Report on movie recommendation system

Chao Wu

2020/10/06

Executive summary

This project is about building a movie recommendation system using the MovieLens dataset. We start with the methods that were provided from previous course. We are required to add our own analysis on top of previous methods, and the final RMSE is expected to be less than 0.86490. I added my own analysis in addition to *Regularized Movie + User Effect Model*.

To achieve the project goal, I took the following steps:

- Generate the dataset using the code provided;
- Perform data wrangling: convert unit in seconds in the 'timestamp' column into a date/time format, extract movie release year from the 'title' column, and check for missing values;
- Split the edx set into training and test sets to avoid overfitting. So that I can use them to train and test intermediate models, as well as optimize the algorithm parameters;
- Explore the dataset: plotting and visualising the data help me to understand how the data is distributed and what the relationships are between the 'rating' and other data columns;
- Analyse 'genres', 'review_date', 'release_year' columns: there are genres, review year, and movie release year effects in these columns that can improve the *Regularized Movie + User Effect Model*;
- Analyse regularization: by adding penalty term lambda to regularize the genres, review year, and movie release year effects, the model has been further improved;
- Test the final model using the final hold-out test set.

The final model is *Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model*. This model adds regularized genres, review year, and movie release year effects, in addition to the *Regularized Movie + User Effect Model*. The final model has been evaluated using the final hold-out test set and the final RMSE achieves the project goal. Errors are reduced by adding more effects to the model, but the model becomes slower.

There are some limitations on the final model, as it doesn't count for the important fact that existing relationships or similarity between movies and users. To further improve the final model, I might consider building a hybrid model of combining a content-based filtering model with a collaborative filtering model using matrix factorization and principal component analysis (PCA) or singular value decomposition (SVD).

Introduction and Objectives

This project is about to create a movie recommendation system using the 10M version of the MovieLens dataset with "10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users" ("MovieLens 10M Dataset," 2009). The link of MovieLens 10M Dataset is: http://grouplens.org/datasets/movielens/10m/.

This project is built in addition to the *Regularized Movie + User Effect Model* that was provided from previous course for recommendation systems which "use ratings that users have given items to make specific recommendations." (Irizarry 2020).

We use root mean squared error (RMSE) as our loss function, to compare with different models and to see how well each model does.

The objectives of this project are to create a model to predict the movie ratings by adding our own analysis; and we expect the RMSE from the final hold-out test set to be less than 0.86490 when evaluating the final model.

Library

In this project, I used the following R libraries:

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

Data extraction

Data extraction:

To generate the dataset, the course provided the following code:

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################

# Note: this process could take a couple of minutes

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)

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 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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

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

```
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)
```

The edx set and validation set:

The code provided from the course has already separated the whole dataset into edx set and validation set. The edx set is treated as training set. The validation set is the final hold-out test set. I used validation set to evaluate the final model. It is expected to report RMSE from the validation set to be less than 0.86490.

Data cleansing and wrangling

The edx set:

From the table below, we can see what the dataset looks like. Each data row represents a rating provided by a user to a movie. The 'genres' column consists of combinations of genres where a movie being rated falls under. The 'timestamp' column is the unit in seconds since January 1, 1970 to the movie being rated. The 'title' column consists the movie title and the movie release year.

```
## # 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
```

Data wrangling:

As the unit of the 'timestamp' column is in seconds, to make it meaningful and understandable, I converted it to a date/time format and saved it into a new column 'review_date'.

To be able to analyse movie release year, I extracted the movie release year from the 'title' column and saved it into a new column 'release_year'.

I performed these data wrangling actions in both edx and validation sets.

In edx set, these two columns look like the following:

```
## # A tibble: 9,000,055 x 2
##    review_date         release_year
##    <dttm>                     <dbl>
##  1 1996-08-02 11:24:06         1992
##  2 1996-08-02 10:58:45         1995
##  3 1996-08-02 10:57:01         1995
##  4 1996-08-02 10:56:32         1994
##  5 1996-08-02 10:56:32         1994
##  6 1996-08-02 11:14:34         1994
##  7 1996-08-02 11:00:53         1994
##  8 1996-08-02 11:21:25         1994
##  9 1996-08-02 11:01:47         1994
## 10 1996-08-02 11:16:36         1994
## # ... with 9,000,045 more rows
```

Missing values:

There is 0 missing value in edx set and 0 missing value in validation set, which means we don't need to fill in any values.

Dataset and Exploration

Training set and test set:

To avoid overfitting, I further split the edx set into training set and test set. I train and test intermediate models, as well as optimize the algorithm parameters, only using training set and test set that are split from the edx set. The test set is 20% of the edx set, and the rest becomes training set.

Data exploration:

To help me understand the data a bit more, I plot to visualise how the data is distributed and what the relationships are between the 'rating' and other data columns.

The average rating of all movies across all users in the edx dataset is:

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

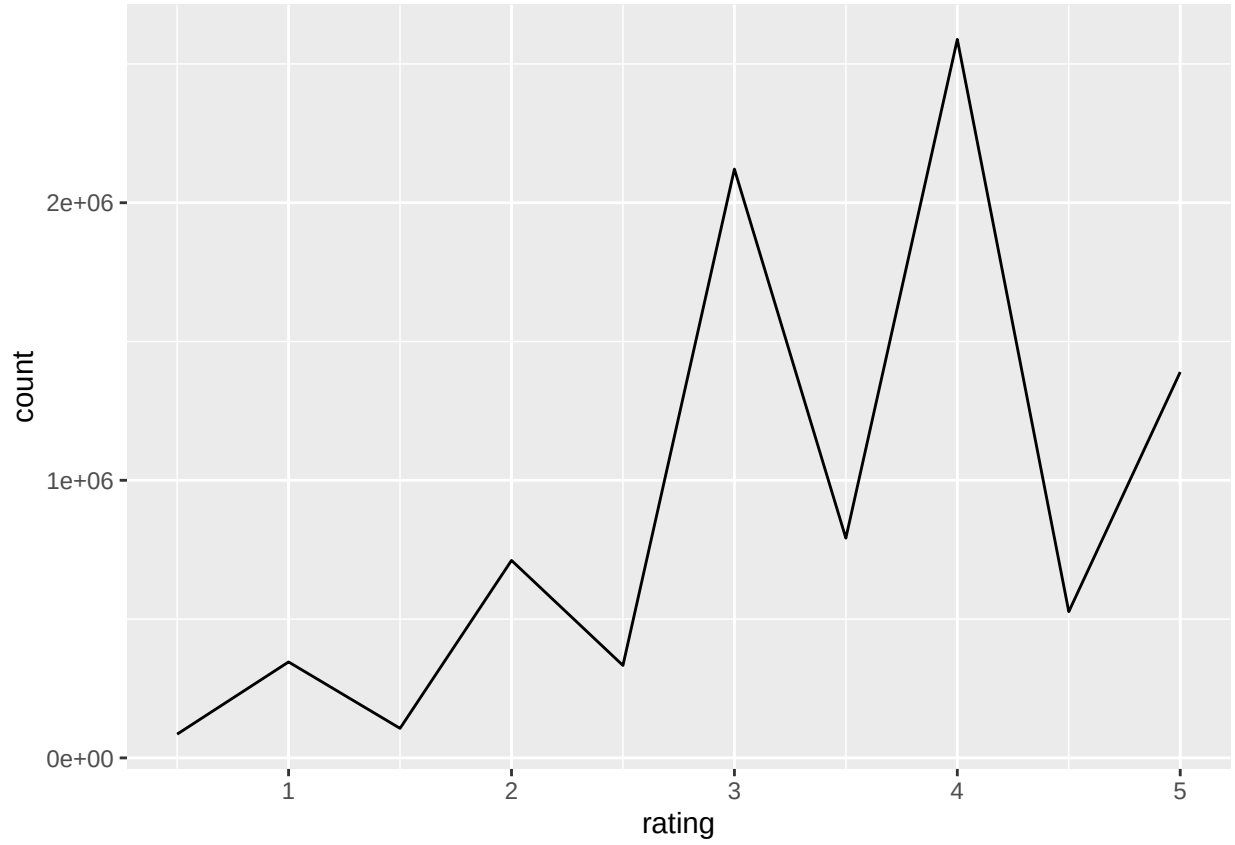
There is not zero was given as rating in the edx dataset.

```
## [1] 0
```

The most popular rating given by users is 4. Below are the five most given ratings in order from most to least.

