MovieLens - Moive Ratings Prediction

Yi Zheng

2020-12-17

1 Introduction

This project predicts users' movie ratings. The files are downloaded from (http://files.grouplens.org/datasets/movielens/ml-10m.zip), which contains three datsets.

Ratings.dat is the primary dataset file, which contains **10,000,054** rows, each row represents a user's specific rating of the movie. There are **4** columns in this dataset. The first column is the ID of the user who rated the movie, the second column is the ID of the movie being rated, the third column is the actual rating number. The range is from **0.5** to **5**. The last column is the date and time of the rating.

These rating data apply to **10677** different movies by **69878** users. The specific movie information is stored in the file **movies.dat**. There are **3** columns in this dataset. The first column is the movie ID, the second column is the movie name and its release year, and the last column is the movie genre. The last file **tags.dat** contains tag information that will not be used in this project.

The goal of the project is to use the given information (such as user ID, movie ID, rated date and time) to successfully predict the movie ratings and reduce the **RMSE** score under **0.86490** generated based on different prdication models.

To achieve the goal, multiple data analysis steps need to be performed: First, the original datasets will be prepared, explored, and preprocessed; then, visuals will be drawn to help us better understand the data and gain helpful insights; finally, different prediction models will be implemented, and the optimal model with the smallest **RMSE** score will be determined and reported.

2 Analysis

2.1 Data Preparation

The project requirements have provided code procedures for data preparation and datasets generation. We first check whether the required libraries are installed, and then import them all.

The code has been modified and can be used in **R 4.0** or later.

```
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)
```

The file package is downloaded, and the required files **ratings.dat** and **movies.dat** are decompressed and loaded into memory. The columns in datasets **ratings** and **movies** are changed to their corresponding names. We also need to modify the data in the **movies** to the correct type.

```
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))
```

We combine **ratings** and **movies** together, and randomly select **90%** rows to create the **edx** set, which will be used to develop our models. Later, we split the **edx** dataset into separate training and testing sets to train and test our models.

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

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

In the meantime, **10%** of the remaining rows will be used to create the **validation** set. This set is the final hold-out test set and we assume that the ratings of these movies are unknown and will be predicted by our optimal algorithm. The **RMSE** will be calculated and used to evaluate how close our prediction is to the actual value in the **validation** set.

We have to make sure that the **userId** and **movieId** in the **validation** set are also in the **edx** set.

```
temp <- movielens[test_index,]
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

For those rows deleted from the **validation** set, we must add them back to the **edx** set. Finally, we remove all temporary variables.

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

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

2.2 Data Cleaning

Initially, we need to preprocess the data before performing any exploration. Using the **summary** method, we can easily examine the distribution of numeric columns, as well as, the number of 0, NA, and other invalid values.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

We can confirm from the above results that there are **6** columns in the **edx** set, and **userId**, **movieId**, **rating**, and **timestamp** are numeric columns. And, no invalid value is found. **title** and **genres** are character columns. We can also find that the **edx** set contains **9,000,055** rows of data.

```
colSums(is.na(edx))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

The original dataset is very well maintained, and the data cleaning process is complete. There are no invalid values in **edx** set.

2.3 Data Exploration

2.3.1 Row

Let's check the content of a specific row. Use the **head** method to review the first five rows of **edx**.

```
edx %>% head(5)
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

Thus, each row represents a specific rating of the movie from the user. **userId** displays the ID of the user who rated the movie. **movieId** is the ID of movie being rated. **rating** is the class label, which is the actual rating number, ranging from **0.5** to **5**. **timestamp** is the date and time of the rating. **title** contains the movie name and its release year. **genres** shows the type of movie.

2.3.2 userId

There are a total of **69878** users, ranging from **1** to **71567**, and the top five users with the highest ratings are shown below.

```
edx %>%
  group_by(userId) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  head(5)
```

userId	count
59269	6616
67385	6360
14463	4648
68259	4036
27468	4023

Hence, the user ID **59269** has the most ratings in the **edx** set, resulting **6616** distinct ratings.

2.3.3 movieId

Movie IDs range from **1** to **65133**. The top five movies with the most number of five-star rating are shown below.

