tibble: 9,000,055 x 7
>>    userId movieId rating timestamp title        genres       date
>>     <int>   <dbl>  <dbl>     <int> <chr>        <chr>        <dttm>
>>  1      1     122      5 838985046 Boomerang (~ Comedy|Roma~ 1996-08-02 11:24:06
>>  2      1     185      5 838983525 Net, The (1~ Action|Crim~ 1996-08-02 10:58:45
>>  3      1     292      5 838983421 Outbreak (1~ Action|Dram~ 1996-08-02 10:57:01
>>  4      1     316      5 838983392 Stargate (1~ Action|Adve~ 1996-08-02 10:56:32
>>  5      1     329      5 838983392 Star Trek: ~ Action|Adve~ 1996-08-02 10:56:32
>>  6      1     355      5 838984474 Flintstones~ Children|Co~ 1996-08-02 11:14:34
>>  7      1     356      5 838983653 Forrest Gum~ Comedy|Dram~ 1996-08-02 11:00:53
>>  8      1     362      5 838984885 Jungle Book~ Adventure|C~ 1996-08-02 11:21:25
>>  9      1     364      5 838983707 Lion King, ~ Adventure|A~ 1996-08-02 11:01:47
>> 10      1     370      5 838984596 Naked Gun 3~ Action|Come~ 1996-08-02 11:16:36
>> # ... with 9,000,045 more rows
```

The new column `date` is of a Date-Time class (`<dttm>`) called POSIXct, and it clearly makes it better to understand the time of rating.

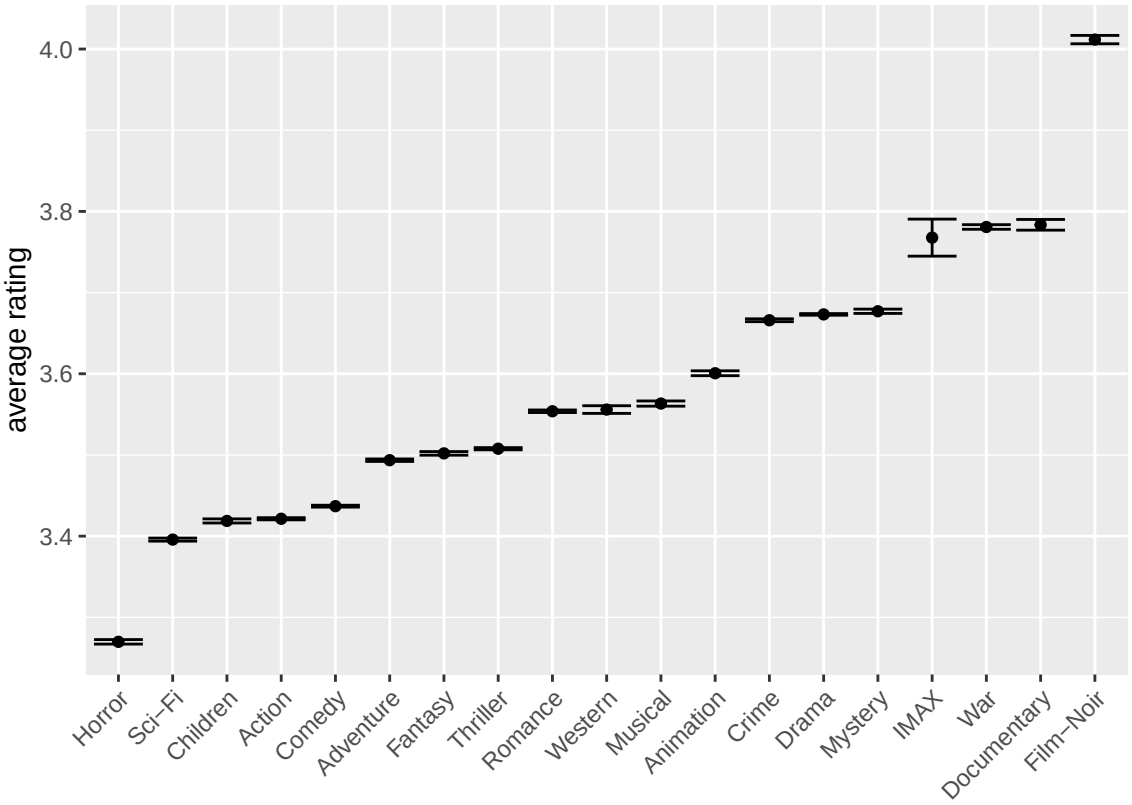

Figure 8: Average rating for each genre

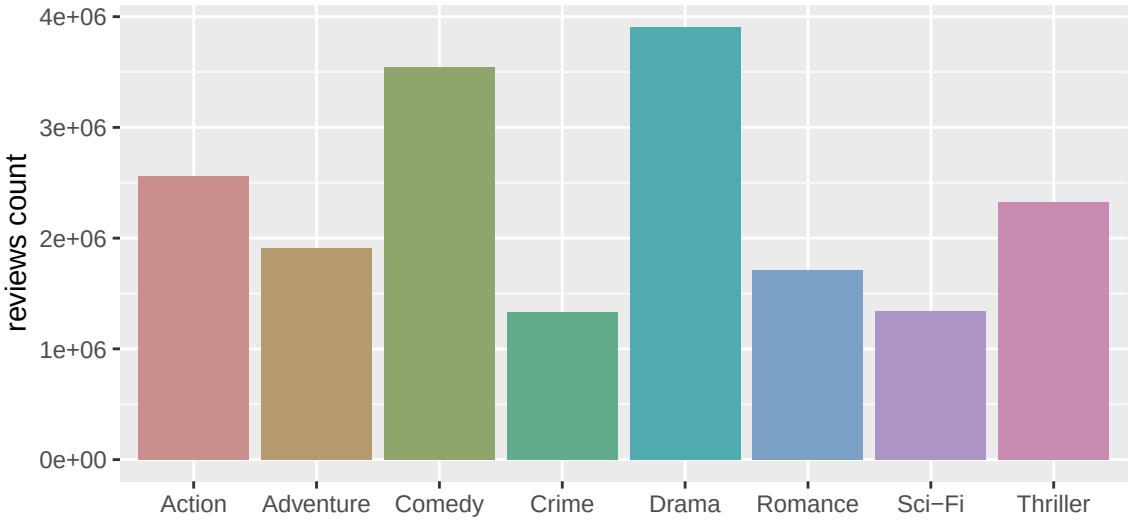

Figure 9: Number of reviews for each of the top 8 most reviewed genres

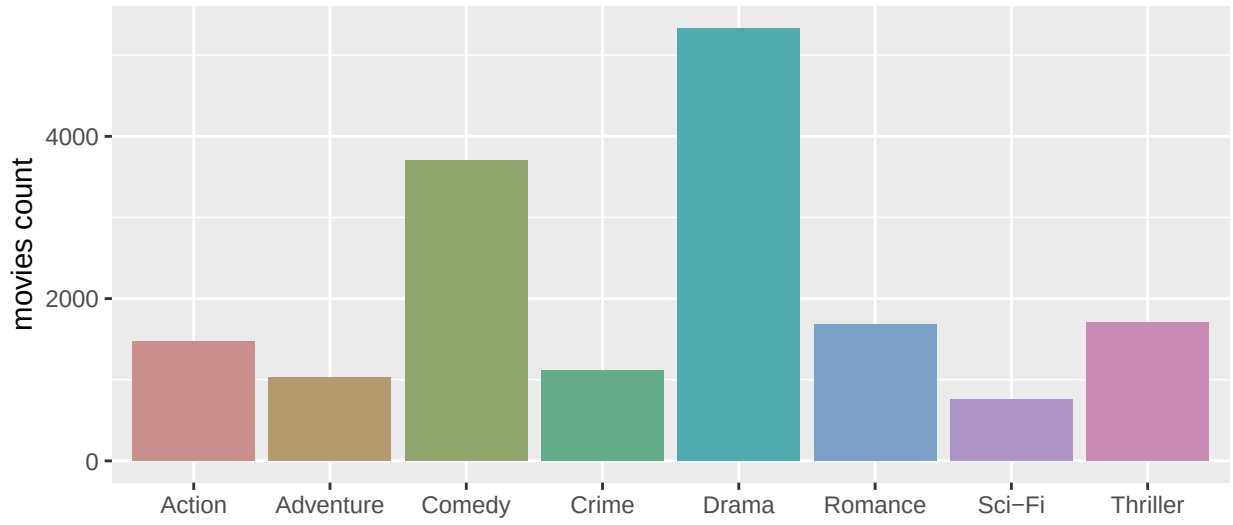

Figure 10: Number of movies that belongs to the top 8 most reviewed genres

We can now plot the time effect on the dataset, showing the influence the week a movie was rated on the average value. Along with ratings, we plot a LOESS smooth line to better show the relationship between variables.