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

Half star ratings are less common than whole star ratings.



Movie "Pulp Fiction" has the greatest number of ratings.

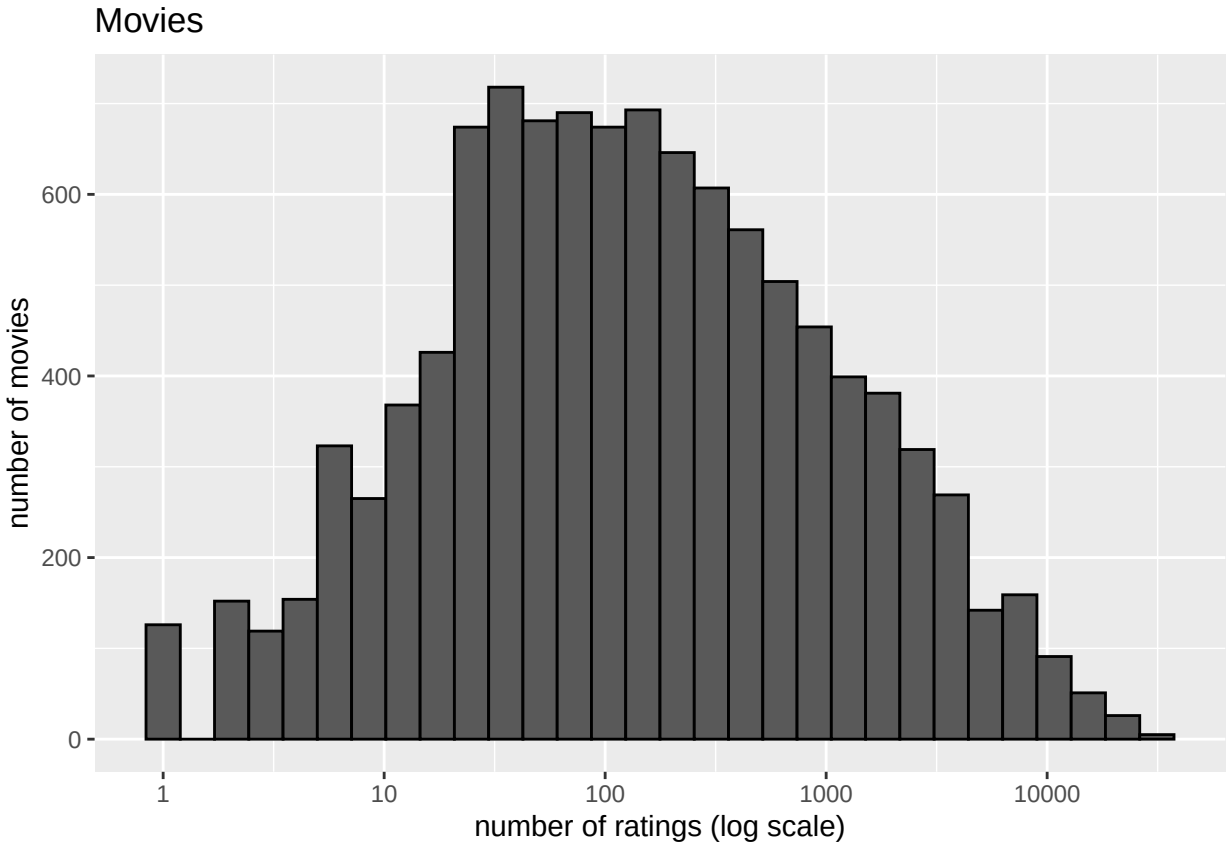
```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                  count
##      <dbl> <chr>                                                  <int>
```

```
##  1     296 Pulp Fiction (1994)                                          31362
##  2     356 Forrest Gump (1994)                                          31079
##  3     593 Silence of the Lambs, The (1991)                             30382
##  4     480 Jurassic Park (1993)                                         29360
##  5     318 Shawshank Redemption, The (1994)                             28015
##  6     110 Braveheart (1995)                                            26212
##  7     457 Fugitive, The (1993)                                         25998
##  8     589 Terminator 2: Judgment Day (1991)                            25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                             24284
## # ... with 10,667 more rows
```

The table below shows the number of unique users that provide ratings, for how many unique movies they provided, and the number of unique genres in the edx set. It shows that a movie can be rated by a number of users, a user can rate a number of movies. In the 'genres' column, if we treat the combination of genres in each data row as a big genre, then there are over 700 genres in that column.

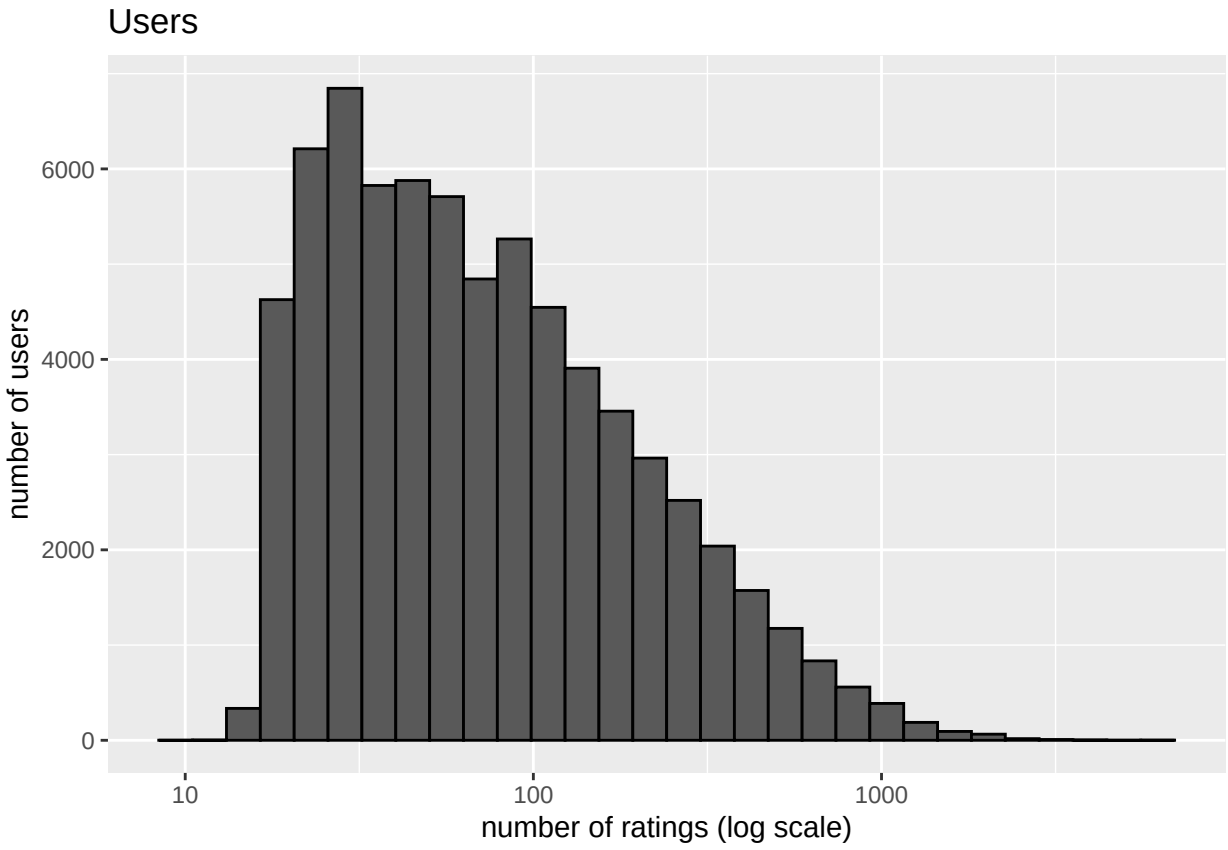
```
##   n_users n_movies n_genres
## 1   69878    10677      797
```

From Figure 1.1, we can see that different movies have been rated differently.