```
edx %>%
  filter(rating == 5) %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  head(5)
```

movieId	count
1	6294
2	844
3	631
4	107
5	458

We can determine that the movie ID **1** gets **6294** five-star, which is the most five-star ratings and much more than the second-place movie.

2.3.4 rating

Since **rating** is the class label, we need to understand it in depth. Use **table** method to obtain the rating distribution.

```
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
```

The ratings are distributed from **0.5** to **5**, and the ratings do not get zero. There are **2,588,430** four-star are given as the most number of ratings in the **edx** dataset. The top five in order from most to least are 4, 3, 5, 3.5, 2. In addition, half-star ratings are not as common as full-star ratings.

2.3.5 timestamp

The date and time of a particular given rating are stored in the column **timestamp**, which represents the number of seconds since midnight UTC on January 1, 1970. The following table shows the earliest five ratings given in the **edx** dataset.

```
edx %>%
  mutate(datetime = as.POSIXct(timestamp, origin="1970-01-01")) %>%
  select(userId, title, rating, timestamp, datetime) %>%
  arrange(datetime) %>%
  head(5)
```

userId	title	rating	timestamp	datetime
36955	Seven (a.k.a. Se7en) (1995)	5	789652009	1995-01-09 03:46:49
36955	Fish Called Wanda, A (1988)	3	789652009	1995-01-09 03:46:49
35139	Toy Story (1995)	4	822873600	1996-01-28 16:00:00
35139	Get Shorty (1995)	5	822873600	1996-01-28 16:00:00
35139	Dangerous Minds (1995)	5	822873600	1996-01-28 16:00:00

Here, the first rating was given at 03:46:49 UTC on January 9, 1995, to the movie **Se7en**.

2.3.6 title

There are a total of **10677** different movies, and the top five movies with the most number of ratings are shown below.

