MovieLens Project Report

Han Lu

6/6/2020

I. Introduction

The recommendation system is an integral part of today's e-commerce and online services industry. It helps the user find the best products and services that suit their needs based on their previous activity. Companies such as Netflix, YouTube, and Amazon rely on these systems to provide their customers personalized experience and increase their satisfaction. For example, in 2006, Netflix offered a one million dollar prize on the improvement of at least 10% of its recommendation system.

The goal of this project is to build a movie recommendation system using the MovieLens dataset. The remainder of the project is organized as follows. After a brief review of the dataset and model evaluation methods in part I, part II presents the analysis and results. This includes data preparation, data exploration, and modeling. The modeling segment consists of linear modeling, regularization, and final model evaluation. Finally, part III concludes the project and provides a summary of the results.

1.1 Dataset

Collected by GroupLens, a research lab at the University of Minnesota, the full MovieLens dataset consists of 27 million ratings and 1.1 million tag applications applied to 58,000 movies by 280,000 users. In this project, we use the MovieLens 10M Dataset, a stable benchmark subset of the full dataset that provides 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

1.2 Evaluation Method

A general approach to evaluate machine learning algorithms is to define a loss function that measures the difference between the predicted value and observed outcome. Since lower loss produces higher accuracy, our goal is to minimize the loss, so it is as close to zero as possible. Here we use three commonly used loss functions: mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). All three metrics are computed and reported in this exercise for comparison, although we only focus on minimizing the RMSE when choosing the best algorithm since it is in the same units as the outcomes. The formulas of these metrics are

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)}$$

where $\hat{y}_i$ is the predicted value, y is the observed outcome, and N is the number of observations. RMSE is the square root of MSE. When the outcomes are binary, both metrics are equivalent since $(\hat{y} - y)^2$ is zero if the prediction was correct and one otherwise. Unlike RMSE and MSE, which use squared loss, MAE uses absolute values instead. We can define the loss functions with the following code.

```r
# Define Mean Absolute Error (MAE)
MAE <- function(true_ratings, predicted_ratings){
  mean(abs(true_ratings - predicted_ratings))
}

# Define Mean Squared Error (MSE)
MSE <- function(true_ratings, predicted_ratings){
  mean((true_ratings - predicted_ratings)^2)
}

# Define Root Mean Squared Error (RMSE)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

II. Analysis and Results

This section presents the data preparation, data exploration, and modeling procedure.

2.1 Data Preparation

We download the MovieLens 10M Dataset and split it into a study set (`edx`) and a validation set (`validation`) with a 90/10 split percentage ratio. The `edx` set is then split again into a training set and a test set with the same split ratio for the modeling process. The `validation` set is only used for evaluating the final algorithm. For the final test of the algorithm, we predict movie ratings in the `validation` set as if they are unknown to us, and use RMSE to determine how close the final predictions are to the true values.

```r
# Create edx set, validation 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")
```

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following objects are masked _by_ '.GlobalEnv':
##
##     MAE, RMSE

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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(levels(movieId))[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
```

```r
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

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

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

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

```
# Train-test split of the edx set

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

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

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set

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

# Add rows removed from test set back into train set

removed <- anti_join(temp, test)
```

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

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

rm(test_index, temp, removed)
```

2.2 Data Exploration

The `edx` set has 9,000,055 entries with six variables or columns. Similarly, the `validation` set has 999,999 entries and six columns. The dataset is in a tidy format, which gives us one observation of each row and columns as variables. Below is a summary of the `edx` set.

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

```r
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
##
##
##
```

The `userId` variable identifies the user information, while the `movieId` and `title` columns identify the movie information. Each movie is tagged with one or more genres, as shown in the `genres` column. Movie ratings are stored in the `rating` column. The variable `timestamp` contains the rating dates measured in seconds with January 1st, 1970, as the epoch.