(Figure 1.1)

From Figure 1.2, we can see that different users rated movies differently.



(Figure 1.2)

Analysis

Methods/models provided from previous course:

Our previous course provided four methods step by step.

Method 1: Just the average

We start with a method that assumes the same rating for all movies and all users, with all the differences explained by random variation. The method can be represented like this:

Y_ui = mu + error_ui

Where:

- mu is the average rating of all movies across all users;
- error_ui is independent errors sampled from the same distribution centered at zero.

The RMSE result shows how well this model does.

Method	Test_on	Parameter	Value	RMSE
Just the average	test set	NA	NA	1.060561

Method 2: Movie Effect Model

By analysing the 'movieId' column, we know that different movies are rated differently (Figure 1.1). We added movie effect in addition to *Method 1: Just the average* to improve it. The new method can be represented like this:

Y_ui = mu + b_i + error_ui

Where b_i is the average rating for movie i, the least squares estimates.

The RMSE results show that *Method 2: Movie Effect Model* improves *Method 1: Just the average*.

Method	Test_on	Parameter	Value	RMSE
Just the average	test set	NA	NA	1.0605613
Movie Effect Model	test set	NA	NA	0.9439868

Method 3: Movie + User Effects Model

By analysing the 'userId' column, we know that different users rated movies differently (Figure 1.2). We added user effect in addition to *Method 2: Movie Effect Model* to improve it. The new method can be represented like this:

Y_ui = mu + b_i + b_u + error_ui

Where b_u is the average rating by user u to movie i, the least squares estimates.

The RMSE results show that *Method 3: Movie + User Effects Model* improves *Method 2: Movie Effect Model*.

Method	Test_on	Parameter	Value	RMSE
Just the average	test set	NA	NA	1.0605613
Movie Effect Model	test set	NA	NA	0.9439868
Movie + User Effects Model	test set	NA	NA	0.8666408

Method 4: Regularized Movie + User Effect Model

Regularization is introduced to improve *Method 3: Movie + User Effects Model*. "Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes." (Irizarry 2020). For example, when the best or worst movies were rated by a small number of users, it is more likely to have larger estimates of movie effects. These estimates are noisy estimates that we should eliminate in our model. The new method can be represented like this:

Y_ui = mu + reg(b_i) + reg(b_u) + error_ui

Where reg(b_i) and reg(b_u) are the penalized estimates of movie and user effects, respectively.

By adding the penalty term lambda to the movie and user effects, regularized estimates largely improve least squares estimates.

The RMSE results approves that *Method 4: Regularized Movie + User Effect Model* improves *Method 3: Movie + User Effects Model*.

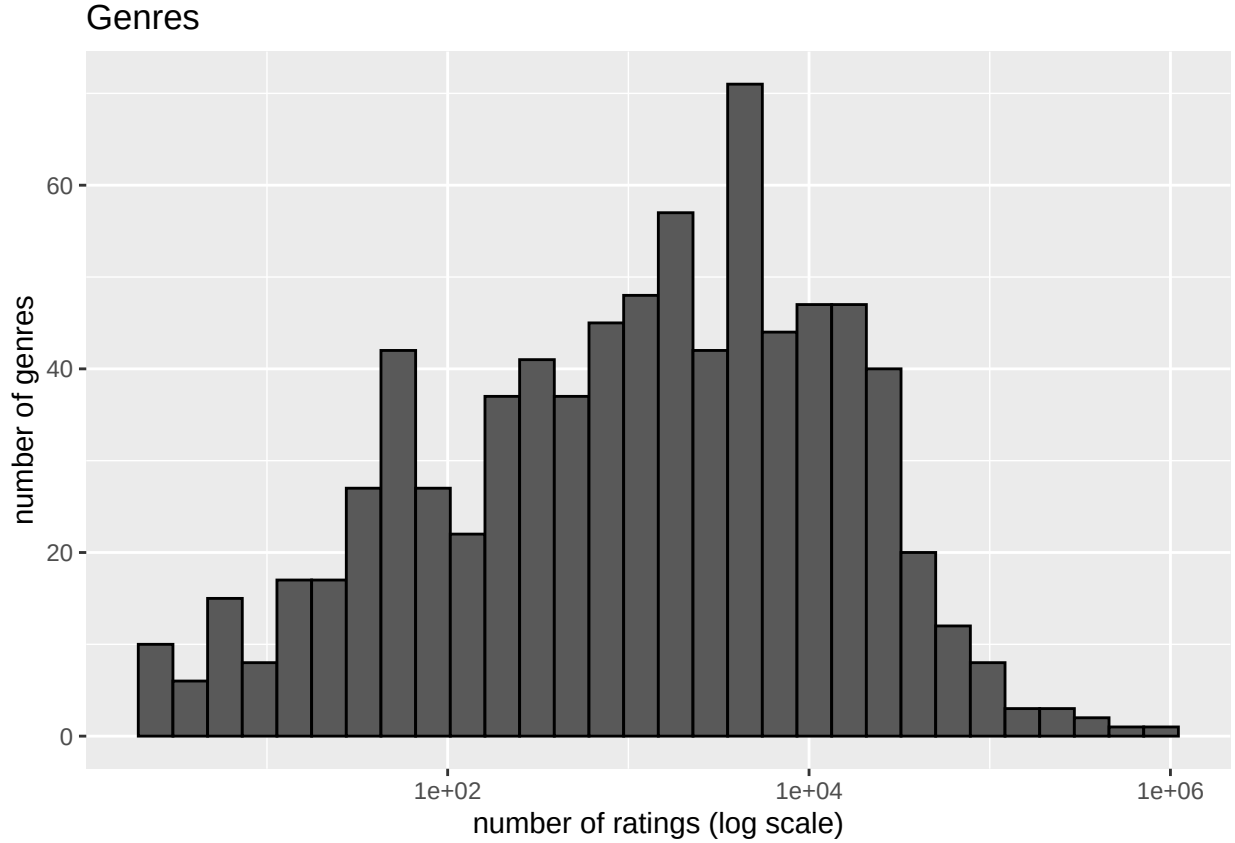
Method	Test_on	Parameter	Value	RMSE
Just the average	test set	NA	NA	1.0605613
Movie Effect Model	test set	NA	NA	0.9439868
Movie + User Effects Model	test set	NA	NA	0.8666408
Regularized Movie + User Effect Model	test set	lambda	5	0.8659628

Analysis on genres, review date, and release year:

I start my analysis from *Method 4: Regularized Movie + User Effect Model*. To further improve this method, I analysed the 'genres', 'review_date', and 'release_year' columns.