```
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  head(5)
```

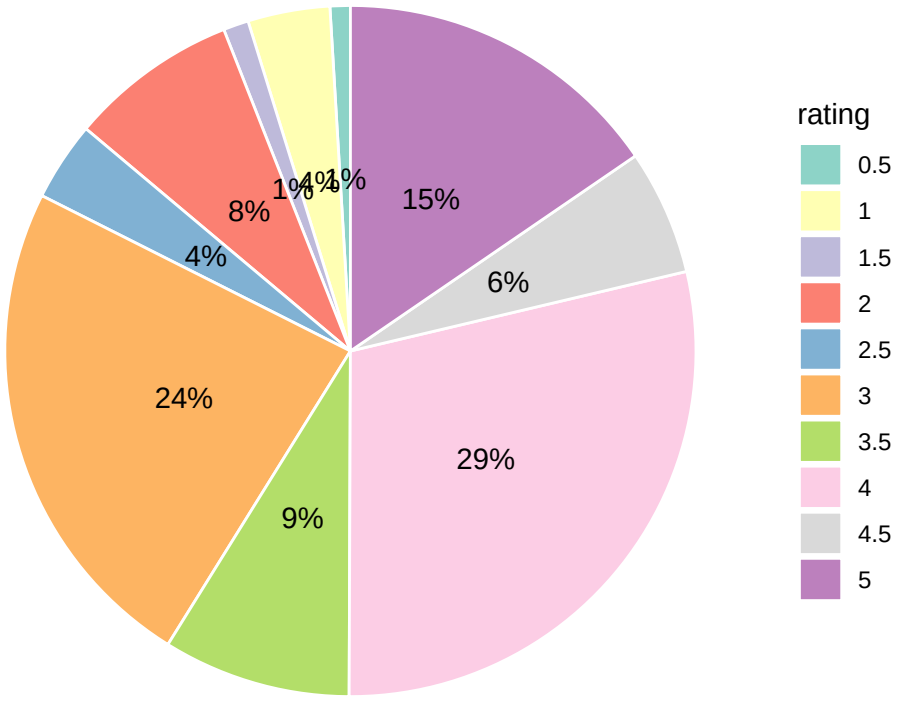
title	count
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015

According to the result illustrated here, the movie **Pulp Fiction** released in 1994 received **31362** ratings, which is the greatest number of ratings in the **edx** set.

2.4 Data Visualization

2.4.1 rating

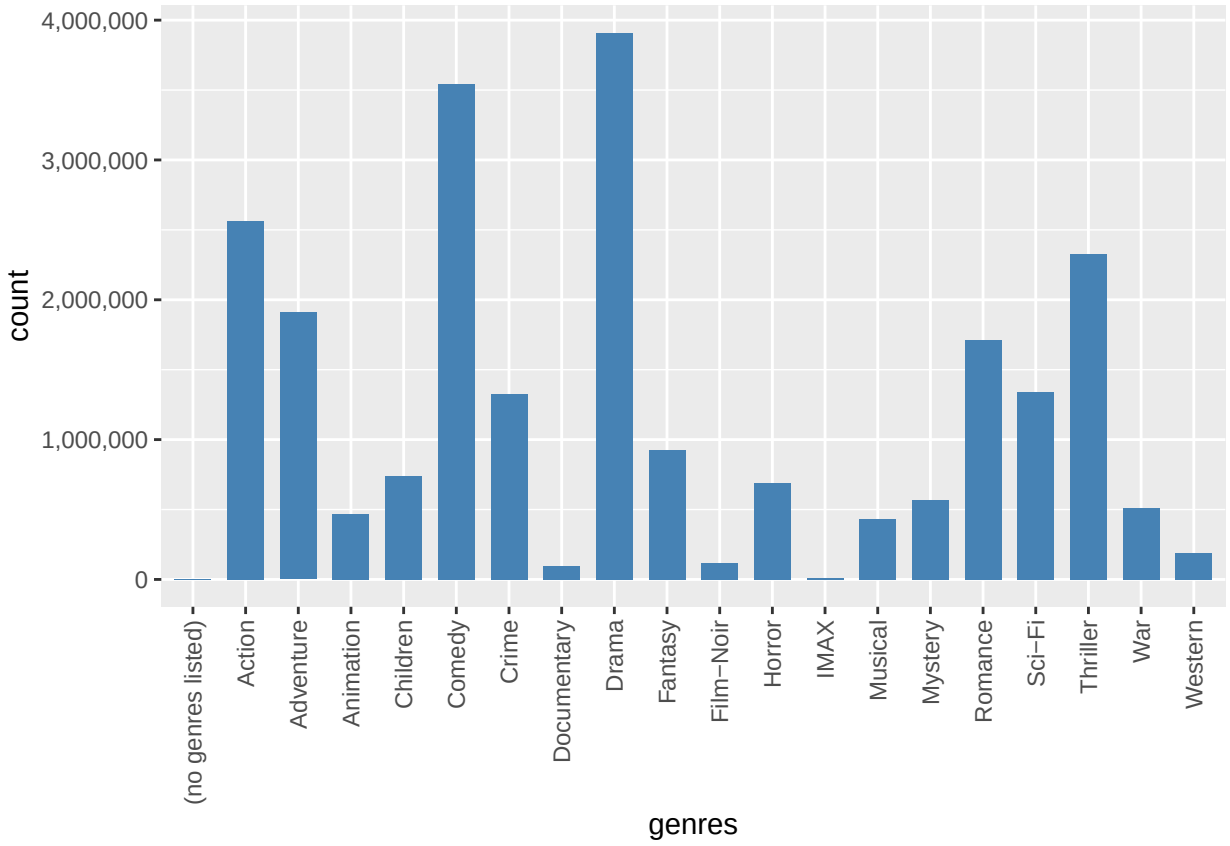
```
edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  mutate(prop = count / sum(count) * 100) %>%
  mutate(rating = as.character(rating)) %>%
  ggplot(aes(x = "", y = count, fill = rating)) +
  geom_bar(width = 1, stat = "identity", color = "white") +
  coord_polar("y", start = 0) +
  geom_text(aes(label = paste0(round(prop), "%")), position = position_stack(vjust = 0.5)) +
  scale_fill_brewer(palette = "Set3")+
  theme_void()
```



The ratings range from **0.5** to **5**. From the pie chart created above, about 29% of the ratings are fou,r which is the most number of ratings. About 24% of the ratings are three, which is the second highest.

2.4.2 genres

