Report on MovieLens

Shide Qiu

22/03/2020

1. Indroduction

This project is related to the **MovieLens Project** of the *HarvardX: PH125.9x Data Science: Capstone course.* In this project, We will be creating a movie recommendation system using the MovieLens dataset. The 10M version of the MovieLens dataset was provided and used in this project.

In this project, we will trian a machine learning algorithm using the inputs in one subset to (edx dataset) predict movie ratings in the validation set. To compare different models or to see how well we're doing compared to a baseline, we will use **root mean squared error (RMSE) as our loss function.** RMSE is one of the most popular measure of accuracy, to compare forecasting errors of different models for a particular dataset. If N is the number of user-movie combination, $y_{u,i}$ is the rating for movie i by user u, and $\hat{y}_{u,i}$ is our prediction, then RMSE is defined as follows:

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

The best model is the one with lowest RMSE and will be used to predict the movie ratings.

1.1. Used Dataset

The MovieLens dataset is provided and downloaded through following resources:

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

1.2. Used Libraries

The following libraries will be used in this report:

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

1.3. Data Loading

```
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")
```

2. Methodology & Analysis

2.1. Data Pre-processing

The MovieLens dataset will be splittedinto 2 subsets: "edx", a training subset to train the algorithm, and "validation", as subset to test toe movie ratings.

```
# Validation set will be 10% of MovieLens data
set.seed(1)
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, removed)

# add year as a column in the edx & validation datasets
edx <- edx %>% mutate(year = as.numeric(str_sub(title, -5, -2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title, -5, -2)))

# split genres in edx and validation datasets
edx_genres <- edx %>% separate_rows(genres, sep = '\\|')
valid_genres <- validation %>% separate_rows(genres, sep = "\\|")
```

2.2. Data Exploration and Visualization

Let's look at some of the general properties of the data to better understand the challenge. We check the first several rows of "edx" subset. It contains seven variables "userId", "movieId", "rating", "timestamp", "title", "genres", and "year". Each row represents a single rating of a user for a movie.

```
head(edx)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046               Boomerang (1992)
```

```
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres year
## 1                Comedy|Romance 1992
## 2         Action|Crime|Thriller 1995
## 3                        Comedy 1994
## 4  Action|Drama|Sci-Fi|Thriller 1995
## 5       Action|Adventure|Sci-Fi 1994
## 6 Action|Adventure|Drama|Sci-Fi 1994
```

A summary of the subset confirms that there are no missing values in it.

```r
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres               year
##  Length:9000061     Length:9000061     Min.   :1915
##  Class :character   Class :character   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :1994
##                                        Mean   :1990
##                                        3rd Qu.:1998
##                                        Max.   :2008
```

A total of unique movies, users and genres in the "edx" subset is shown as below.

```r
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId), n_genres = n_distinct(g
  knitr::kable()
```

n_users	n_movies	n_genres
69878	10677	797

2.2.1. Total Movie Ratings per genre

```r
edx_genres %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    genres     count
##    <chr>      <int>
##  1 Drama    3909401
##  2 Comedy   3541284
##  3 Action   2560649
##  4 Thriller 2325349
```

```
## 5 Adventure 1908692
## 6 Romance   1712232
## 7 Sci-Fi    1341750
## 8 Crime     1326917
## 9 Fantasy    925624
## 10 Children  737851
```

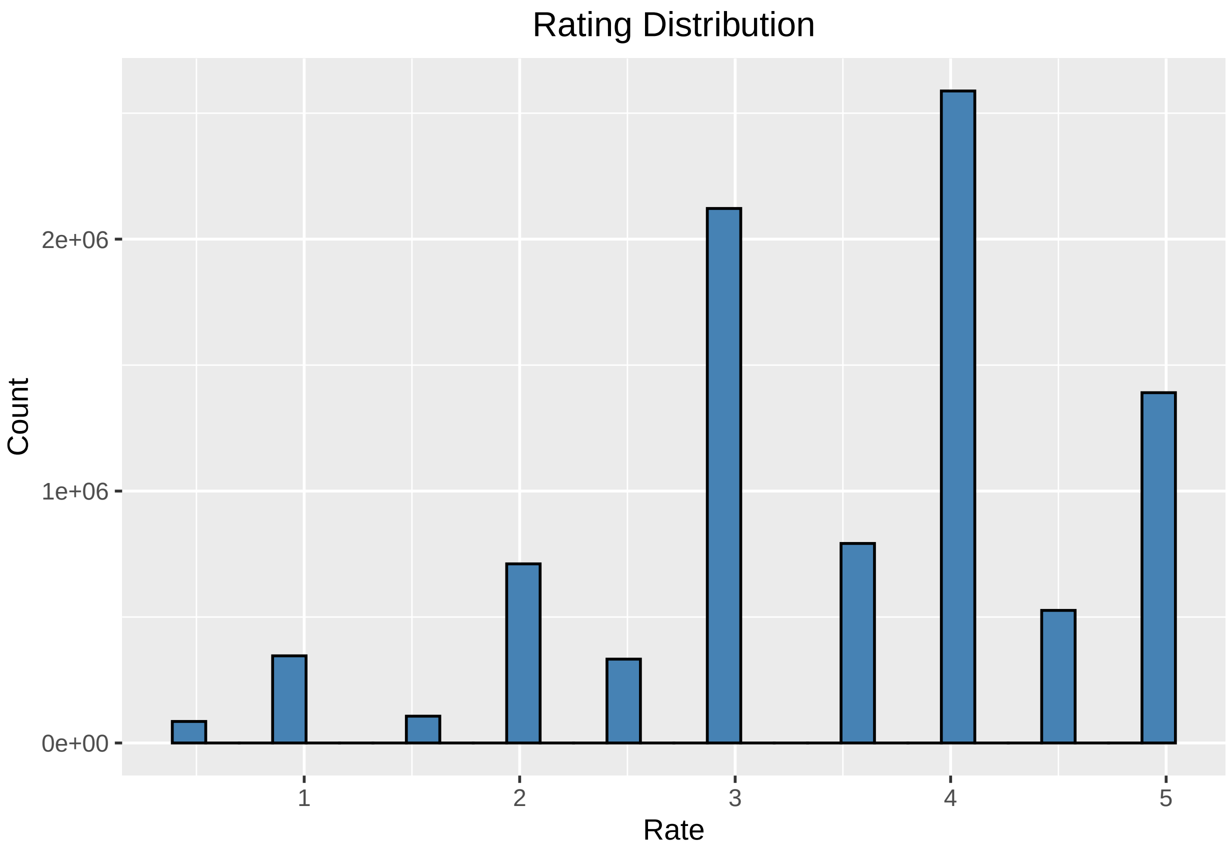
2.2.2. Top 10 Movies Ranked by Number of Ratings

```
edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                        count
##      <dbl> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          31336
##  2     356 Forrest Gump (1994)                                          31076
##  3     593 Silence of the Lambs, The (1991)                             30280
##  4     480 Jurassic Park (1993)                                         29291
##  5     318 Shawshank Redemption, The (1994)                             27988
##  6     110 Braveheart (1995)                                            26258
##  7     589 Terminator 2: Judgment Day (1991)                            26115
##  8     457 Fugitive, The (1993)                                         26050
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25809
## 10     592 Batman (1989)                                                24343
## # ... with 10,667 more rows
```