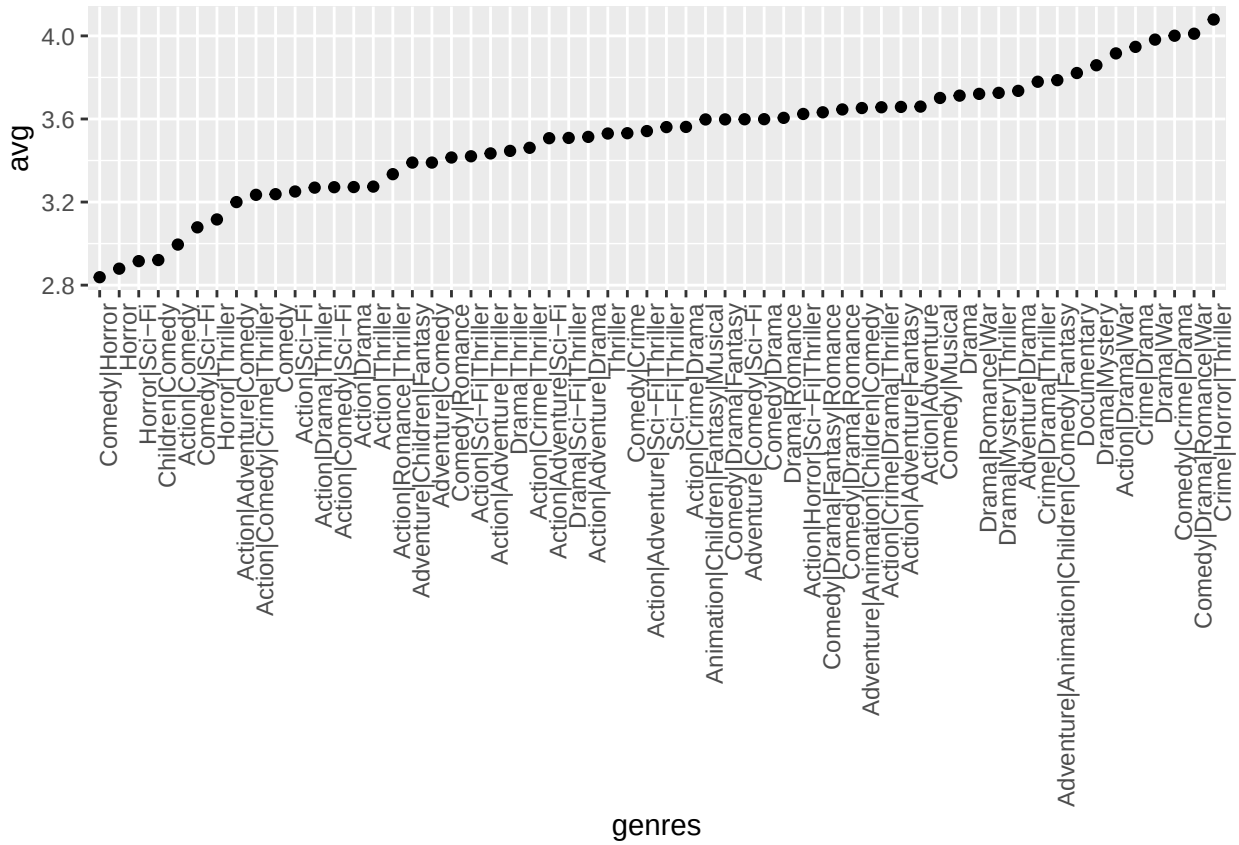
Looking at 'genres' column

I treat the combination of genres in each data row as a big genre. From Figure 1.3, we can see that different number of ratings are in different genres.



(Figure 1.3)

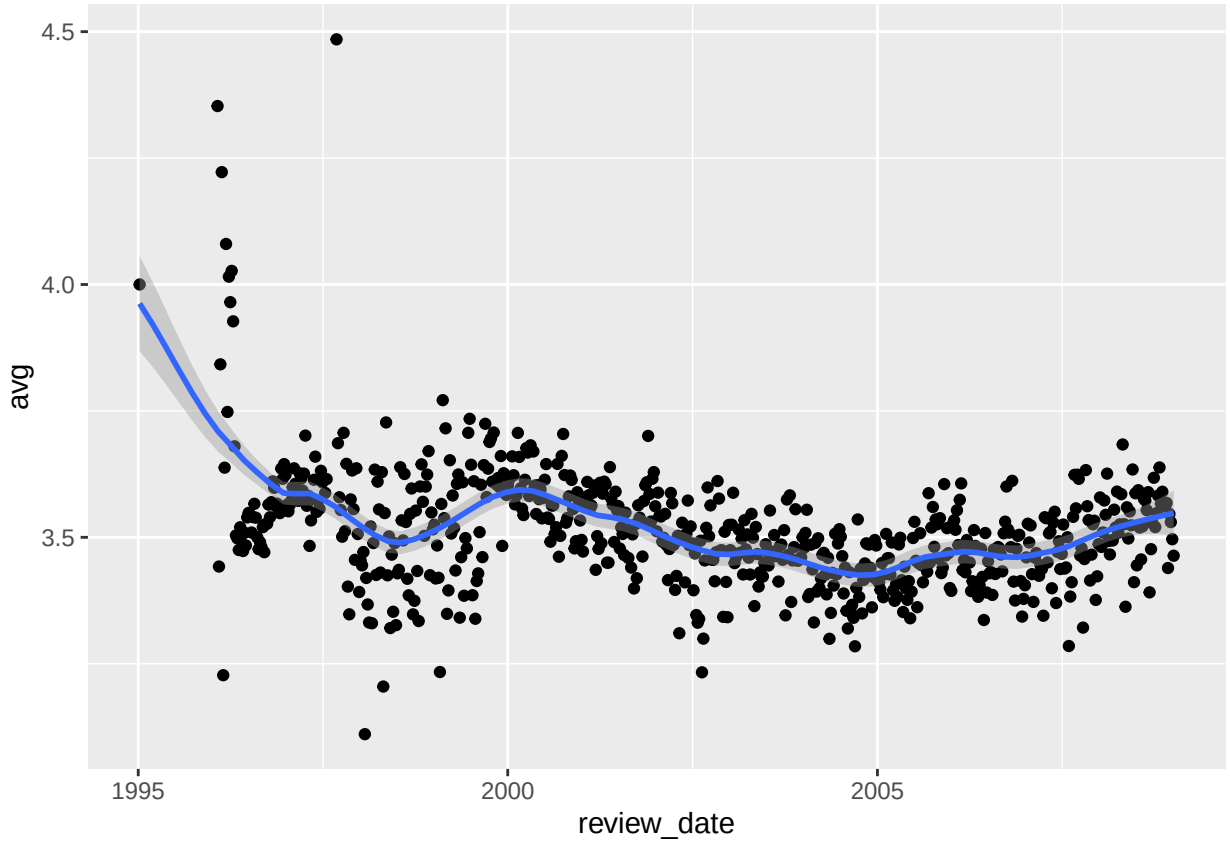
I compute the average rating for each genre and plot this average against each genre. Figure 1.4 shows a strong genres effect between genres and the average ratings.



(Figure 1.4, as there are more than 700 genres, to make the labels in the x-axis readable, the figure only shows the genres with greater than or equal to 30000 ratings.)