```
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = genres, y = count)) +
  geom_bar(stat="identity", width=0.7, fill="steelblue") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5)) +
  scale_y_continuous(labels = scales::comma)
```



There are 19 movie genres in the **edx** set. The bar chart above shows the distribution of ratings for different movie genres. Obviously, the ratings of drama movies are about 4 million, comedy movies are about 3.5 million, and action movies are about 2.5 million.

2.5 Insights gained

In **edx**, more than **6K** ratings were made by the user who rates the most and about **29%** of the ratings are four-star which is the most given ratings. For a specific movie, **Pulp Fiction** released in **1994** gets more than **30K** ratings which is the greatest number of ratings, and **Toy Story** released in **1995** gets more than **6K** five-stars ratings which is the greatest number of five-star ratings. There are **19** movie genres and the most genre is drama movies which received about **4M** ratings.

2.6 Modeling Approach

In order to evaluate the performance of the project, the **edx** set is divided into a training set and a testing set. **3** different models will be implemented for analysis and comparison. Each model will be constructed base on the training set and predict the movie ratings in the testing set.

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

test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
```

We then compare the predicted values and the actual ratings and report the RMSE score. The RMSE function is defined as follows.

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

2.6.1 Model 1: Movie Effects Model

The first Model is the movie effects model, which predicts movie ratings based on the overall mean and considers movie bias. We start by calculating the overall mean **mu**, regardless of movies and users.

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

We create a new dataset **avg_movie** and add a term **b_i**, which represents the bias against the overall mean **mu** for movie **i**. **train_set** is then grouped by movies, and **b_i** is the average of the difference between the ratings of movie **i** and the overall mean **mu**.

```
avg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

We compute the predicted values by adding the overall mean **mu** and the movie bias **b_i**.

```
predicted <- mu + test_set %>%
  left_join(avg_movie, by = 'movieId') %>%
  pull(b_i)
```

In the end, we compare the predicted values with the actual rating in **test_set**, and report the RMSE score.

```
model_1_rmse <- RMSE(test_set$rating, predicted)
model_1_rmse
```

```
## [1] 0.9429615
```

2.6.2 Model 2: Movie + User Effects Model

The second Model is the movie plus user effects model, which predicts movie rating based on the overall mean, not only considering movie bias, but also user-specific bias.

We start by creating a new dataset **avg_user**, and then add a term **b_u**, which represents the bias against the overall mean **mu** for user **u**. **train_set** is grouped by users, and **b_u** is the average of the difference between the ratings from users **u** and the overall mean **mu** and movie bias **b_i**.

```
avg_user <- train_set %>%
  left_join(avg_movie, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

We compute the predicted values by adding the overall mean **mu**, movie bias **b_i**, and user bias **b_u**.

```
predicted <- test_set %>%
  left_join(avg_movie, by = 'movieId') %>%
  left_join(avg_user,  by = 'userId')  %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

In the end, we compare the predicted values and actual rating in **test_set** and report the RMSE score.

```
model_2_rmse <- RMSE(test_set$rating, predicted)
model_2_rmse
```

```
## [1] 0.8646843
```

2.6.3 Model 3: Movie + User-specific Genre Effects Model

The third Model is the movie plus user-specific genre effects model, which predicts the movie rating based on the overall mean and considers the movie, user, and user-specific genre bias.

We start by creating a new dataset **avg_genre**, and then add a term **b_g**, which represents the bias against the overall mean **mu** for user-specific genre **g**. We first add the previous biases **b_i** and **b_u** to the **train_set**, then separate each row and make sure that the **genres** column only contains a single genre.

After that, group by user and genre, and **b_g** is the average of the difference between ratings from user **u** and genre **g**, the overall mean **mu**, and the movie bias **b_i**, and the user bias **b_u**.

```
avg_genre <- train_set %>%
  left_join(avg_movie, by = 'movieId') %>%
  left_join(avg_user,  by = 'userId')  %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(userId, genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
```

We compute the predictions by adding the overall mean **mu**, movie bias **b_i**, user bias **b_u**, and user-specific genre bias **b_g**.