Now we take a closer look at each variable. The `timestamp` variable indicates that the data was collected over almost 14 years.

```r
# Summarize the dates of the dataset
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
tibble('Start Date' = date(as_datetime(min(edx$timestamp), origin="1970-01-01")),
       'End Date' = date(as_datetime(max(edx$timestamp), origin="1970-01-01"))) %>%
  mutate(Span = duration(max(edx$timestamp)-min(edx$timestamp)))
```

```
## # A tibble: 1 x 3
##   `Start Date` `End Date` Span
##   <date>       <date>     <Duration>
## 1 1995-01-09   2009-01-05 441479727s (~13.99 years)
```

The `userId` and `movieId` columns show that there are 69,878 unique users rating 10,677 different movies, indicating that the number of ratings varies. Some of the movies are more popular and rated more than others. We can see this pattern clearly from the movies' distribution by the number of ratings.

```
# Number of users in the edx set
length(unique(edx$userId))
```

```
## [1] 69878
```

```
# Number of movies in the edx set
length(unique(edx$movieId))
```

```
## [1] 10677
```

```
if(!require(scales))
  install.packages("scales", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: scales
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

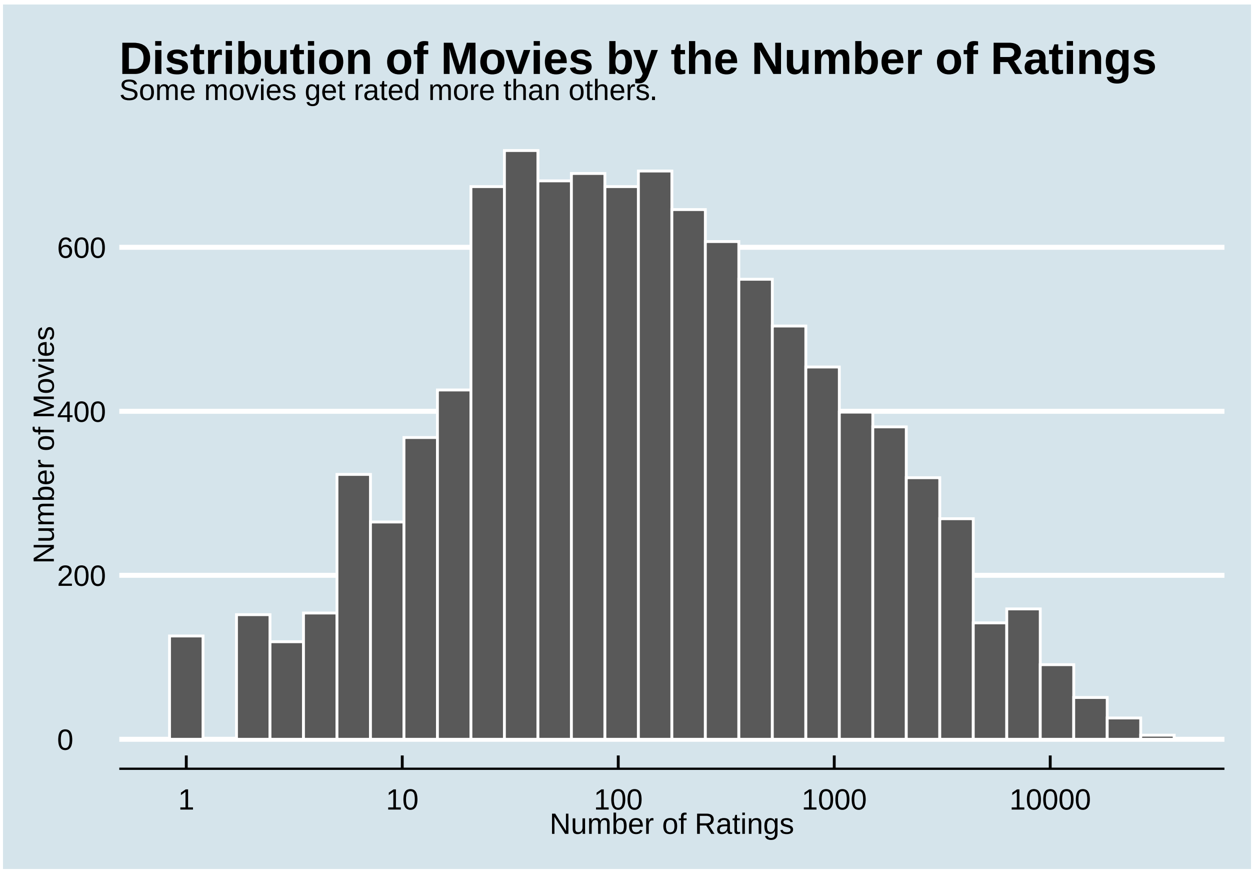
```
if(!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ggthemes
```