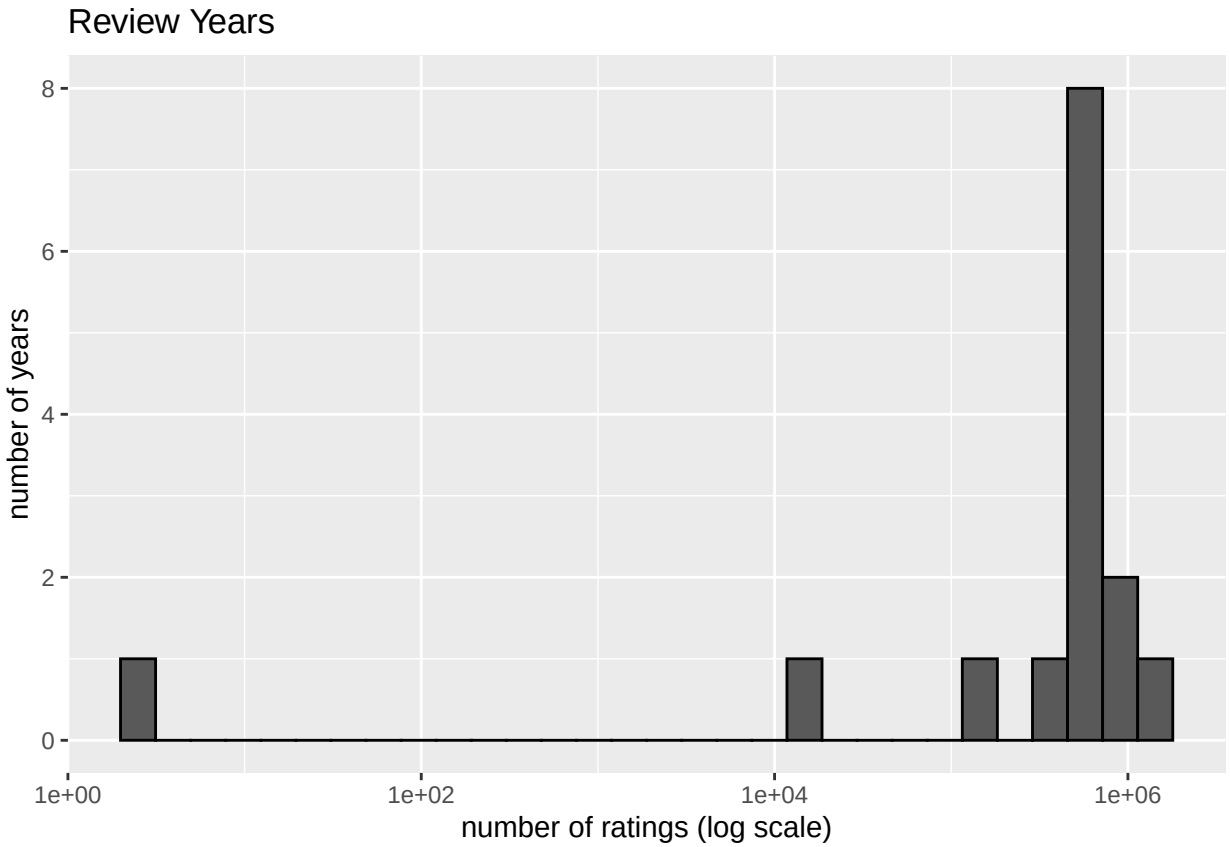
Looking at 'review_date' column

I compute the average rating for each week and plot this average against date. The plot in Figure 1.5 shows some effect of time, but not a strong effect, if I only look at the cluster that most of the data points group together.



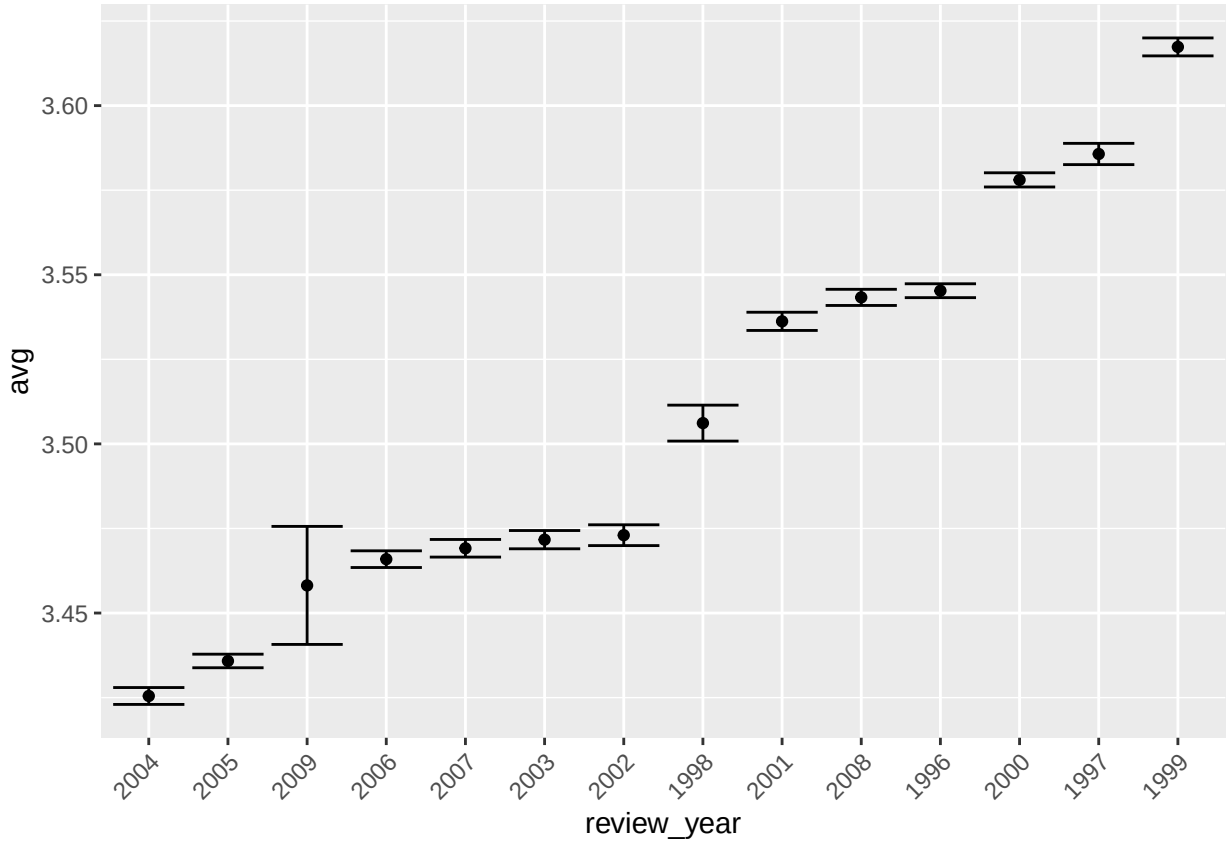
(Figure 1.5)

I also look at each year in the 'review_date' column. Figure 1.6 shows that there are a lot of ratings given in some years, but there are only a few ratings in a particular year. There are big intervals among different years.



(Figure 1.6)

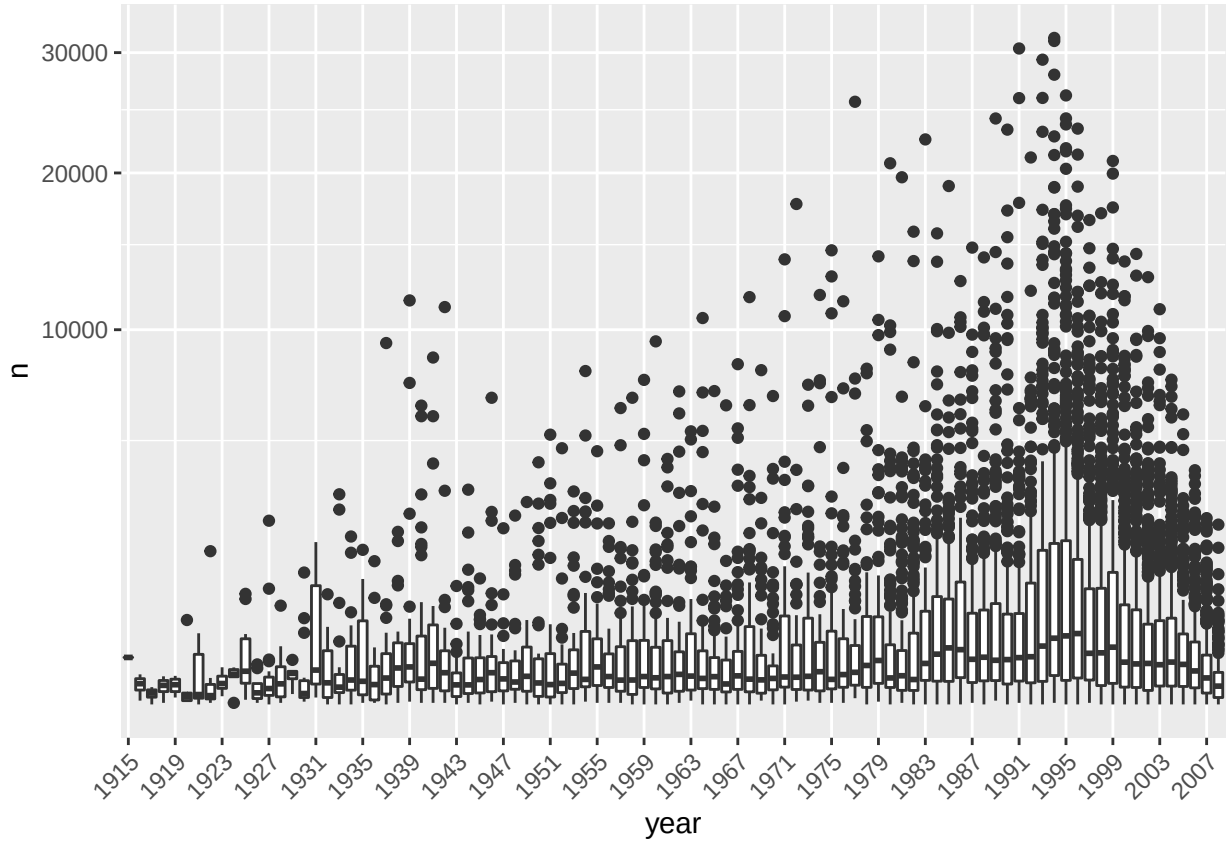
I compute the average rating and standard error for each review year and plot this average against year. The plot in Figure 1.7 shows some effect of review year.