```
predicted <- test_set %>%
  separate_rows(genres, sep = "\\|") %>%
  left_join(avg_genre, by = c('userId','genres')) %>%
  group_by(userId, movieId, rating) %>%
  summarize(b_g = mean(b_g, na.rm = TRUE)) %>%
  left_join(avg_movie, by = 'movieId') %>%
  left_join(avg_user,  by = 'userId')  %>%
  mutate(pred = sum(mu, b_i, b_u, b_g, na.rm = TRUE)) %>%
  pull(pred)
```

At the end, we compare the predicted values and actual ratings in **test_set** and report the RMSE score.

```
model_3_rmse <- RMSE(test_set$rating, predicted)
model_3_rmse
```

```
## [1] 0.850371
```

3 Results

3.1 Optimal Model

Obviously, model 3 produces the best RMSE which is **0.85**. The score is passing **0.86490** and we reach the goal of this project.

Model 3 is the best since it considers the most bias effects including the movie bias, the user bias, and the user-specific genre bias. The last added bias improves the model performance magnificently since it takes considering the user's specific emotion for different kinds of movies.

Some users may prefer romance movies and they can rate each romance movie a litttle bit higher than their criterions. Some user may dislike action movies and they can rate each action movie a bit lower than their standard ratings for all kinds of movies.

Therefore, the user-specific genre bias should be included and make the model more accuracy. We will choose the third model for this project and predict the final hold-out test set **validation**.

3.2 Final Validation

To evaluate the performance of the project, We implement the movie plus user-specific genre effects model on both **edx** and **validation** sets. The **edx** dataset becomes the training set and the **validation** set becomes the testing sets.

The model will be build base on the **edx** set and predict the movie ratings in the **validation** set. We start with updating overall mean **mu** and datasets **avg_movie**, **avg_user**, and **avg_genre**.

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

avg_movie <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

avg_user <- edx %>%
  left_join(avg_movie, by = 'movieId') %>%
```

```
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

avg_genre <- edx %>%
  left_join(avg_movie, by = 'movieId') %>%
  left_join(avg_user,  by = 'userId')  %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(userId, genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
```

We compute the predictions with the updated training data for the movie ratings in the **validation** set.

```
predicted <- validation %>%
  separate_rows(genres, sep = "\\|") %>%
  left_join(avg_genre, by = c('userId','genres')) %>%
  group_by(userId, movieId, rating) %>%
  summarize(b_g = mean(b_g, na.rm = TRUE)) %>%
  left_join(avg_movie, by = 'movieId') %>%
  left_join(avg_user,  by = 'userId')  %>%
  mutate(pred = sum(mu, b_i, b_u, b_g, na.rm = TRUE)) %>%
  pull(pred)
```

Finally, we compare the predicted values and actual ratings in **validation** and report the final RMSE score.

```
final_rmse <- RMSE(validation$rating, predicted)
final_rmse
```

```
## [1] 0.8499802
```

Interestingly, the RMSE score reached **0.84998**, which is even lower than the previous best score. It illustrates the fact that more training data makes the model more accurate.

4 Conclusion

4.1 Project Summary

First, we acquired and preprocessed data for analysis; explored and visualized them to better understand the data, and gained insights.

Secondly, we tried **3** different models and gradually improved the RMSE score. The last RMSE score was **0.85**, and it passed **0.86490**, which is the goal of the project. We have determined that **Movie + User-specific Genre Effects Model** is the optimal.

Finally, we execute the optimal model on the **edx** and **validation** sets as the training and testing sets, and obtain **0.84998** as the final RMSE score.

4.2 Limitations

In our project, if the user has never rated a particular genre of movie, the model has no idea about the user's preference for such kind of movies, so it will return **0** for the user-specific genre bias. Thus, the model could not make accurate or better predictions for the type of movies rated by the user.

In terms of efficiency, the calculation time based on **genres** is already very slow. If the dataset grows, the running time will be worse.

4.3 Future Work

In the section of recommendation system in HarvardX Machine Learning course, it introduced that regularization can be implemented to improve our results. Regularization constrains the total variability of the effect size by penalizing large estimates from a small sample size.

In addition, we can use matrix factorization to account for changes related to movie groups and users with similar rating patterns. These patterns can be learned through residuals, and our data will be transformed into a matrix in which each user gets a row and each movie gets a column.

These techniques could definitely improve the performance of our project.

5 Reference

HarvardX PH125.8x Data Science: Machine Learning, Section 6.2 Recommendation Systems.