```
# Distribution of movies by the number of ratings
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
    scale_x_log10() +
    ggtitle("Distribution of Movies by the Number of Ratings",
            subtitle = "Some movies get rated more than others.") +
    xlab("Number of Ratings") +
    ylab("Number of Movies") +
    theme_economist()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

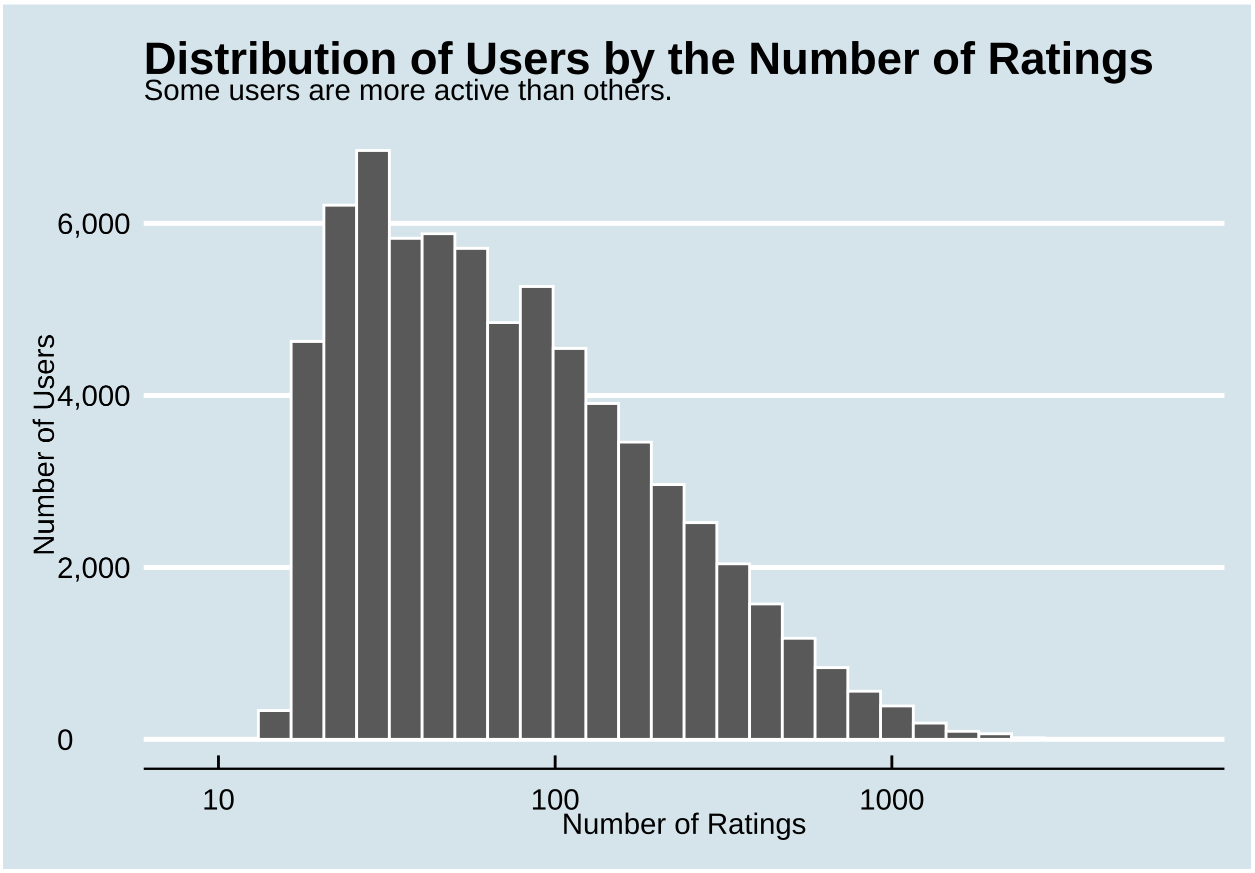


Some users are also more active than others, as suggested in the users' distribution below.

```r
# Distribution of users by the number of ratings
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
    scale_x_log10() +
    ggtitle("Distribution of Users by the Number of Ratings",
            subtitle="Some users are more active than others.") +