(Figure 1.7, as there are more than 20 different years, to make the labels in the x-axis readable, the figure only shows the years with greater than or equal to 1000 ratings.)

Looking at 'release_year' column

I compute the number of ratings for each movie and then plot it against the year the movie came out. From Figure 1.8, we can see that, movies that came out after 1993 get more ratings. But starting in 1993, the number of ratings decreases with year.

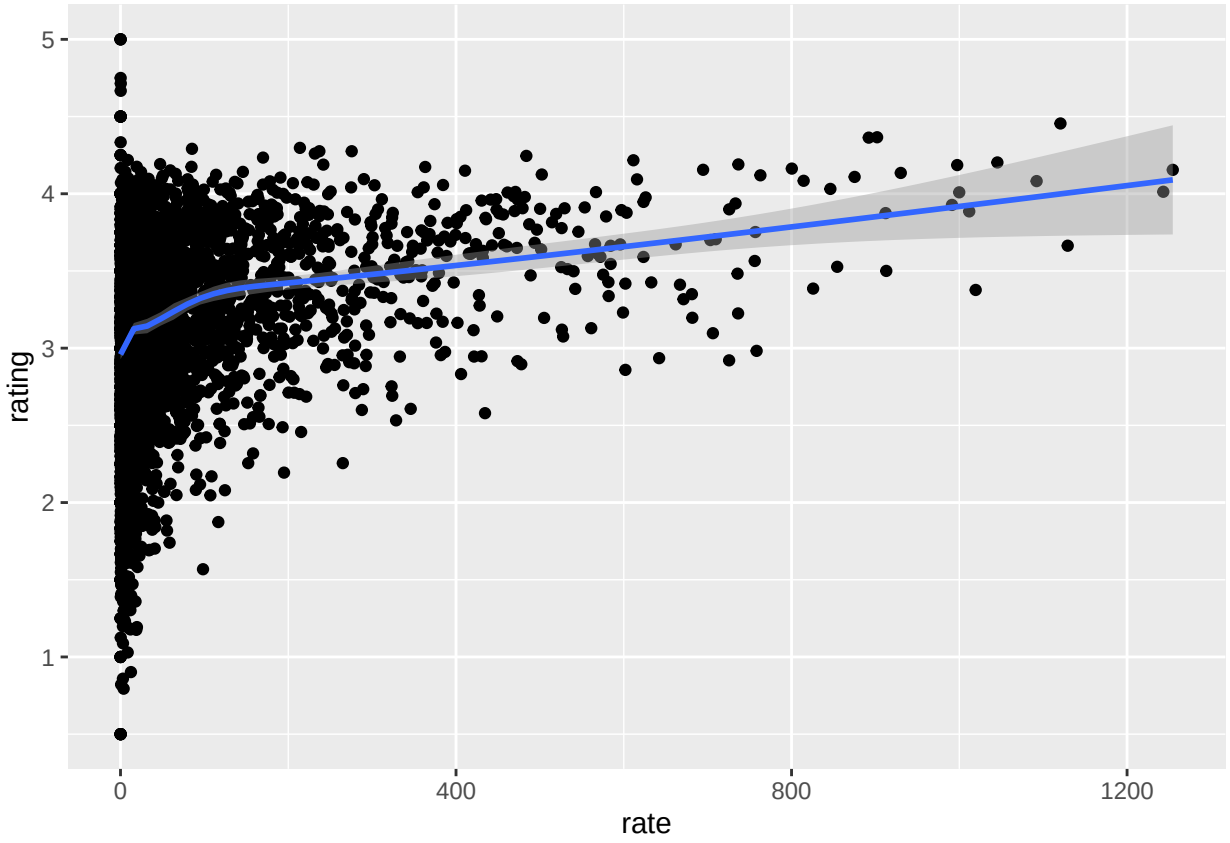


(Figure 1.8)

The table below shows the top 25 movies with the highest average number of ratings per year. The more recent a movie is, the less time users have had to rate it.

```
## # A tibble: 25 x 5
##    movieId     n years rating  rate
##      <dbl> <int> <dbl>  <dbl> <dbl>
##  1     296 31362    25   4.15 1254.
##  2     356 31079    25   4.01 1243.
##  3     480 29360    26   3.66 1129.
##  4     318 28015    25   4.46 1121.
##  5     110 26212    24   4.08 1092.
##  6    2571 20908    20   4.20 1045.
##  7     780 23449    23   3.38 1020.
##  8     150 24284    24   3.89 1012.
##  9     457 25998    26   4.01  999.
## 10    2858 19950    20   4.19  998.
## # ... with 15 more rows
```

From Figure 1.9, we can see that the most frequently rated movies have above average ratings. As more people watch popular movies. The plot shows strong effect of movie release year.



(Figure 1.9)

Method 5: (Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model

From the analysis above, the data shows genres, review year, and movie release year effects. After adding the genres, review year, and movie release year effects to *Method 4: Regularized Movie + User Effect Model*, the new method can be represented like this:

Y_ui = mu + reg(b_i) + reg(b_u) + b_g + b_t + b_y + error_ui

Where:

- b_g is the average rating by user u to movie i in genre g, the least squares estimates;
- b_t is the average rating by user u to movie i in genre g rated in year t, the least squares estimates;
- b_y is the average rating by user u to movie i released in year y in genre g in year t, the least squares estimates.

The RMSE results show that *Method 5: (Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model* improves *Method 4: Regularized Movie + User Effect Model*.