```
edx %>% mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() + labs(y='average rating')
```

```
  geom_smooth()
```

```
>> geom_smooth: na.rm = FALSE, orientation = NA, se = TRUE
>> stat_smooth: na.rm = FALSE, orientation = NA, se = TRUE
>> position_identity
```

The review date definitely has some effect on ratings, but it isn't significant.

2.3 Feature engineering

Down the line, we will need all genres available in the dataset as binary values, so it is necessary to create columns for all 20 of them and apply for each row the values correspoding for the character column value. here we see the first 6 rows, with only the movieId, title and genres of the movies, so we can compare the genres further in the analysis.

```
edx[,c(2,5,6)] %>% as_tibble(.rows = 6)
```

```
>> # A tibble: 6 x 3
>>   movieId title                          genres
>>     <dbl> <chr>                          <chr>
>> 1     122 Boomerang (1992)               Comedy|Romance
```

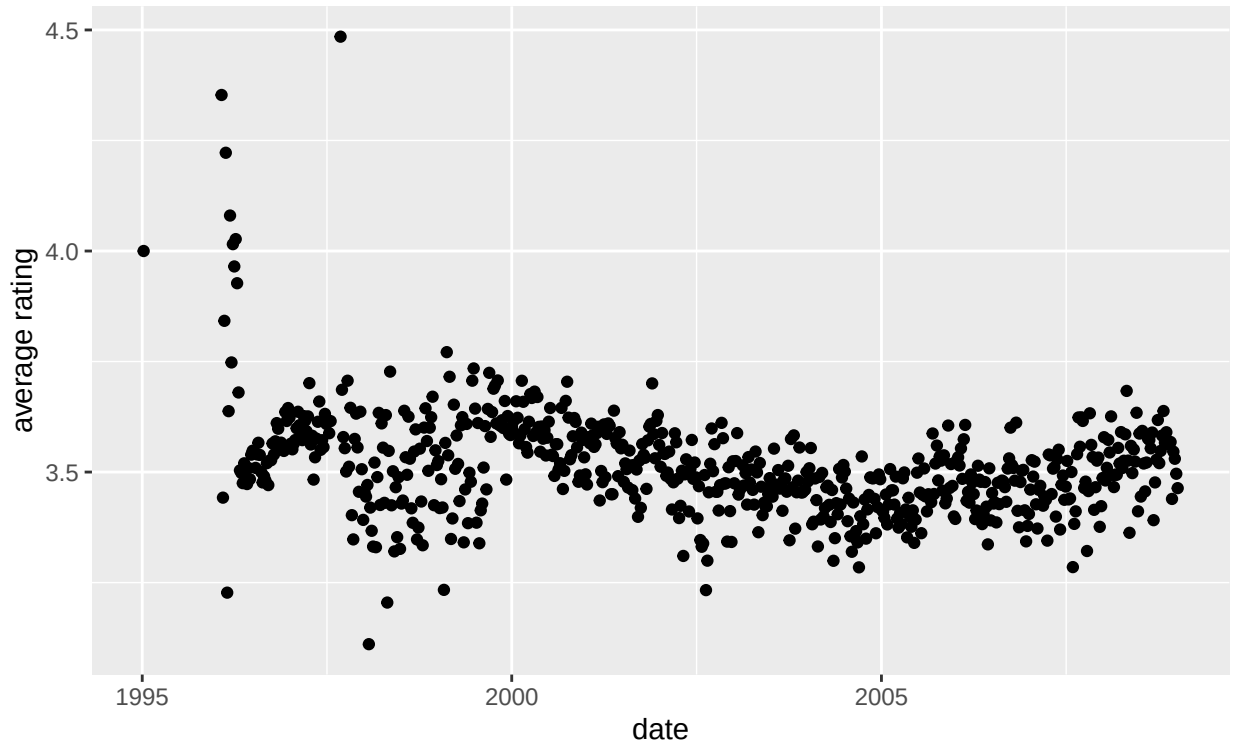

Figure 11: The review date effect on the average rating

```
>> 2     185 Net, The (1995)              Action|Crime|Thriller
>> 3     292 Outbreak (1995)              Action|Drama|Sci-Fi|Thriller
>> 4     316 Stargate (1994)              Action|Adventure|Sci-Fi
>> 5     329 Star Trek: Generations (1994) Action|Adventure|Drama|Sci-Fi
>> 6     355 Flintstones, The (1994)      Children|Comedy|Fantasy
```

We apply the transformation by doing the following `for()` command. We will also permanently remove the timestamp, title, genres and date columns, since we won't be needing them to train the prediction models.

```
for (var in all_genres) {
  edx <- edx %>% mutate(genre=ifelse(str_detect(genres,as.character(var)),1,0))
  colnames(edx)[ncol(edx)] <- as.character(var)
}
rm(var)
edx <- edx[,-(4:7),drop=FALSE]
edx[,4:23] %>% as_tibble(.rows=6)
```

```
>> # A tibble: 6 x 20
>>   Comedy Romance Action Crime Thriller Drama `Sci-Fi` Adventure Children Fantasy
>>    <dbl>   <dbl>  <dbl> <dbl>    <dbl> <dbl>    <dbl>     <dbl>    <dbl>   <dbl>
>> 1      1       1      0     0        0     0        0         0        0       0
>> 2      0       0      1     1        1     0        0         0        0       0
>> 3      0       0      1     0        1     1        1         0        0       0
>> 4      0       0      1     0        0     0        1         1        0       0
>> 5      0       0      1     0        0     1        1         1        0       0
>> 6      1       0      0     0        0     0        0         0        1       1
>> # ... with 10 more variables: War <dbl>, Animation <dbl>, Musical <dbl>,
```

```
>> #   Western <dbl>, Mystery <dbl>, Film-Noir <dbl>, Horror <dbl>,
>> #   Documentary <dbl>, IMAX <dbl>, (no genres listed) <dbl>
```

Comparing each row in this tibble above with the ones right before, it is possible to see that the genres in the first one are represented as 1s in the other, while genres that don't appear before are kept as zero values. The transformation occurred accurately.

2.4 Creating training and test sets

Before training any model, we need to create a train and a test set. We first set the seed to guarantee the reproducibility of the model, and we divide the `edx` dataframe into a 80-20% partition.

```
set.seed(123, sample.kind = "Rounding")
index <- createDataPartition(edx$rating, times=1, p=0.2, list= FALSE)
test <- edx[index,]
train <- edx[-index,]
```

To make sure the user and movie IDs in `test` are also in `train`, we apply the `semi_join()` function, just as we did in the beginning of the report for the first two datasets.

```
test <- test %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
```

2.5 Prediction models

Now that we have our training and test sets, we can start evaluating prediction models. We will begin by following the approach in Prof. Irrizarry's book, starting with just the Average Rating, going through the Movie Effect, Movie + User Effect and Regularized Movie + User Effects models. But we will also apply a method that includes the Genre Effect and one that perform Matrix Factorization. And to evaluate their quality, we will use the RMSE function.

If N is the number of user-movie combinations, $y_{u,i}$ is the rating for movie i by user u, and $\hat{y}_{u,i}$ is our prediction, then RMSE is defined as follows:

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

We then write the function as follows:

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

2.5.1 Average Rating Value

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

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

In this case, the least squares estimate of μ — the estimate that minimizes the root mean squared error — is the average rating of all movies across all users.

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

In this first model, we will predict all ratings with the average rating of the training set and compare it to the actual rating values in the test set (`RMSE(test$rating, mu)`).

Table 8: RMSE result for the Average Method

Method	RMSE
Just the Average	1.060013

we can observe that the value obtained was not that acceptable.

2.5.2 Movie Effects Model

We can improve our model by adding a term, b_i, that represents the average rating for movie i:

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

Here, b_i is the average of $Y_{u,i}$ minus the overall mean for each movie i. Note that because there are thousands of b's, the `lm()` function will be very slow or cause R to crash, so we don't recommend using linear regression to calculate these effects.

```
movie_avgs <- train %>% group_by(movieId) %>% summarize(b_i = mean(rating-mu))

predicted_ratings <- mu + test %>% left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>   0.500   3.217   3.586   3.512   3.878   4.750
```

From the summary above, we can see that the values obtained for predicted ratings range from 0.5 and 4.75, inside the real range (0.5-5). Calculating the RMSE:

```
>> [1] 0.9436204
```

Table 9: RMSE results with the Movie Effect Model

Method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204

This is a big improvement, but we can do better.

2.5.3 Movie + User Effects Model

We can further improve our model by adding b_u, the user-specific effect:

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

b_u is the average of $Y_{u,i}$ minus the overall rating for each movie i and the movie-specific coefficient. We will follow the same approach of the model before, as showed in the code chunk below:

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

predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%  pull(pred)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>> -0.7161  3.1353  3.5651  3.5127  3.9450  6.0547
```

Predicted ratings now have values ranging from -0.7161 to 6.0547, which is bigger than the real range. There are 4149 values under 0.5 and 4149 over 5. But what we really need to evaluate is the RMSE value by applying `RMSE(test$rating, predicted_ratings)`.

Table 10: RMSE results with the Movie and User Effects Model

Method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966

Another big improvement in the RMSE value. Let's proceed with other models.

2.5.4 Movie + User + Genre Effects Model

Now we try an approach, proposed by Prof. Irrizarry in his book, that adds the genre effect to it. It was presented in the book but not utilized. As it was presented, the formula is written as shown below:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + \varepsilon_{u,i}$$

with $x^k_{u,i} = 1$ if $g_{u,i}$ is genre k. We already determined $x_{u,i}$ values earlier by creating binary columns for each of the 20 genres available in the training and test data. Now we determine β_k values. We do it with the same approach used in the last two models, and each β_k is the rating $Y_{u,i}$ minus the overall average rating, the movie-specific coefficient b_i and user-specific coefficient b_u. To make all the process easier down the line, we will add b_i and b_u to the training and test sets, and then we calculate all β_k's.

```
train <- train %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')
test <- test %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')
rm(user_avgs,movie_avgs)

beta_k <- vector() # empty vector
for (i in seq_along(all_genres)) {
  b_value<- train %>%
    group_by(!!sym(all_genres[[i]])) %>%
    summarize(beta_k=mean(rating-mu-b_i-b_u)) %>%
    filter((.[1])==1) %>% .[[2]]
  beta_k <- append(beta_k, b_value)
}
rm(i,b_value)

df_beta<-data.frame(beta_k)
rownames(df_beta) <- all_genres
```

unquoting with !! and sym, beta value for all_genre[i], filter for all_genre[i]==1 and pulling beta value, appending it to beta vector. Transforming it into a dataframe.

Table 11: Genre-specific beta values

df_beta[1:10]		df_beta[11:20]	
genre	beta_k	genre	beta_k
Comedy	-0.0022244	War	0.0017305
Romance	-0.0035617	Animation	-0.0153201
Action	-0.0126152	Musical	-0.0102387
Crime	0.0079932	Western	-0.0066510
Thriller	-0.0047380	Mystery	0.0138908
Drama	0.0108468	Film-Noir	0.0306395
Sci-Fi	-0.0119931	Horror	0.0066441
Adventure	-0.0148381	Documentary	0.0624559
Children	-0.0241908	IMAX	-0.0016830
Fantasy	-0.0055682	(no genres listed)	0.1164495

The sum inside the prediction equation with genre effect is represented below:

$$\sum_{k=1}^{K} x_{u,i}\beta_k = x_{u,i}^{\{1\}}\beta_1 + x_{u,i}^{\{2\}}\beta_2 + ... + x_{u,i}^{\{20\}}\beta_{20}$$

We apply the sum above by performing a matrix multiplication between the test set genre columns (`test[,4:23]`) and all 20 β_k values calculated. This multiplication between a 1799965 per 20 matrix and a 20 per 1 vector results in a 1799965 per 1 vector. We then add this as a column to the test set, so we can calculate the predictions.

```
sum_x_beta <- as.matrix(test[,4:23])%*%as.matrix(df_beta)
sum_x_beta <- data.frame(sum_x_beta=sum_x_beta[,1])

test <- data.frame(test, sum_x_beta)
```

```
rm(df_beta,beta_k,sum_x_beta,all_genres)

predicted_ratings <- test %>%
  mutate(pred = mu + b_i + b_u + sum_x_beta) %>%
  pull(pred)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>> -0.7183  3.1260  3.5581  3.5058  3.9406  6.0652
```

Analyzing the values obtained, we see there are 216 values under 0.5 and 4157 predicted ratings over 5.

Table 12: RMSE results with the Movie, User and Genre Effects Model

Method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966
Movie + User + Genre Effects Model	0.8661932

There is very little improvement from the last model, while the work needed to obtain it was really arduous.

2.5.5 Regularized Movie + User Effects Model

We will now add regularization to one of the last models. Since the genre addition didn't have much effect on the RMSE and both β_k calculations and optimal lambda determination are costly, in a computational point of view, we use the Movie + User Effects model.

penalized least squares

$$\sum_{u,i} \left(y_{u,i} - \mu - b_i - b_u\right)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2\right)$$

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{\mu}\right)$$

$$\hat{b}_u(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{b}_i - \hat{\mu}\right)$$

```
train <- train %>% select(userId, movieId, rating)
test <- test %>% select(userId, movieId, rating)

lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
```

```
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(test$rating, predicted_ratings))
})
qplot(lambdas, rmses, xlab='lambda', ylab='RMSE')
```

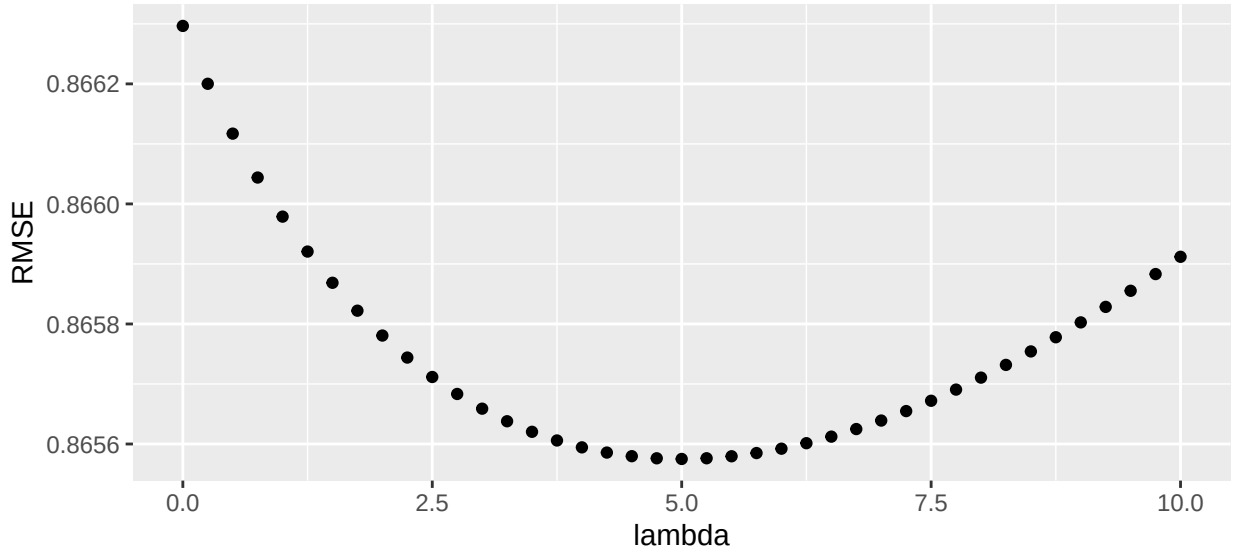

Figure 12: RMSE values per lambda

optimal lambda is 5 and its corresponding RMSE is 0.8655751.

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>> -0.5784  3.1407  3.5624  3.5100  3.9346  5.9789
```

137 and 2671.

Table 13: RMSE results with the Regularized Movie and User Effects Model

Method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966
Movie + User + Genre Effects Model	0.8661932
Regularized Movie + User Effects Model	0.8655751

2.5.6 Matrix Factorization with *recosystem* package

Now with a different approach, we try applying matrix factorization. P is the user index and Q the movie index. the *recosystem* easily performs the factorization.

```
train_dm <- data_memory(user_index = train$userId, item_index = train$movieId,
                        rating = train$rating, index1 = TRUE)
rm(train)

test_dm <- data_memory(user_index = test$userId, item_index = test$movieId,
                       index1 = TRUE)
test_rating <- test$rating
rm(test)

set.seed(123, sample.kind = "Rounding")
r <- Reco()
params = r$tune(train_dm, opts = list(dim = c(15, 20),
                                      costp_l1 = 0, #c(0, 0.1),
                                      costp_l2 = c(0.01, 0.1),
                                      costq_l1 = 0, #c(0, 0.1),
                                      costq_l2 = c(0.01, 0.1),
                                      lrate = c(0.075, 0.1), nthread = 2))

optimal_params = params$min

r$train(train_dm, opts = c(optimal_params, nthread = 1, niter = 20))
```

20, 0, 0.01, 0, 0.1, 0.1, 0.805147268198569

with the trained model, it is possible to predict the ratings for the test set.

```
predicted_ratings = r$predict(test_dm, out_memory())

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>  -1.594   3.057   3.543   3.472   3.963   6.300
```

859 and 6221.

```
>> [1] 0.7951871
```

3 Results

3.1 Training with full edX dataset

To guarantee the prediction model can be applied to the whole validation set, we will train with full edX set.

Table 14: RMSE results with the Matrix Factorization Model

Method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966
Movie + User + Genre Effects Model	0.8661932
Regularized Movie + User Effects Model	0.8655751
Matrix Factorization Model	0.7951871

```
train_dm <- data_memory(user_index = edx$userId, item_index = edx$movieId,
                        rating = edx$rating, index1 = TRUE)
rm(edx)

r$train(train_dm, opts = c(optimal_params, nthread = 1, niter = 20))
```

predicting for validation dataset, we get the following values:

```
validation_dm <- data_memory(user_index = validation$userId,
                             item_index = validation$movieId, index1 = TRUE)
validation_rating <- validation$rating

predicted_ratings = r$predict(validation_dm, out_memory())
rm(train_dm,validation)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>  -1.883   3.058   3.545   3.473   3.965   6.159
```

433 and 3586.

```
>> [1] 0.7887703
```

Table 15: RMSE result for the validation set with Matrix Factorization Model

Method	Validation RMSE
Matrix Factorization Model	0.7887703

4 Conclusion

We visualized the importance of variability between different movies and users, the genre and genre combinations effect on ratings, some not that significant. A lot of models were evaluated, the genre addition didn't have a great effect on the model so we didn't utilize it during the regularized trial, and the best and easiest model to apply was Matrix Factorization with *recosystem*.