    xlab("Number of Ratings") +
    ylab("Number of Users") +
    scale_y_continuous(labels = comma) +
    theme_economist()
```

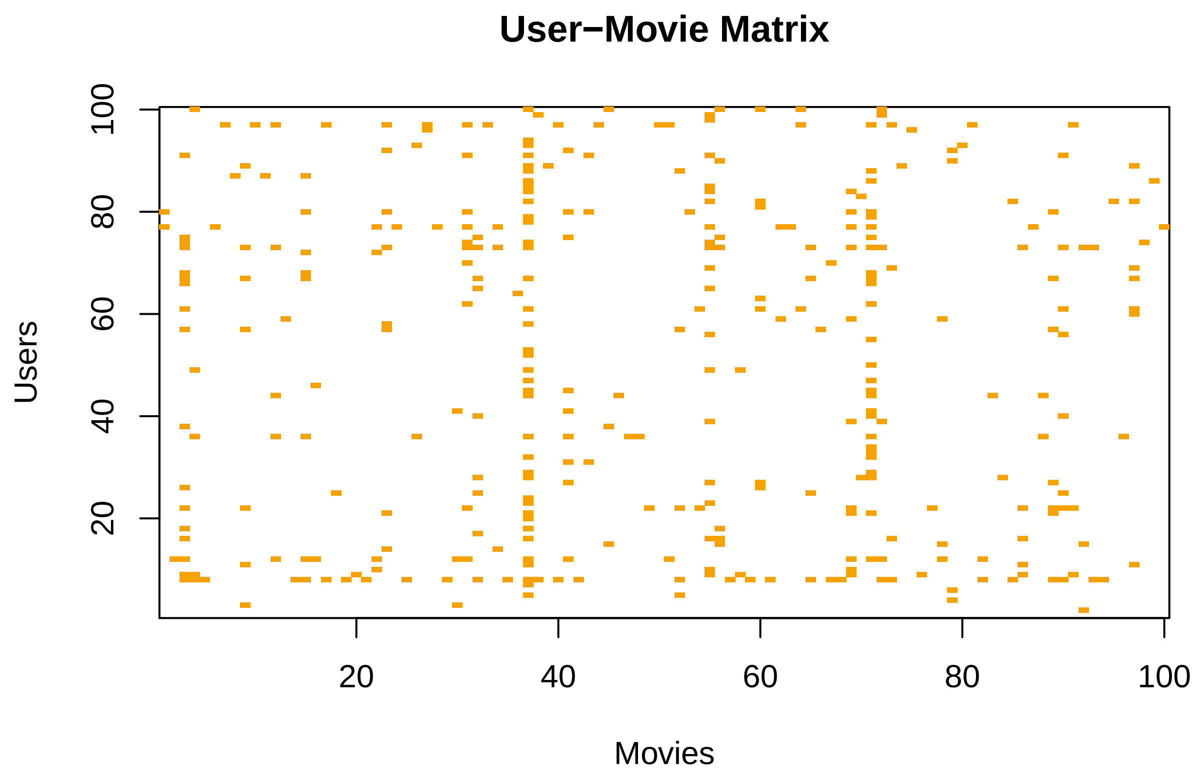
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The following user-movie sparse matrix presents a random sample of 100 movies and 100 users with yellow indicating that there is a rating that exists for that user-movie combination. The matrix is sparse with the majority of empty cells. This makes it easy for us to identify that some movies get more ratings, and some users are more active than others.

```r
# User-movie sparse matrix
users <- sample(unique(edx$userId), 100)
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
title("User-Movie Matrix")
```



The data also shows, as expected, that the well-known blockbusters tend to have the highest number of ratings, with *Pulp Fiction* (1994) ranked on top of the list with 31,362 ratings in total.

```
# Most rated movies
edx %>% group_by(title) %>%
  summarize(n= n()) %>%
  arrange(desc(n))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 10,676 x 2
##    title                                                            n
##    <chr>                                                        <int>
##  1 Pulp Fiction (1994)                                          31362
##  2 Forrest Gump (1994)                                          31079
##  3 Silence of the Lambs, The (1991)                             30382
##  4 Jurassic Park (1993)                                         29360
##  5 Shawshank Redemption, The (1994)                             28015
##  6 Braveheart (1995)                                            26212
##  7 Fugitive, The (1993)                                         25998
##  8 Terminator 2: Judgment Day (1991)                            25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10 Apollo 13 (1995)                                             24284
## # ... with 10,666 more rows
```

Meanwhile, there are more than 100 movies with only a single rating.

```r
# Number of movies rated once
edx %>% group_by(title) %>%
  summarize(n = n()) %>%
  filter(n==1) %>%
  count() %>%
  pull()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## Using `n` as weighting variable
## i Quiet this message with `wt = n` or count rows with `wt = 1`
```

```
## [1] 126
```

When rating movies, users can choose to give one of the ten rating points ranging from 0.5 to 5. The distribution of ratings indicates that the five most given ratings in order from most to least are 4, 3, 5, 3.5, and 2. In general, half-star ratings are less common than whole star ratings. For instance, there are fewer ratings of 3.5 than there are ratings of 3 or 4. This indicates that most users tend to round decimal scores to integers when giving ratings.

```r
# Count the number of each rating
edx %>% group_by(rating) %>% summarize(n=n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
##  3    1.5  106426
##  4    2    711422
##  5    2.5  333010
##  6    3   2121240
##  7    3.5  791624
##  8    4   2588430
##  9    4.5  526736
## 10    5   1390114
```

Finally, there are 797 different combinations of genres.

```r
length(unique(edx$genres))
```

```
## [1] 797
```

2.3 Modeling

This segment presents linear modeling, regularization, and final model evaluation.

A. Linear Model The simplest model is to use the average rating across all users and movies for prediction. This method assumes, as shown in the formula below that the rating variation is completely from the randomly distributed error term.

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

The $\hat{Y_{u,i}}$ is the predicted rating of user u and movie i, μ is the mean rating from the observed data, and ϵ is the error term. The code and results of this simple model are shown as follows.

```
# Mean of the observed values in the training set
mu <- mean(train$rating)

# Report results
result <- bind_rows(tibble(Method = "Mean",
                           RMSE = RMSE(test$rating, mu),
                           MSE  = MSE(test$rating, mu),
                           MAE  = MAE(test$rating, mu)))
result
```

```
## # A tibble: 1 x 4
##   Method  RMSE   MSE   MAE
##   <chr>  <dbl> <dbl> <dbl>
## 1 Mean    1.06  1.12 0.855
```

Since we see in the data exploration section that movies differ in popularity, therefore it is reasonable to add a movie effect term, b_i, that count this movie to movie variability to our mean model. The movie effect term can be defined as the difference between the observed and the mean rating. The model can be written as

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

where

$$\hat{b_i} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu).$$

As we can see in the results, this change brings improvement to the RMSE.

```
# Create movie effect term (b_i)
b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
head(b_i)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.415
## 2       2 -0.306
## 3       3 -0.361
## 4       4 -0.637
## 5       5 -0.442
## 6       6  0.302
```

```
# Predict ratings with mu and b_i
y_hat <- test %>%
  left_join(b_i, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

# Report results
result <- bind_rows(result,
                    tibble(Method = "Movie Effect",
                           RMSE = RMSE(test$rating, y_hat),
                           MSE  = MSE(test$rating, y_hat),
                           MAE  = MAE(test$rating, y_hat)))
result
```

```
## # A tibble: 2 x 4
##   Method        RMSE   MSE   MAE
##   <chr>        <dbl> <dbl> <dbl>
## 1 Mean         1.06  1.12  0.855
## 2 Movie Effect 0.943 0.889 0.737
```

Similar to the movie effect, the user to user variability can also be modeled by adding a user effect term $\hat{b_u}$, which captures the users' rating patterns. The model can be written as

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

where

$$\hat{b_u} = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - b_i - \mu).$$

This model further improves the RMSE..

```
# Create user effect term (b_u)
b_u <- train %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# Predict ratings with mu, b_i, and b_u
y_hat <- test %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Update results
result <- bind_rows(result,
                    tibble(Method = "Movie and User Effect",
                           RMSE = RMSE(test$rating, y_hat),
                           MSE  = MSE(test$rating, y_hat),
                           MAE  = MAE(test$rating, y_hat)))
result
```

```
## # A tibble: 3 x 4
##   Method                  RMSE   MSE   MAE
##   <chr>                  <dbl> <dbl> <dbl>
## 1 Mean                   1.06  1.12  0.855
## 2 Movie Effect           0.943 0.889 0.737
## 3 Movie and User Effect 0.865 0.748 0.668
```

B. Regularization As we see during the data exploration process, many movies and users have a very low number of ratings. This leads to a large estimated error due to small sample size. We employ regularization, penalizing small sample sizes to reduce the effect of error and controlling the total variability of the movie and user effects in our estimation. Instead of minimizing the least squares equation, we minimize

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2)$$

where the first term is our previous least squares equation, and the last term is the penalty for large values of b_i and b_u. The equations of b_i and b_u corresponding to this minimization are

$$\hat{b_i}=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(y_{u,i}-\hat{\mu})$$

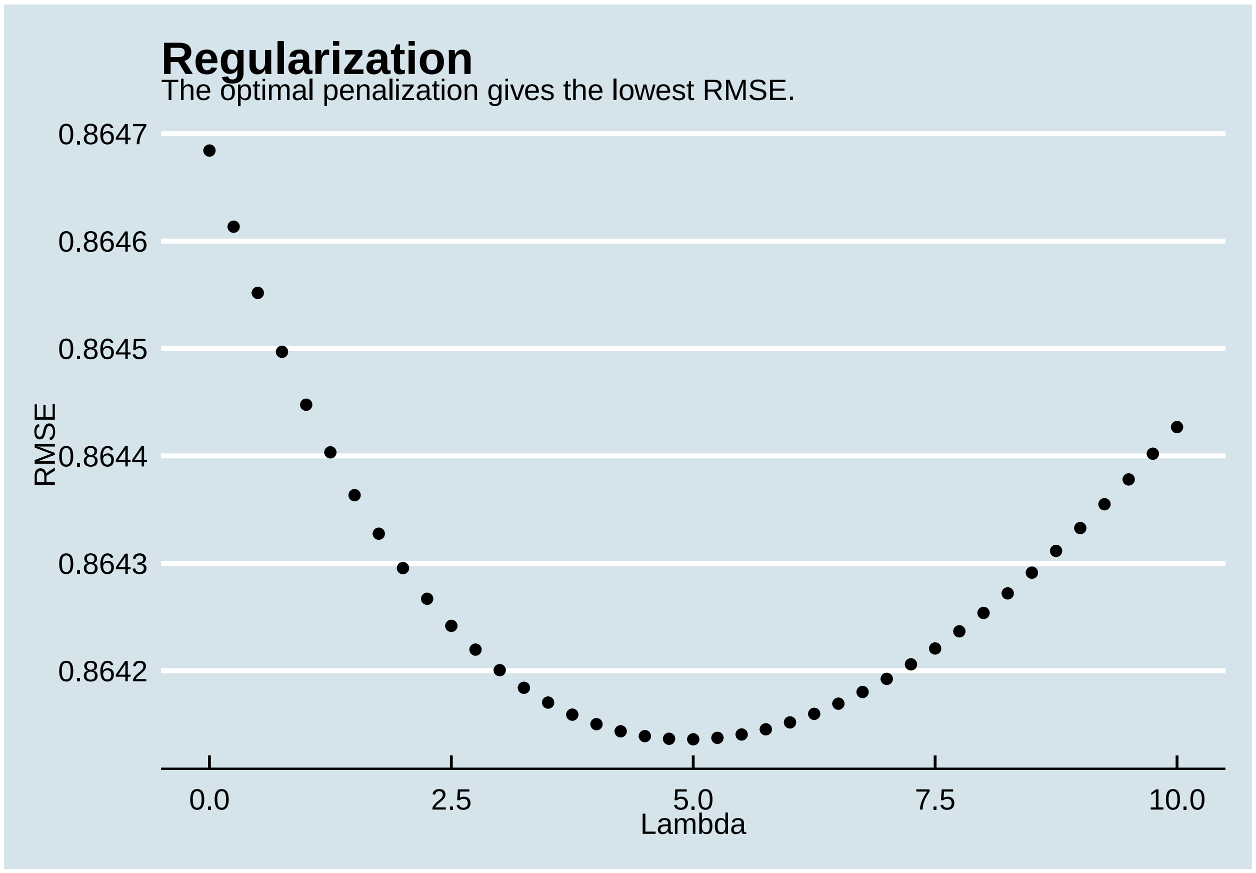
$$\hat{b_u}=\frac{1}{\lambda+n_u}\sum_{i=1}^{n_u}(y_{u,i}-\hat{\mu}-\hat{b_i}).$$

When n_i is the number of ratings made for movie i, n_u is the number of ratings made by user u. When the sample size is very large, we have $n_i+\lambda\approx n_i$, and the penalty λ is effectively ignored. However, when the sample size is small, $\hat{b_i}$ and $\hat{b_u}$ are shrunken toward 0 by the scale of λ. Since λ is a tuning parameter, we use cross-validation to choose the one that minimizes RMSE.

```r
# Define lambdas
lambdas <- seq(from=0, to=10, by=0.25)

# Compute RMSE for each lambda
rmses <- sapply(lambdas, function(l){

  # Mean rating
  mu <- mean(train$rating)

  # Movie effect (b_i)
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  # User effect (b_u)
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  # Predictions from y_hat = mu + b_i + b_u
  predicted_ratings <- test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  # Report RMSE
  return(RMSE(predicted_ratings, test$rating))
})
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# Plot the lambda vs RMSE
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
  geom_point() +
  ggtitle("Regularization",
          subtitle = "The optimal penalization gives the lowest RMSE.") +
  theme_economist()
```



```r
# Define the optimal lambda
lambda <- lambdas[which.min(rmses)]
```

Now we are ready to estimate the regularized model with the optimal λ.

```r
# Regularized movie effect (b_i)
b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# Regularized user effect (b_u)
b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# Prediction
y_hat <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Report results
result <- bind_rows(result,
                    tibble(Method = "Regularized Movie and User Effect",
                           RMSE = RMSE(test$rating, y_hat),
                           MSE  = MSE(test$rating, y_hat),
                           MAE  = MAE(test$rating, y_hat)))
result
```

```
## # A tibble: 4 x 4
##   Method                             RMSE   MSE   MAE
##   <chr>                             <dbl> <dbl> <dbl>
## 1 Mean                              1.06  1.12  0.855
## 2 Movie Effect                      0.943 0.889 0.737
## 3 Movie and User Effect             0.865 0.748 0.668
## 4 Regularized Movie and User Effect 0.864 0.747 0.669
```

The RMSE in the regularized model is slightly lower than the previous estimations.