Method	Test_on	Parameter	Value	RMSE
Just the average	test set	NA	NA	1.0605613
Movie Effect Model	test set	NA	NA	0.9439868

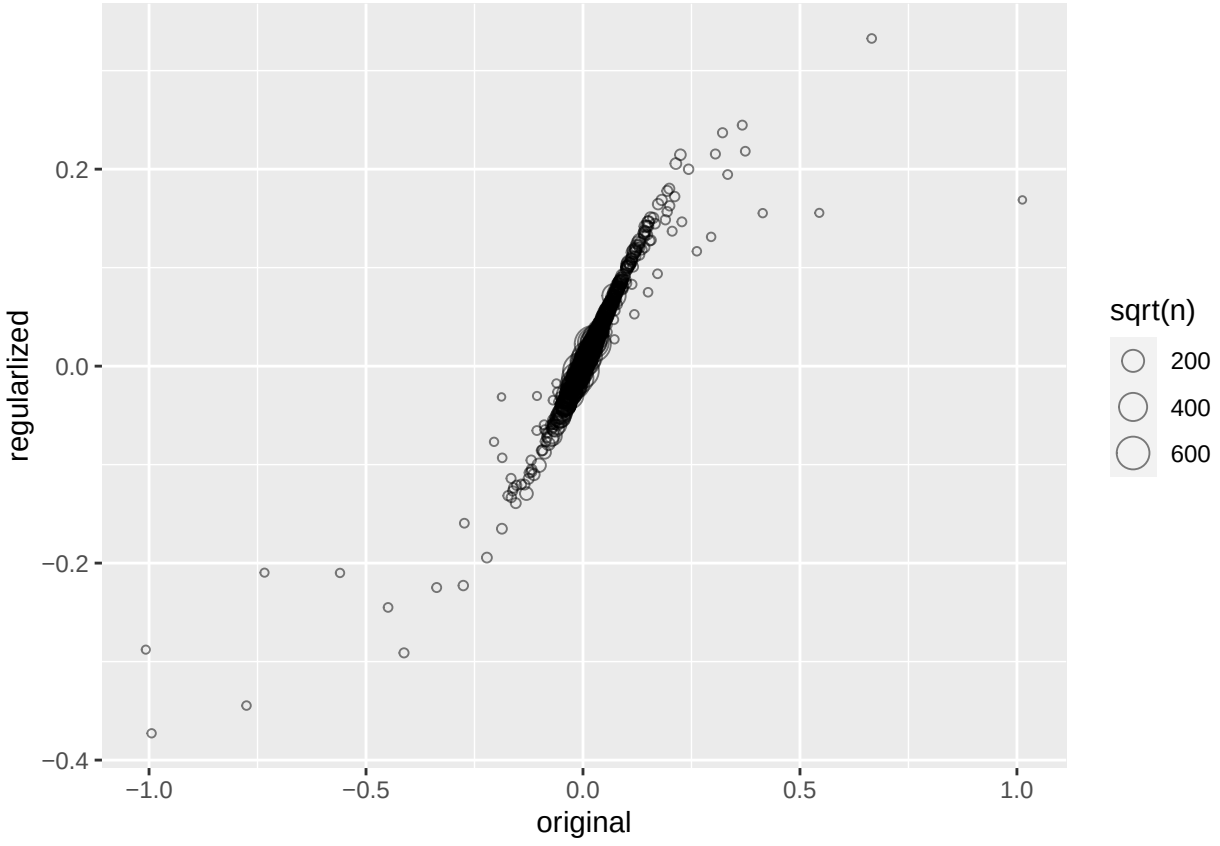
Method	Test_on	Parameter	Value	RMSE
Movie + User Effects Model	test set	NA	NA	0.8666408
Regularized Movie + User Effect Model	test set	lambda	5	0.8659628
(Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model	test set	lambda	5	0.8653491

Regularization:

To see whether regularization can improve *Method 5: (Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model* even further and by how much the method can be improved, I analysed regularization on genres, review year, and movie release year effects, separately.

Adding regularization to genres effect:

I added penalty term lambda to regularize the genres effect. To compare this method with *Method 5: (Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model* using RMSE, I used the lambda value that gave the lowest RMSE to the model of Method 5. From Figure 2.0, we can see that when n is small, the values are shrinking more towards zero. The RMSE result approves that regularized estimates of genres effect improves the least squares estimates.

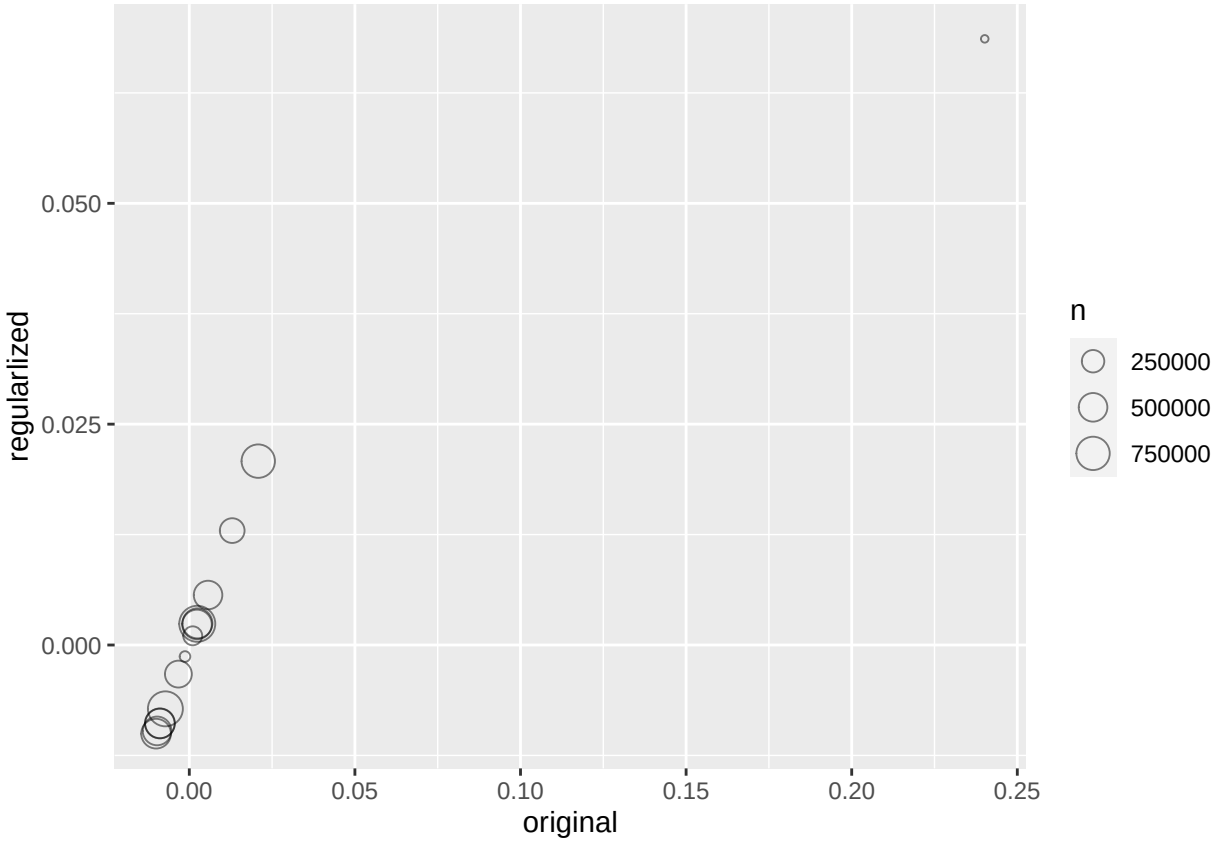


Method	RMSE
Genres average effect	0.8653491
Regularized Genres effect	0.8653473

(Figure 2.0)

Adding regularization to review year effect:

I added the penalty term lambda to regularize the review year effect, like what I did to the genres effect. From Figure 2.1, we can see that regularization has very minor (almost no) impact on the review year effect. The RMSE result approves this.