2.2.3. Rating Distribution

```
edx %>% group_by(rating) %>%
  ggplot(aes(rating)) +
  geom_histogram(bins = 30, fill = 'steelblue', col = 'black') +
  xlab('Rate') +
  ylab('Count') +
  ggtitle('Rating Distribution') +
  theme(plot.title = element_text(hjust = 0.5))
```

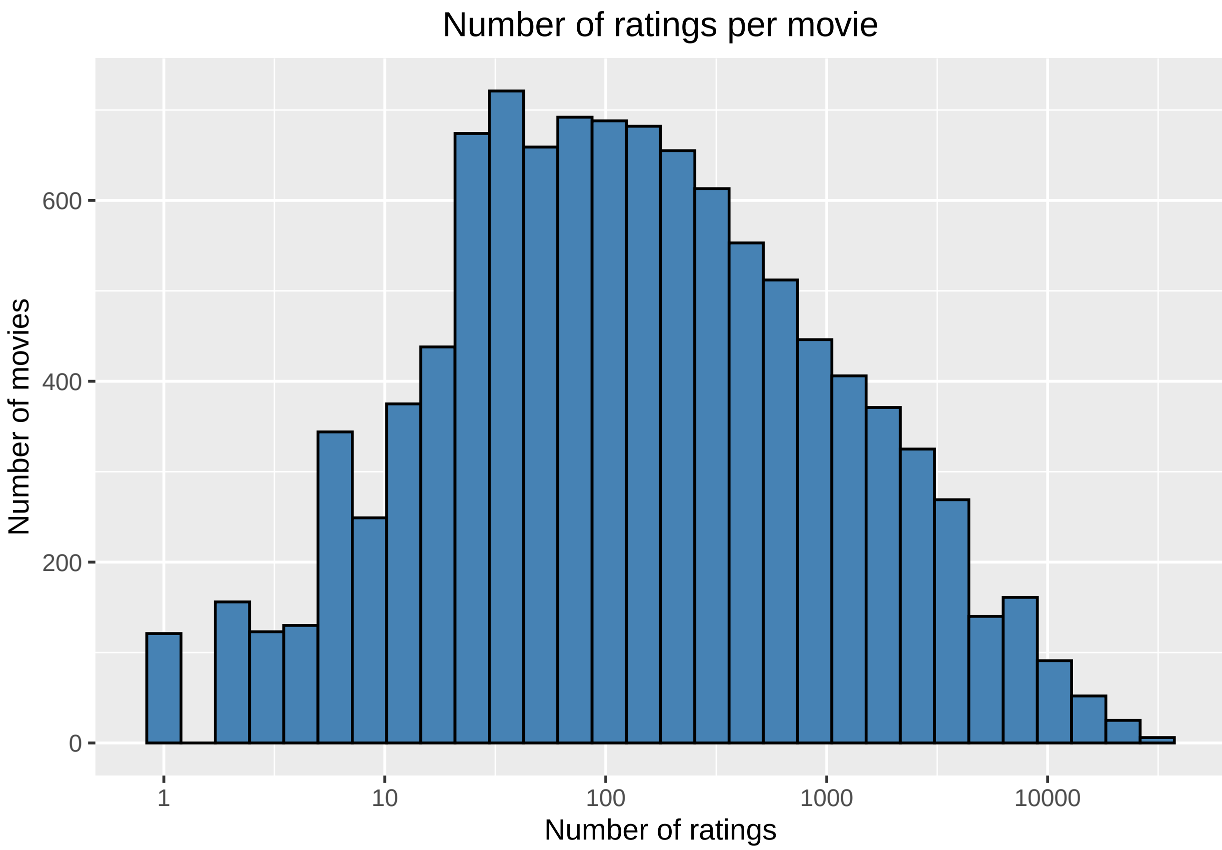


The rating distribution shows that users have a general tendency to rate movies between 3 and 4. However, this is a very general conclusion. We will further explore the effect of different features to make a good model.

2.2.4. Movie Bias

We notice that some movies get rated more than others. Here is the distribution.

```
edx %>% count(movieId) %>% ggplot(aes(n)) +
  geom_histogram(bins = 30, fill = 'steelblue', col = 'black') +
  scale_x_log10() +
  xlab('Number of ratings') +
  ylab('Number of movies') +
  ggtitle('Number of ratings per movie') +
  theme(plot.title = element_text(hjust = 0.5))
```

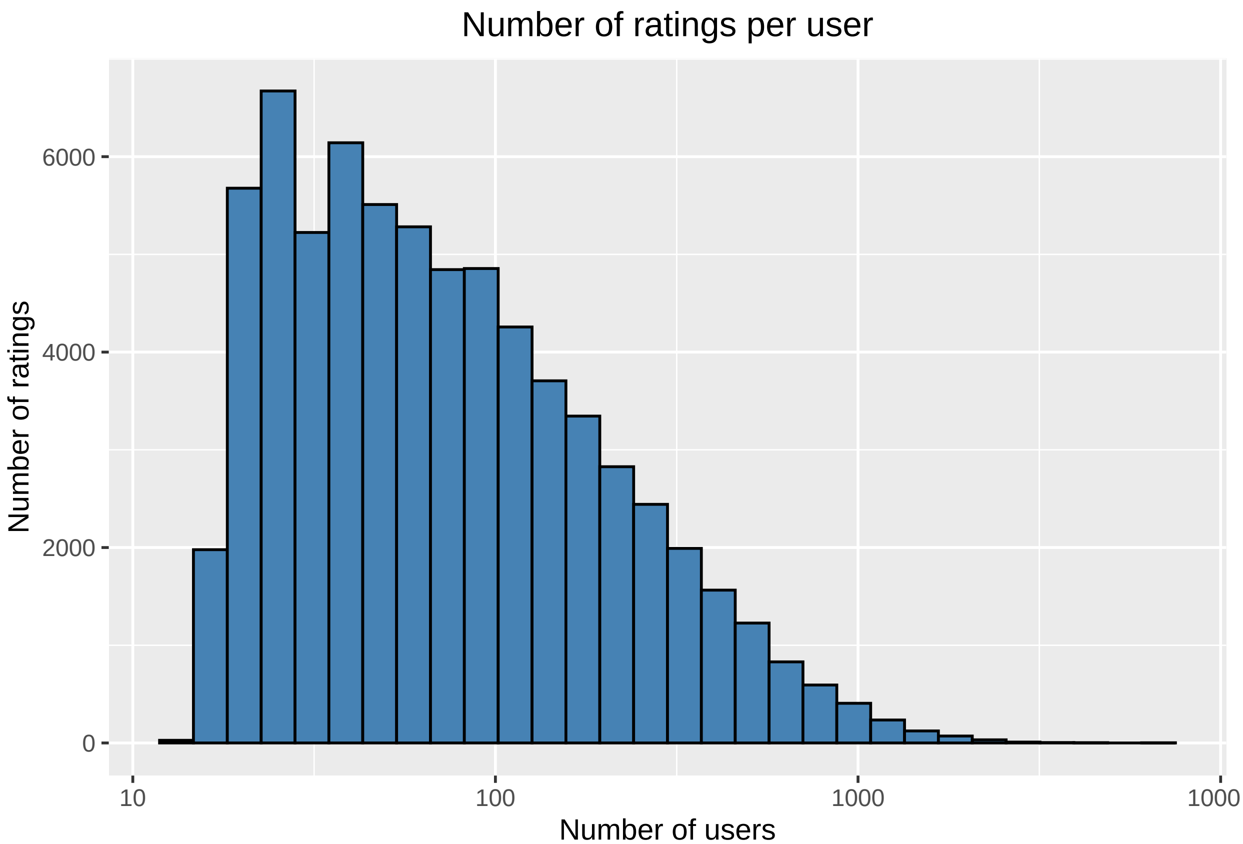


This should not surprise us given that there are blockbusters watched by millions and artsy independent movies wathced by just a few.

2.2.5. User Bias

A second observation is that some users are more active than others at rating movies. Notice that some users ahve rated over 1,000 movies while others have only rated a handful.

```
edx %>% count(userId) %>% ggplot(aes(n)) +
  geom_histogram(bins = 30, fill = 'steelblue', col = 'black') +
  scale_x_log10() +
  xlab('Number of users') +
  ylab('Number of ratings') +
  ggtitle('Number of ratings per user') +
  theme(plot.title = element_text(hjust = 0.5))
```

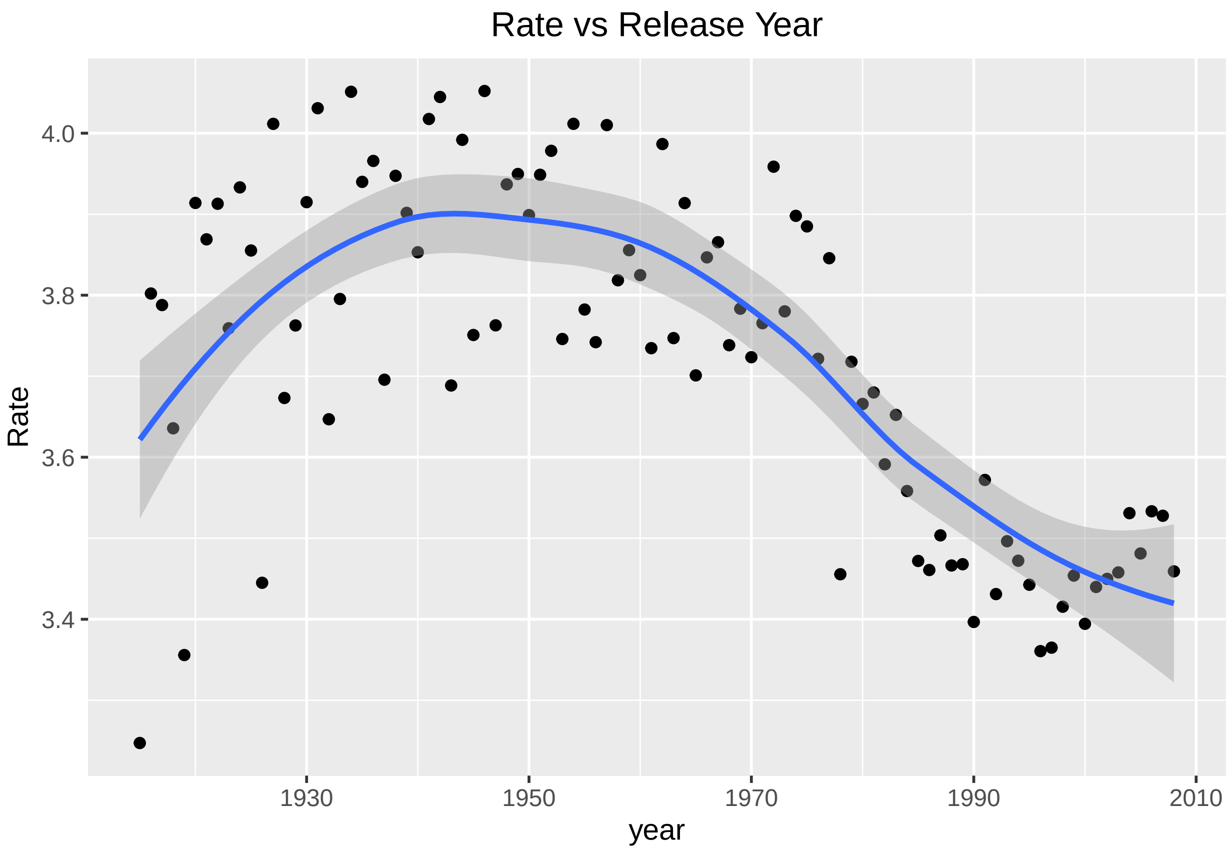


2.2.6. Year Bias

Users' taste also gets pickier over time and thus we should explore the average rating of movies over years.

```
edx %>% group_by(year) %>% summarize(rating = mean(rating)) %>% ggplot(aes(year, rating)) +
  geom_point() +
  geom_smooth() +
  ylab('Rate') +
  ggtitle('Rate vs Release Year') +
  theme(plot.title = element_text(hjust = 0.5))
```

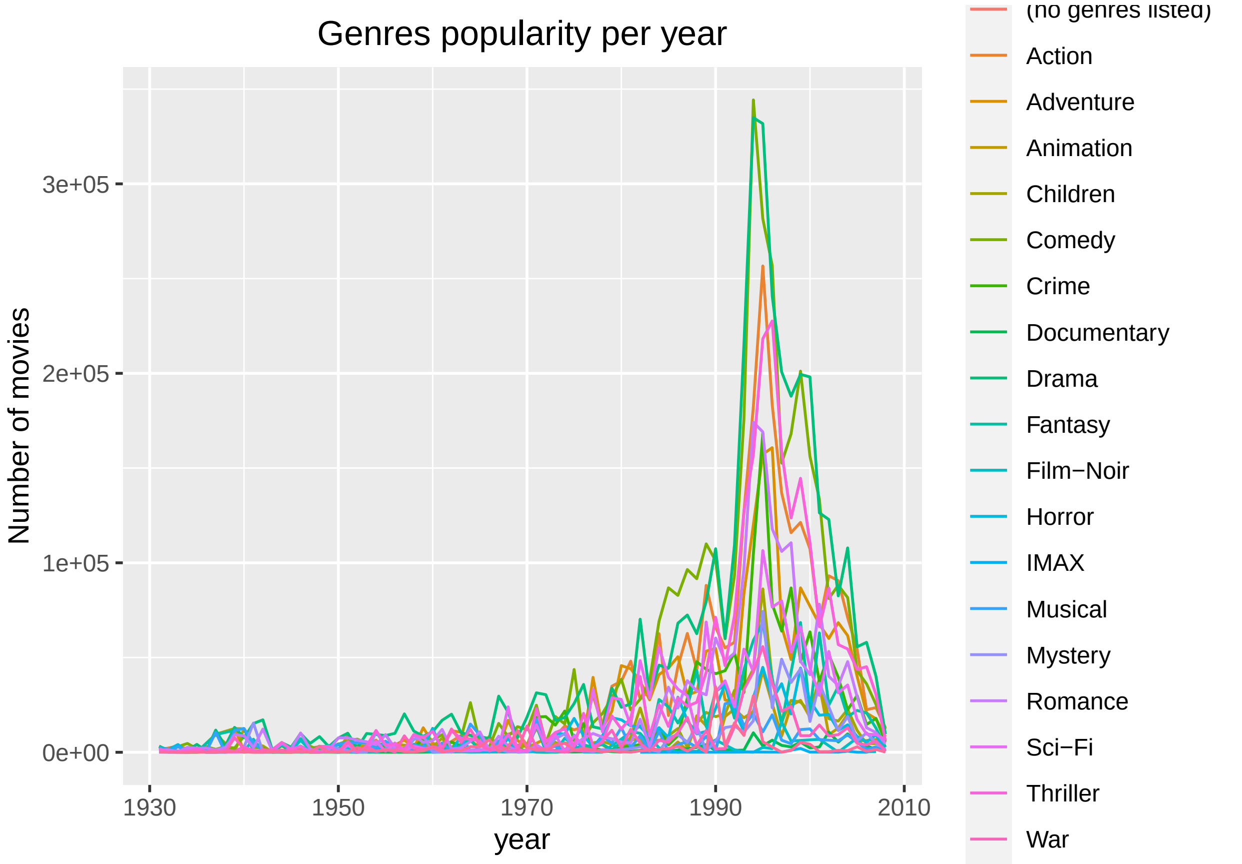
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



2.2.7. Genre Bias

The popularity of the movie genre depends strongly on the contemporary issues. Therefore, we should explore the genre popularity over years.

```
edx_genres %>%
  group_by(year, genres) %>%
  summarize(number = n()) %>%
  filter(year > 1930) %>%
  ggplot(aes(year, number)) +
  geom_line(aes(color = genres)) +
  ylab('Number of movies') +
  ggtitle('Genres popularity per year') +
  theme(plot.title = element_text(hjust = 0.5))
```



3. Modeling Approach

Firstly, we create the loss fuction, RMSE as shown below:

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

3.1. Simplest Model (Average Movie Rating Model)

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:

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

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

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

```
## [1] 3.512464
```

```
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.060651
```

We compute this average on the training data. And then we compute the RMSE on the test set data. We get a RMSE of about 1.05. That's pretty bit.

```
rmse_results <- data_frame(method = 'Just the average', RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

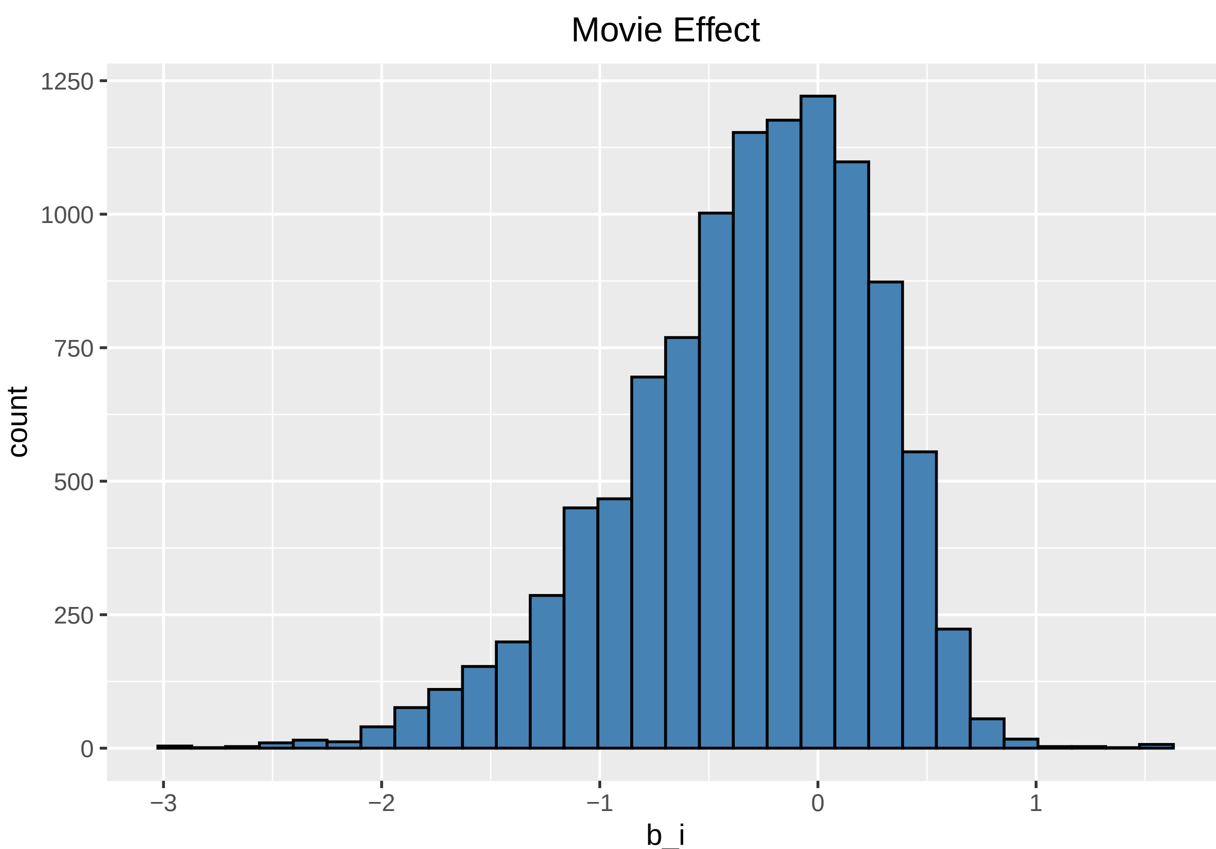
3.2. Movie Effect Model

We know from experience that some movies are just generally rated highter than others. We can see this by simply making a plot of the average rating that each movie got as shown in the Rate Distribution plot. Thus, our intuition that different movies are rated differently is confirmed by data. We can augment our previous model by adding a term, b_i, that represents the average rating for movie i:

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

b_i is the average of $Y_{u,i}$ minus the overall mean for each movie i.

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% ggplot(aes(b_i)) +
  geom_histogram(bins = 30, fill = 'steelblue', col = 'black') +
  ggtitle('Movie Effect') +
  theme(plot.title = element_text(hjust = 0.5))
```



The movie effect can be taken into account by taking the difference form mean rating as shown in the following chunk of code.

```r
predicted_rating <- mu + validation %>%
  left_join(movie_avgs, by = 'movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(validation$rating, predicted_rating)
model_1_rmse
```

```
## [1] 0.9437046
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = 'Movie Effect Model',
                                     RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model	0.9437046

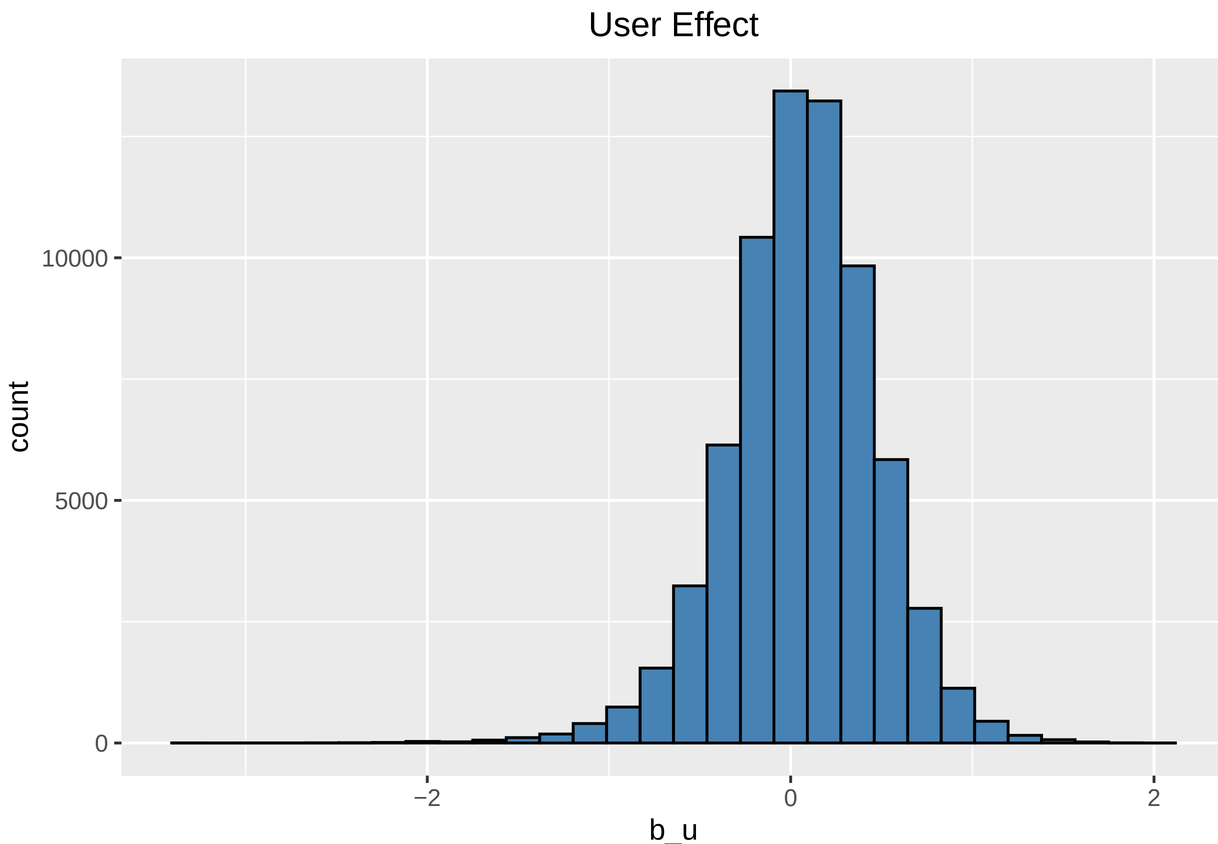
The error drops by 11% after we apply the movie effect into our model.

3.3. Movie and User Effect Model

Different users will rate movies differently. We can make a histogram of those values as shown before. Note that there is substantial variability across users as well. Some cranky users may rate a good movie lower or some users love every move they watch. Thus, we can further improve our model by adding b_u, the user-specific effect:

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

```r
user_avgs <- edx %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
user_avgs %>% ggplot(aes(b_u)) +
  geom_histogram(bins = 30, fill = 'steelblue', col = 'black') +
  ggtitle('User Effect') +
  theme(plot.title = element_text(hjust = 0.5))
```



Now, if a cranky user (negative b_u) rates a great movie (positive b_i), the effects counter each other and we may be able to correclty predict that this user gave a great movie a three rather a five, which will happen. And that should improve our predictions.

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

model_2_rmse <- RMSE(validation$rating, predicted_rating)
model_2_rmse
```

```
## [1] 0.8655329
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = 'Movie and User Effect Model',
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model	0.9437046
Movie and User Effect Model	0.8655329

We see now we obtain a further improvement.

3.4. Regularized Movie and User Effect Model

To further improve our model, let's look at the top 10 best movies based on the estimates of the movie effect model.

```
movie_titles <- movielens %>%
  select(movieId, title) %>%
  distinct()

movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487536
Satan's Tango (Sátántangó) (1994)	1.487536
Shadows of Forgotten Ancestors (1964)	1.487536
Fighting Elegy (Kenka erejii) (1966)	1.487536
Sun Alley (Sonnenallee) (1999)	1.487536
Blue Light, The (Das Blaue Licht) (1932)	1.487536
Constantine's Sword (2007)	1.487536
Human Condition II, The (Ningen no joken II) (1959)	1.320869
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237536
Human Condition III, The (Ningen no joken III) (1961)	1.237536

Note that these all seem to be obscure movies. So why did this happen? To see what's going on, let's look at how often they were rated.

```
edx %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

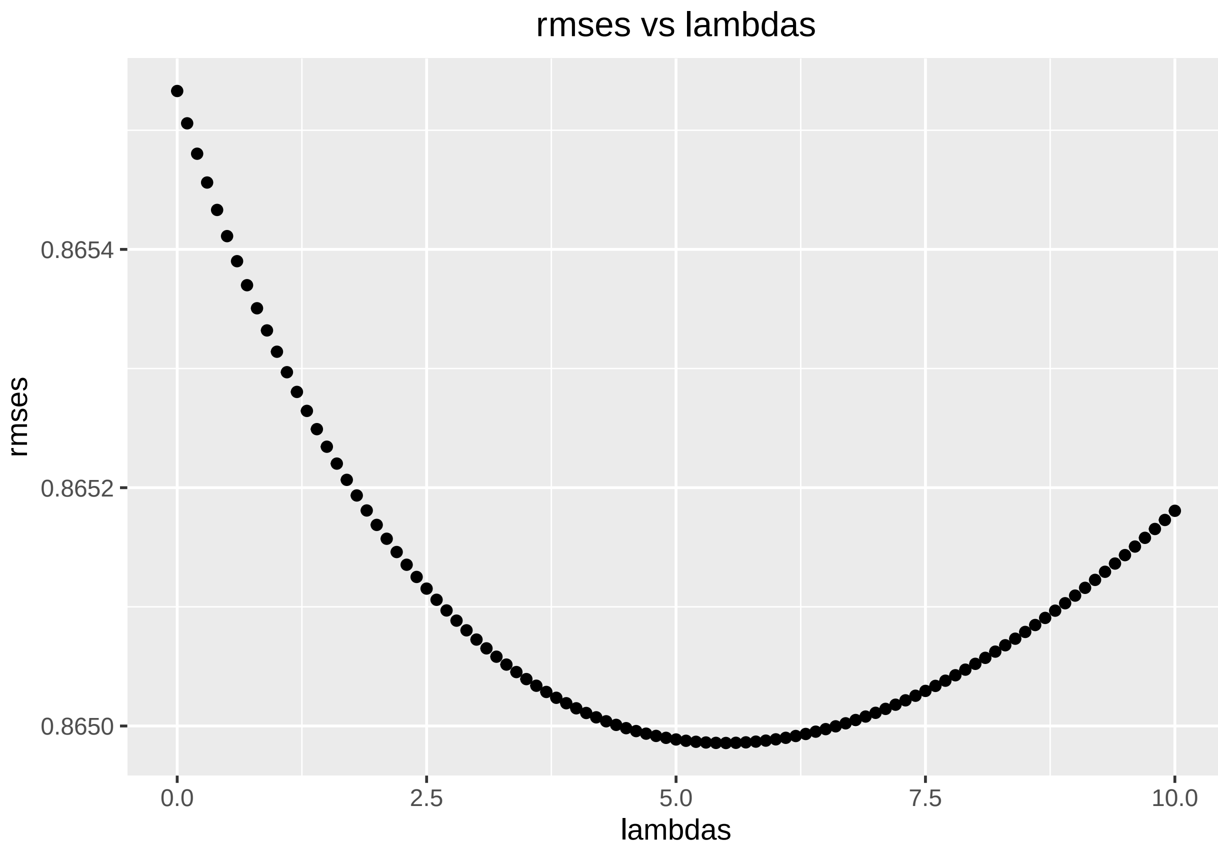
title	b_i	n
Hellhounds on My Trail (1999)	1.487536	1
Satan's Tango (Sátántangó) (1994)	1.487536	2
Shadows of Forgotten Ancestors (1964)	1.487536	1
Fighting Elegy (Kenka erejii) (1966)	1.487536	1
Sun Alley (Sonnenallee) (1999)	1.487536	1
Blue Light, The (Das Blaue Licht) (1932)	1.487536	1
Constantine's Sword (2007)	1.487536	1
Human Condition II, The (Ningen no joken II) (1959)	1.320869	3
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237536	4
Human Condition III, The (Ningen no joken III) (1961)	1.237536	4

Here is the same table, but not we include the number of ratings they received in our training set. Thus, the supposed best movies were rated by very few users, in most cases just one. These movies were mostly obscure ones. This is because with just a few users, we have more uncertainty. Therefore, larger estimates of b_i are more likely when fewer users rate the movies. There are basically noisy estimates that we should not trust, especially when it comes to prediction. To improve our results, we will use regularization. Regularizaion contrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. Here, we should find the optimal value of lambda (tuning parameter) that will minimize the RMSE.

```
lambdas <- seq(0, 10, 0.1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

  b_u <- edx %>%
    left_join(b_i, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n() + l))

  predicted_rating <- validation %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(validation$rating, predicted_rating))
})
```

We plot rmses vs lambdas to select the optimal lambda.

```
data.frame(lambdas, rmses) %>%
  ggplot(aes(lambdas, rmses)) +
  geom_point() +
  ggtitle('rmses vs lambdas') +
  theme(plot.title = element_text(hjust = 0.5))
```



For this model, the optimal lambda is:

```
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.5
```

We use the optimal lambda to compute the RMSE with “validation” dataset

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

user_avgs_reg <- edx %>%
  left_join(movie_avgs_reg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n() + lambda))

predicted_rating <- validation %>%
  left_join(movie_avgs_reg, by = 'movieId') %>%
  left_join(user_avgs_reg, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)


model_3_rmse <- RMSE(validation$rating, predicted_rating)
model_3_rmse
```

```
## [1] 0.8649857
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = 'Regularized Movie and User Effect Model',
                                     RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model	0.9437046
Movie and User Effect Model	0.8655329
Regularized Movie and User Effect Model	0.8649857

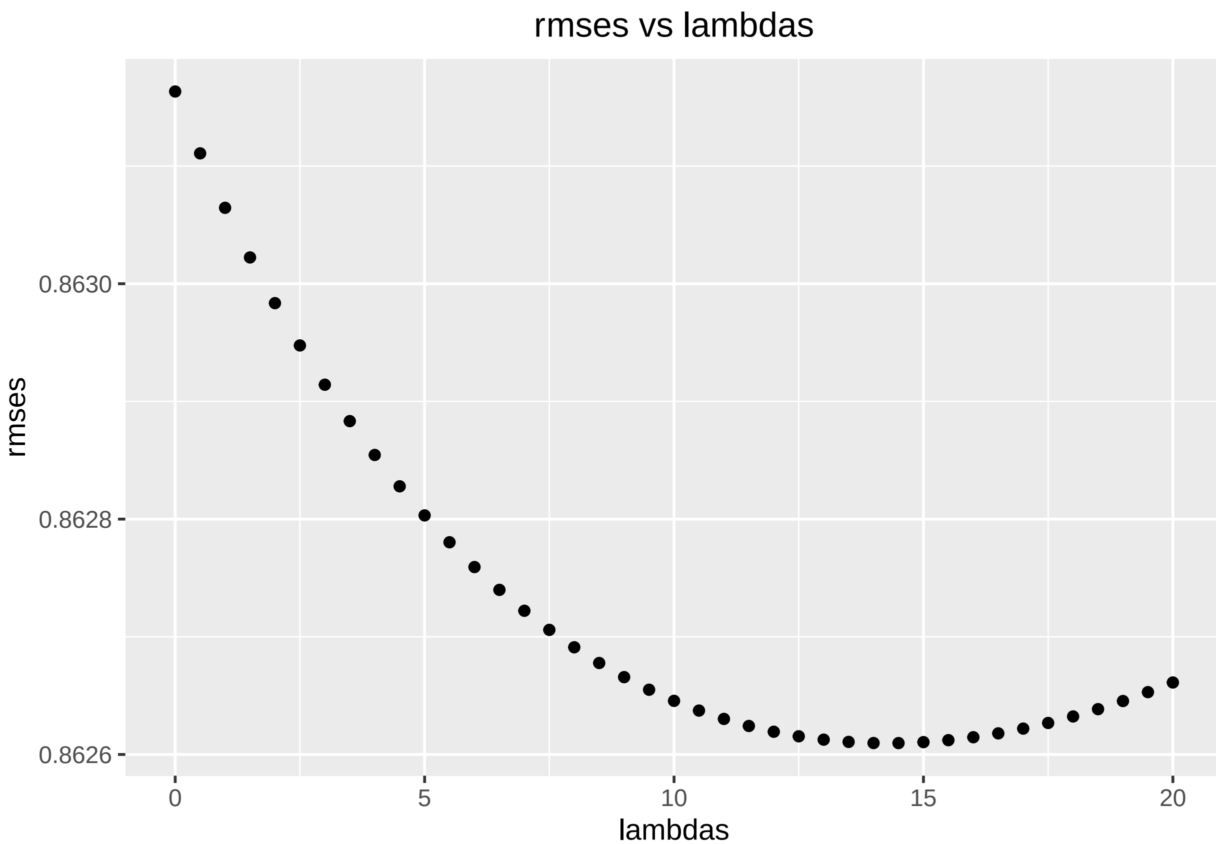
3.4. Regularized Movie, User, Year, and Genre Effect Model

We further apply all features including movie, user, year, and genre into our model.

```
lambdas <- seq(0, 20, 0.5)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)

  b_i <- edx_genres %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

  b_u <- edx_genres %>%
    left_join(b_i, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() + l))

  b_y <- edx_genres %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n() + l))

  b_g <- edx_genres %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_y, by = 'year') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n() + l))

  predicted_rating <- valid_genres %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_y, by = 'year') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    pull(pred)

  return(RMSE(valid_genres$rating, predicted_rating))
})
```

We plot rmses vs lambdas to select the optimal lambda

```r
data.frame(lambdas, rmses) %>%
  ggplot(aes(lambdas, rmses)) +
  geom_point() +
  ggtitle('rmses vs lambdas') +
  theme(plot.title = element_text(hjust = 0.5))
```



For this model, the optimal lambda is:

```r
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 14.5
```

We use the optimal lambda to compute the RMSE with "validation" dataset

```r
movie_avgs_reg <- edx_genres %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

user_avgs_reg <- edx_genres %>%
  left_join(movie_avgs_reg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n() + lambda))

year_avgs_reg <- edx_genres %>%
  left_join(movie_avgs_reg, by = 'movieId') %>%
  left_join(user_avgs_reg, by = 'userId') %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - b_i - b_u)/(n() + lambda))
```

```r
genre_avgs_reg <- edx_genres %>%
  left_join(movie_avgs_reg, by = 'movieId') %>%
  left_join(user_avgs_reg, by = 'userId') %>%
  left_join(year_avgs_reg, by = 'year') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n() + lambda))

predicted_rating <- valid_genres %>%
  left_join(movie_avgs_reg, by = 'movieId') %>%
  left_join(user_avgs_reg, by = 'userId') %>%
  left_join(year_avgs_reg, by = 'year') %>%
  left_join(genre_avgs_reg, by = 'genres') %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  pull(pred)

model_4_rmse <- RMSE(valid_genres$rating, predicted_rating)
model_4_rmse
```

```
## [1] 0.8626097
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = 'Regularized Movie, User, Year and Genre Effect Model',
                                     RMSE = model_4_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model	0.9437046
Movie and User Effect Model	0.8655329
Regularized Movie and User Effect Model	0.8649857
Regularized Movie, User, Year and Genre Effect Model	0.8626097

4. Conclusion

The RMSE values for the used models are shown below:

```r
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model	0.9437046
Movie and User Effect Model	0.8655329
Regularized Movie and User Effect Model	0.8649857
Regularized Movie, User, Year and Genre Effect Model	0.8626097

We can confirm that we have built a machine learning algorithm to predict movie ratings with MovieLens dataset. The RMSE table shows an improvement over different models. The simplest model (Average Movie Rating Model) calculates the RMSE more than 1 which means we may miss the rating by one star. The "Movie Effect" and "Movie and User Effect" model improve the accurancy by 11% and 18.4% respectively. This is a significant improvent due to the simplicity of the model. A deeper exploration to the data reveals that we ignore the overfitting issue. Thus, we applied regularization to get of this issue and further improve our result. In conclusion, the final RMSE is 0.8626097 with an improvement over 18.7% with respect to the simples model.