C. Model Evaluation We proceed to the final evaluation of this model since the regularized movie and user effect model produces the lowest RMSE. We train the model again, use the entire `edx` set, and use the `validation` set to check the performance.

```r
# Mean rating
mu <- mean(edx$rating)

# Regularized movie effect (b_i)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# Regularized user effect (b_u)
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# Prediction
y_hat <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Report results
```

```r
result <- bind_rows(result,
                    tibble(Method = "Final Validation",
                           RMSE = RMSE(validation$rating, y_hat),
                           MSE  = MSE(validation$rating, y_hat),
                           MAE  = MAE(validation$rating, y_hat)))
result
```

```
## # A tibble: 5 x 4
##   Method                             RMSE   MSE   MAE
##   <chr>                             <dbl> <dbl> <dbl>
## 1 Mean                              1.06  1.12  0.855
## 2 Movie Effect                      0.943 0.889 0.737
## 3 Movie and User Effect             0.865 0.748 0.668
## 4 Regularized Movie and User Effect 0.864 0.747 0.669
## 5 Final Validation                  0.865 0.748 0.669
```

The RMSE computed using the `validation` set is similar to what we have achieved during the model building process, confirming the performance of the model. The top 10 best and top 10 worst movies predicted by this algorithm are shown below.

Top 10 Best Movies:

```r
# Top 10 best movies
validation %>%
left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  arrange(-pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

```
## # A tibble: 10 x 1
## # Groups:   title [7]
##    title
##    <chr>
##  1 Usual Suspects, The (1995)
##  2 Shawshank Redemption, The (1994)
##  3 Shawshank Redemption, The (1994)
##  4 Shawshank Redemption, The (1994)
##  5 Eternal Sunshine of the Spotless Mind (2004)
##  6 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
##  7 Schindler's List (1993)
##  8 Donnie Darko (2001)
##  9 Star Wars: Episode VI - Return of the Jedi (1983)
## 10 Schindler's List (1993)
```

Top 10 Worst Movies:

```r
# Top 10 worst movies
validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  arrange(pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

```
## # A tibble: 10 x 1
## # Groups:   title [9]
##    title
##    <chr>
##  1 Battlefield Earth (2000)
##  2 Police Academy 4: Citizens on Patrol (1987)
##  3 Karate Kid Part III, The (1989)
##  4 PokÃ©mon Heroes (2003)
##  5 Turbo: A Power Rangers Movie (1997)
##  6 Kazaam (1996)
##  7 PokÃ©mon Heroes (2003)
##  8 Free Willy 3: The Rescue (1997)
##  9 Shanghai Surprise (1986)
## 10 Steel (1997)
```

III. Conclusion

In this project, we built a linear model that predicts movie ratings. After collecting, preparing, and exploring the dataset, we started with a simple mean model, which is just the average of the observed ratings. We then added the movie effect and user effect to the model to capture the variability caused by the movie's popularity and user's rating behavior. The regularization process reduced the estimation error generated by the movies and users with very few ratings. Our final model achieved the RMSE of 0.865. The table below is a summary of the results.

Model	RMSE	MSE	MAE
Mean	1.06	1.12	0.855
Movie Effect	0.943	0.889	0.737
Movie and User Effect	0.865	0.748	0.668
Regularized Movie and User Effect	0.864	0.747	0.669
Final Validation	0.865	0.748	0.669