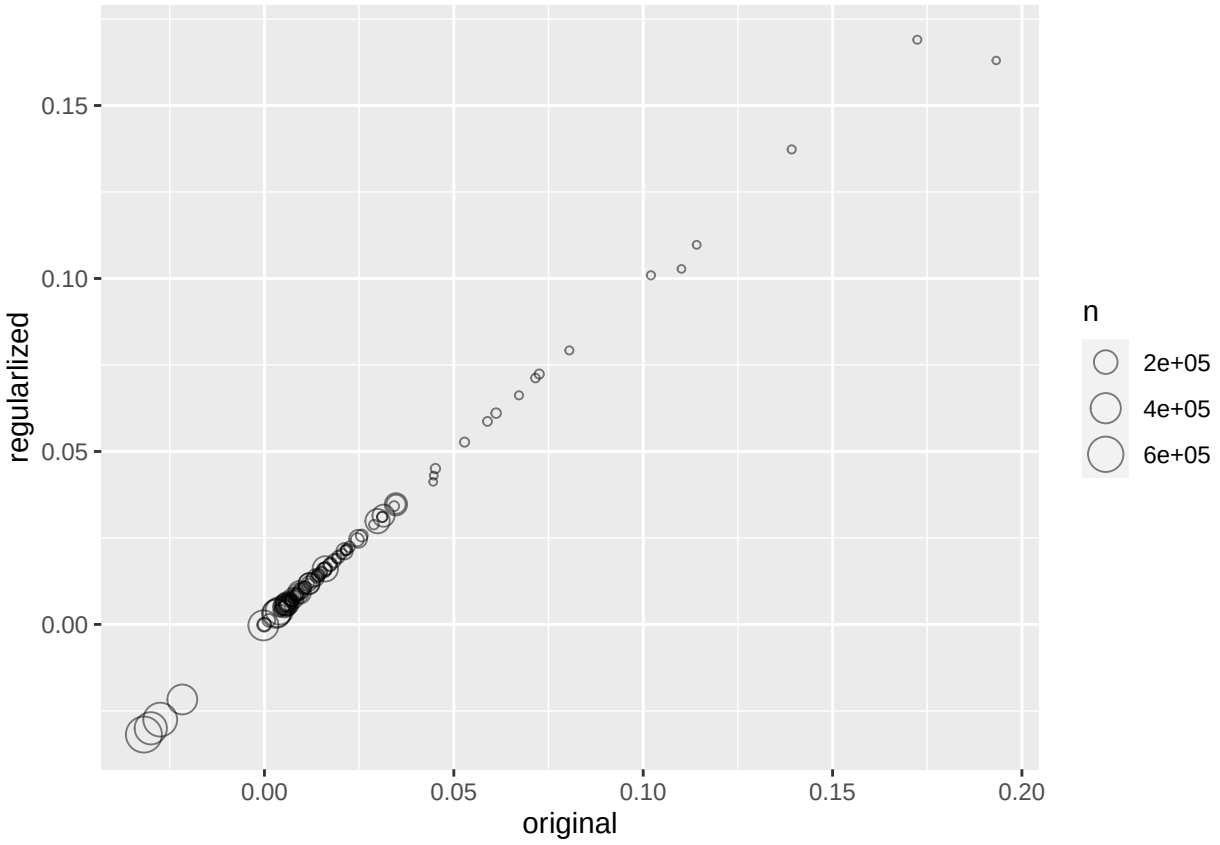
Method	RMSE
Review Year average effect	0.8653491
Regularized Review Year effect	0.8653491

(Figure 2.1)

Adding regularization to movie release year effect:

I added the penalty term lambda to regularize the movie release year effect, like what I did to the genres effect. From Figure 2.2, we can see that regularization has minor impact on the movie release year effect.

The RMSE result approves this.



Method	RMSE
Release Year average effect	0.8653491
Regularized Release Year effect	0.8653490

(Figure 2.2)

Method 6: Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model

The regularization analysis approves that *Method 5: (Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model* can be further improved by applying regularization to the genres, review year, and movie release year effects. The new method can be represented like this:

Y_ui = mu + reg(b_i) + reg(b_u) + reg(b_g) + reg(b_t) + reg(b_y) + error_ui

Where reg(b_g), reg(b_t), and reg(b_y) are the penalized estimates of genres, review year, and movie release year effects, respectively.

The RMSE results show that *Method 6: Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model* improves *Method 5: (Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model*.

Method	Test_on	Parameter	Value	RMSE
Just the average	test set	NA	NA	1.0605613
Movie Effect Model	test set	NA	NA	0.9439868
Movie + User Effects Model	test set	NA	NA	0.8666408
Regularized Movie + User Effect Model	test set	lambda	5	0.8659628
(Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model	test set	lambda	5	0.8653491
Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model	test set	lambda	5	0.8653473

Parameter tuning:

The penalty term lambda is a tuning parameter. I use cross-validation to choose it. To avoid overfitting, I have further split the edx set into training and test sets, and do the following steps:

- generate a sequence of lambda;
- train an intermediate model using the training set and a lambda in the sequence;
- generate the predicted ratings using the test set;
- calculate RMSE using the predicted ratings and the true ratings from the 'rating' column in the test set;
- repeat the steps of training and prediction for each lambda in the sequence;
- finally, pick up the lambda that minimises the RMSE.

The minimised RMSE for that intermediate model will be used to compare with other intermediate models.

Final model:

The RMSE results show that model of *Method 6: Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model* gives the lowest RMSE. So, the final model is *Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model*.

The method of the final model can be represented like this:

Y_ui = mu + reg(b_i) + reg(b_u) + reg(b_g) + reg(b_t) + reg(b_y) + error_ui

Where:

- mu is the average rating of all movies across all users;
- reg(b_i), reg(b_u), reg(b_g), reg(b_t), and reg(b_y) are the penalized estimates of movie, user, genres, review year, and movie release year effects, respectively;
- error_ui is independent errors sampled from the same distribution centered at zero.

In the final model, I used the whole edx set as training set, and the validation set as test set. The parameter lambda is generated from the model of *Method 6: Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model*, that has been optimised for that model.

Results and Performance

Final RMSE:

To calculate the final RMSE, I used the 'rating' column from the validation set as true ratings to compare with the predicted ratings from the final model.

The last RMSE result shows that the final RMSE is less than 0.86490. This implies that the final model meets the project goal.

Method	Test_on	Parameter	Value	RMSE
Just the average	test set	NA	NA	1.0605613
Movie Effect Model	test set	NA	NA	0.9439868
Movie + User Effects Model	test set	NA	NA	0.8666408
Regularized Movie + User Effect Model	test set	lambda	5	0.8659628
(Regularized Movie + User) + Genres + Review Year + Movie Release Year Effect Model	test set	lambda	5	0.8653491
Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model	test set	lambda	5	0.8653473
Final model: Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model	validation set	lambda	5	0.8641664

Performance:

RMSE has been reduced by adding more effects to the model. Regularization also improves the model. However, the more effects add to the model, the slower the model is. *Method 6: Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model* improves *Method 4: Regularized Movie + User Effect Model*, but the model of Method 6 is slower than of Method 4.

Conclusion

Limitations:

There are limitations on the final model. The final model doesn't count for the important fact that existing relationships or similarity between movies and users. For example, some users might rate something similar; users might have their preferences and tastes; some movies might have similar rating patterns.

Future work:

To further improve the final model, there are a few methods to be considered:

- Building a collaborative filtering model using matrix factorization: this relates to factor analysis; we try to explain more of variance by analysing vectors of factors. PCA or SVD can be used to estimate factors from the data. The limitation of this model would be due to data sparsity, for example, some users only rate a few numbers of movies.
- Building a content-based filtering model based on user's preferences and tastes, this type of model could solve data sparsity issue.
- Building an ensemble or hybrid model that combine multiple models into one model to improve predictions.

Conclusion:

This project is built in addition to the *Regularized Movie + User Effect Model*. I generated the dataset; converted data into a different format; plotted to visualise the data; and split the edx set into training and test sets. I used these training and test sets to train and test intermediate models, and also optimise the tuning parameter lambda. By analysing the 'genres', 'review_date', 'release_year' columns, I found out that there are genres, review year, and movie release year effects in these columns. I added genres, review year, and movie release year effects to the model; and then applied regularization to these effects. The final model is *Regularized Movie + User + Genres + Review Year + Movie Release Year Effect Model*. The RMSE results from intermediate models approve that the *Regularized Movie + User Effect Model* has been improved. The final RMSE from the final model is less than 0.86490, which meets the project goal. To further improve the final model, I might consider a hybrid model of combining a content-based filtering model with matrix factorization and PCA or SVD.

References

MovieLens 10M Dataset. (2009, January). Retrieved from grouplens: https://grouplens.org/datasets/movielens/10m/

Irizarry, R. A. (2020, September 06). Retrieved from Github: https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems