Data Science: Capstone Project - MovieLens

Elvin Tam

3/10/2021

Introduction

In this report, our goal is to predict the movie rating by using a machine learning algorithm. The data set is coming from MovieLens with about 10M rows of userId, movieId, rating, timestamp, title and genres. Movies are released from early 20th century till 2008. Movie rating are made by user from 1995 to 2009.

Data cleaning is applied to the original data following with data exploration. 4 major effects are identified. Our approach is using normalization to these global effects on baseline rating and regularization (by tuning parameter on lambda) to penalize large estimates that come from small sample size.

1. Movie specific effect
2. User specific effect
3. Genre specific effect
4. Rate per Year specific effect

The evaluation of algorithm is based on root mean squared error (RMSE) of predicted rating against actual rating. Algorithm is trained on train set and being test on test set. Final RMSE is presented basing the on the final hold-out validation set with result in the tier of RMSE < 0.86490.

Method

1. Data Cleaning

edx data set contains 6 columns (user Id, movieId, rating, timestamp, title and genres).

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
```

```
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

In order to facilitate movie rating prediction modelling, pre-process data cleaning is applied to edx data set prior to data partition creation. Title column is split to title and year. Timestamp which is number of second since 1-Jan-1970 00:00:00 is converted to date. Using createDataPatition function from caret package to create train set and test set with percentage of 80% and 20% correspondingly. Semi-join by movieId and userId is applied to test set to avoid NA situation when joining is applied to test set in validation stage.

```
#Data Cleaning
temp <- edx

temp <- temp %>% separate(title, into = c("title", "year"),
                          sep = "\\s\\((?=[0-9]{4}\\))", remove = TRUE) %>%
  mutate(year = as.numeric(str_sub(year, 1, 4))) %>%
  mutate(date = as_datetime(timestamp)) %>% select(-timestamp)

#Create Data Partition
set.seed(12345, sample.kind="Rounding")
test_index <- createDataPartition(y = temp$rating, times = 1,
                                  p = 0.2, list = FALSE)
test_set <- temp[test_index,]
train_set <- temp[-test_index,]

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

rm(temp, test_index)
```

```
head(train_set)
```

```
##    userId movieId rating                title year
## 1:      1     122      5              Boomerang 1992
## 2:      1     185      5               Net, The 1995
## 3:      1     292      5               Outbreak 1995
## 4:      1     316      5               Stargate 1994
## 5:      1     329      5 Star Trek: Generations 1994
## 6:      1     356      5           Forrest Gump 1994
##                            genres                date
## 1:                Comedy|Romance 1996-08-02 11:24:06
## 2:         Action|Crime|Thriller 1996-08-02 10:58:45
## 3:  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01
## 4:       Action|Adventure|Sci-Fi 1996-08-02 10:56:32
## 5: Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32
## 6:      Comedy|Drama|Romance|War 1996-08-02 11:00:53
```

2. Data Exploration

From train set, we can simplify find the average rating across all movies and all users is mu = 3.51 and the end year of all movies is 2008. To avoid dividing by zero in rate per year calculation, we are using 2009 (2008 + 1) as the end year.

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

```
## [1] 3.512451
```

```r
maxyear <- max(train_set$year) + 1
```

```
## [1] 2009
```

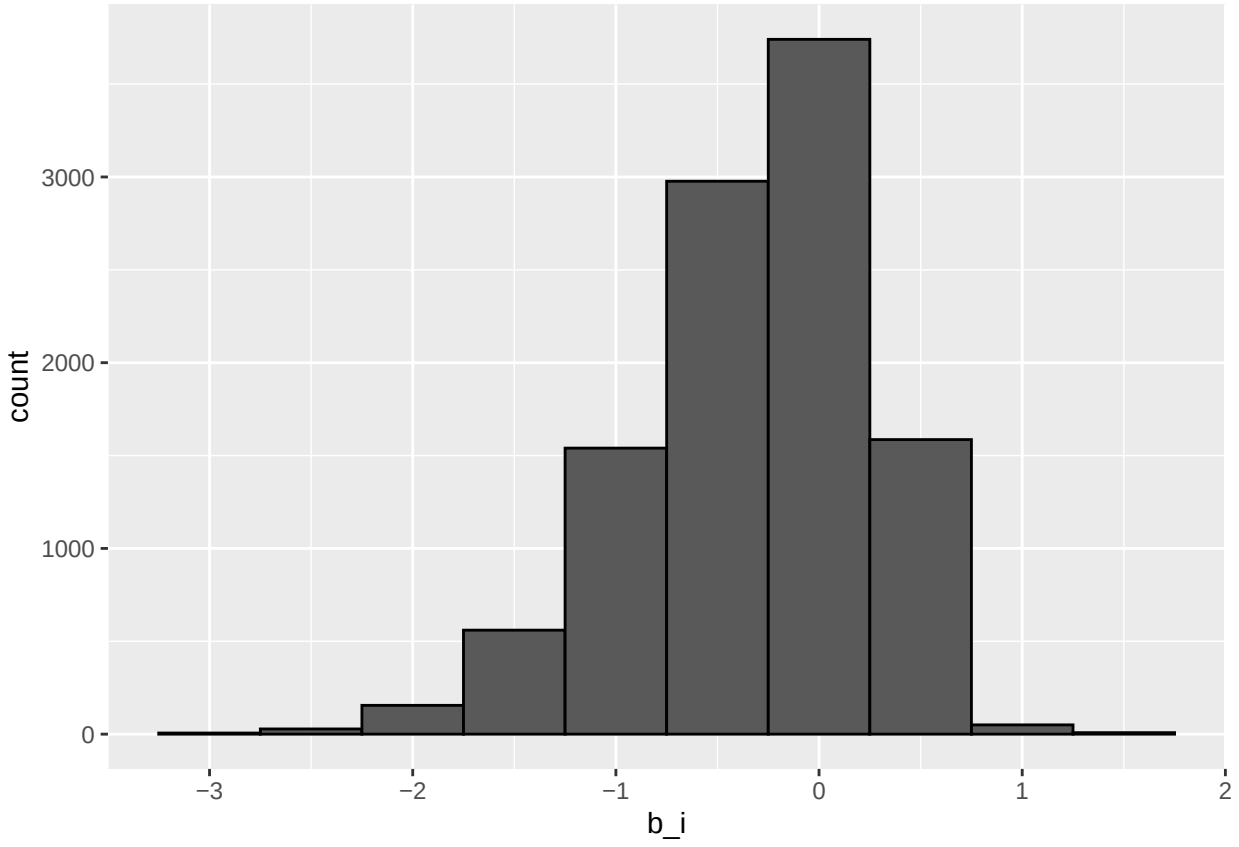
A. Movie specific effect (b_i)

From below chart, we can find that average movie rating adjusted by mu is at -0.32 with distribution skewed to the left side.

```r
train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) %>%
  summarize(avg = mean(b_i)) %>% pull(avg)
```

```
## [1] -0.3187216
```

```r
train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) %>%
  ggplot(aes(b_i)) +
    geom_histogram(bins = 10, color = "black")
```



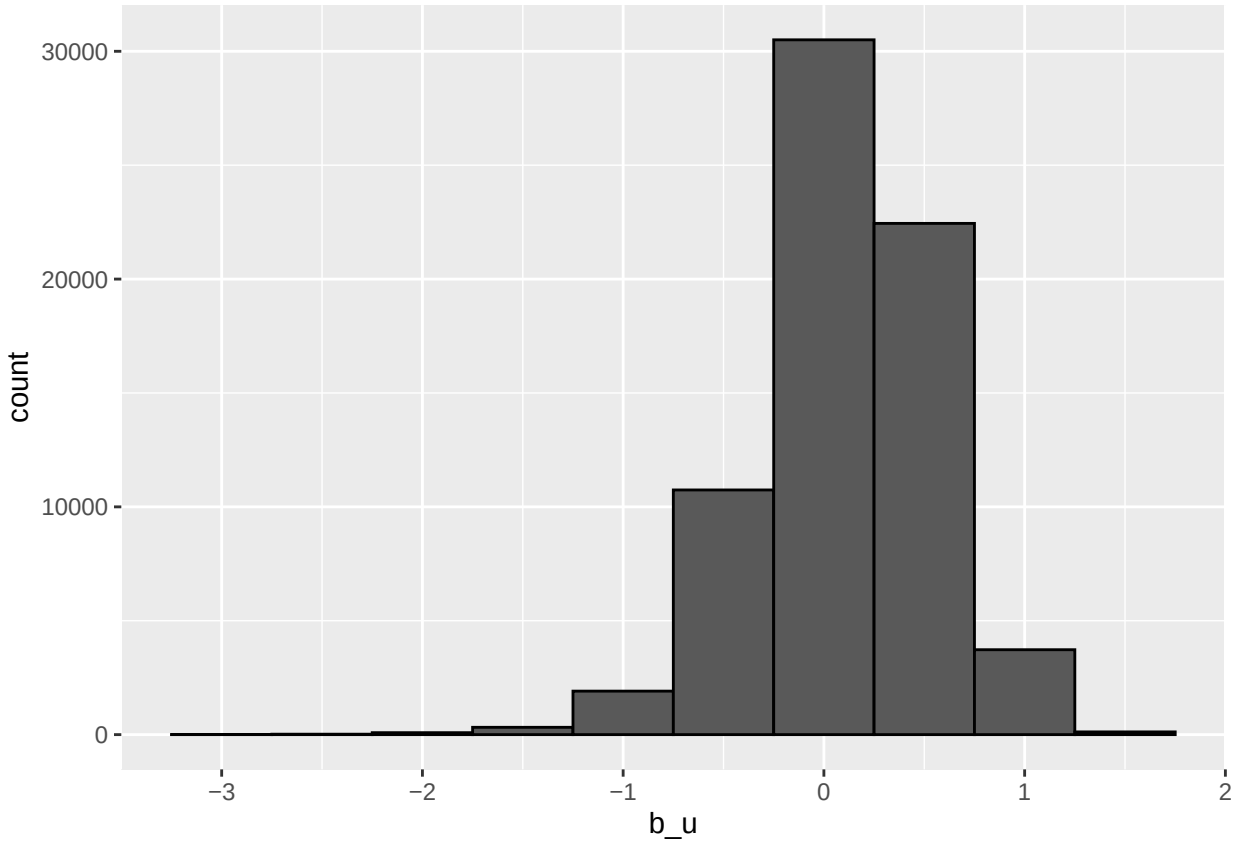
B. User specific effect (b_u)

On the other hand, we can find that average user rating adjusted by mu is 0.10 with more user giving above average rating.

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  summarize(avg = mean(b_u)) %>% pull(avg)
```

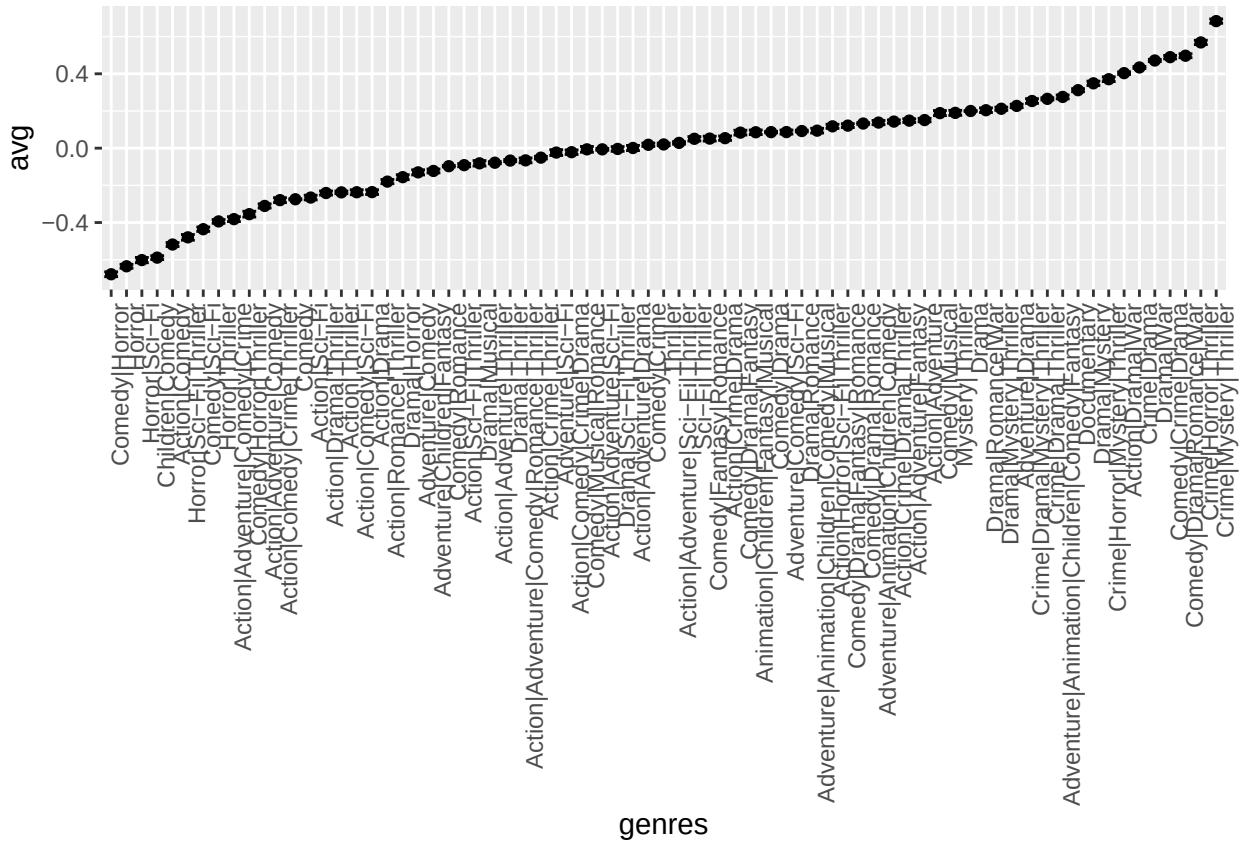
```
## [1] 0.1011655
```

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 10, color = "black")
```



C. Genre specific effect We only illustrate the genres with rating more than 20K times in below chart. We can see that there is clear relation between rating adjusted by mu and genre. The lowest average rating is coming from "Comedy | Horror" while "Crime | Mystery | Thriller" has the highest average rating.

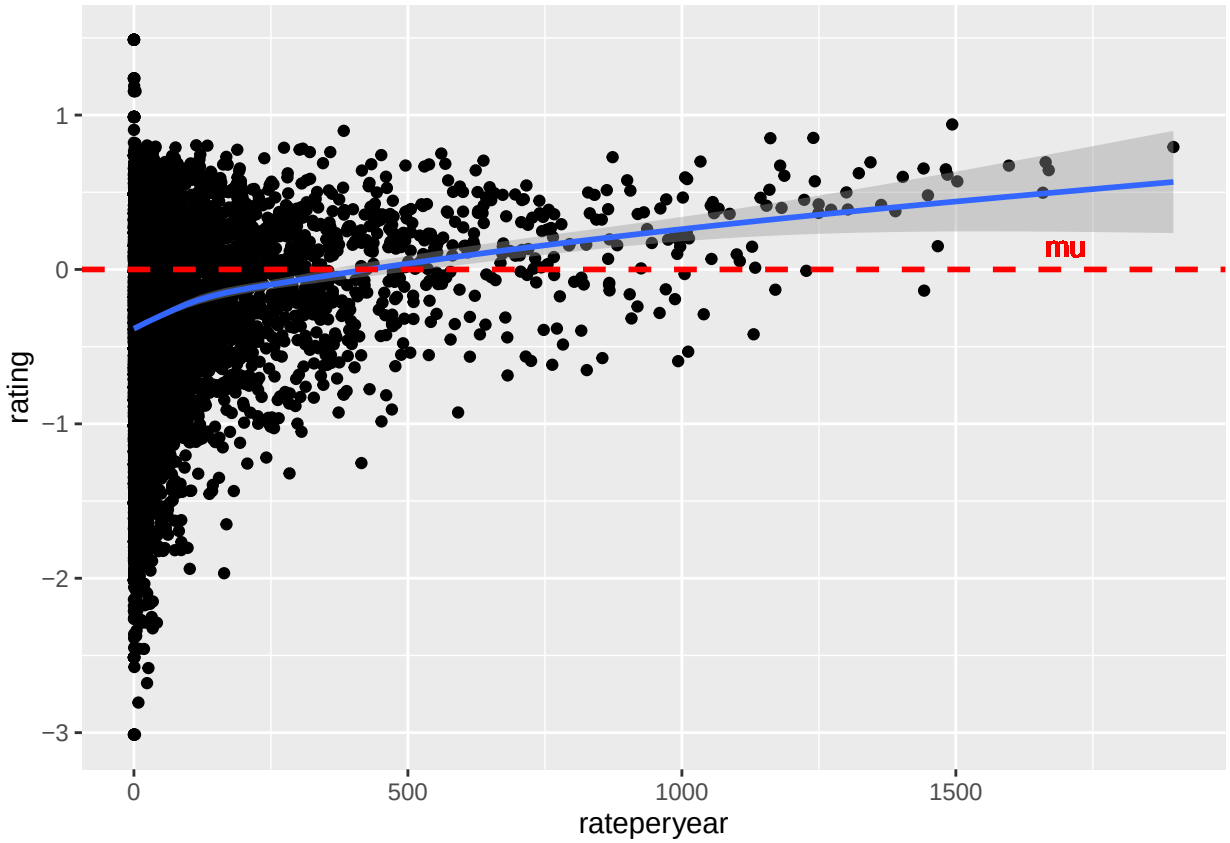
```
train_set %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating - mu), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 20000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



D. Rate per Year specific effect (b_r)

From below chart, we can see that the more often a movie is rated per year, the higher its average rating adjusted by mu (the blue line). Basing on the observation, a lower value will be given if rate per year is lower the corresponding rating per year of mu. For details, please refer to section 3 of modeling approach.

```
train_set %>%
  group_by(movieId) %>%
  summarize(n = n(), years = maxyear - min(year),
            rating = mean(rating - mu)) %>%
  mutate(rateperyear = n/years) %>%
  ggplot(aes(rateperyear, rating)) +
  geom_point() +
  geom_smooth() +
  geom_hline(yintercept = 0, color = "red", size = 1, linetype = "dashed") +
  geom_text(label = "mu", x= 1700, y = 0.15, color = "red")
```



3. Modeling Approach

Normalization of Global Effects

We are using the approach of normalization of global effects in this project. Basing on the above 4 findings, we decompose 4 global effects starting from assuming the same rating (mu, average rating) across all movies and all users. The differences is explained by specific effects from Movie (b_i) / User (b_u) / Genre (b_g) / Rate per Year (b_r) and random variation (eu,i).

Yu,i = mu + b_i + b_u + b_g + b_r + eu,i

Movie / User / Genre effects tables are created by taking average of the rating minus mu and the other bias one by one. Codes are extracted below.

summarize(b_i = sum(rating - mu) / (n() + l))

summarize(b_u = sum(rating - mu - b_i) / (n() + l))

summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + l))

For Rate per Year effect table, a linear regression model fit_rateperyear is created. First, we adjust the rating with mu and above 3 effects and fit the rating with rate per year in a simple linear regression model. Using the regression model, if rate per year is less than the corresponding number of rating per year of average rating, we use model to predict new rating. If not, we keep the original rating. With this approach, we are trying to lower the prediction of rating affected by rate per year.

```
fit_rateperyear <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  left_join(genre_avgs, by="genres") %>%
  mutate(rating = rating - mu - b_i - b_u - b_g) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = maxyear - min(year),
            rating = mean(rating)) %>%
  mutate(rateperyear = n/years) %>%
  lm(rating ~ rateperyear, data = .)

rateperyear <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  left_join(genre_avgs, by="genres") %>%
  mutate(rating = rating - mu - b_i - b_u - b_g) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = maxyear - min(year),
            rating = mean(rating)) %>%
  mutate(rateperyear = n/years,
         pred = ifelse(n < (mean(rating) - fit_rateperyear$coef[1])/fit_rateperyear$coef[2],
                       fit_rateperyear$coef[1] + fit_rateperyear$coef[2] * rateperyear,
                       rating )) %>%
  mutate(b_r = pred - mean(rating)) %>%
  select(movieId, b_r)
```

Regularization

Movie / User / Genre effects are regularized with to penalize large estimates that come from small sample size with the shrunk prediction. l(lambda) is a tuning parameter using cross-validation to choose minimum RSME on train set only. This process doesn't apply to Rate per Year effect because this effect has already penalized the prediction with lower-than-average Rate per Year. l of 2.9 is chosen.

summarize(b_i = sum(rating - mu) / (n() + l))

summarize(b_u = sum(rating - mu - b_i) / (n() + l))

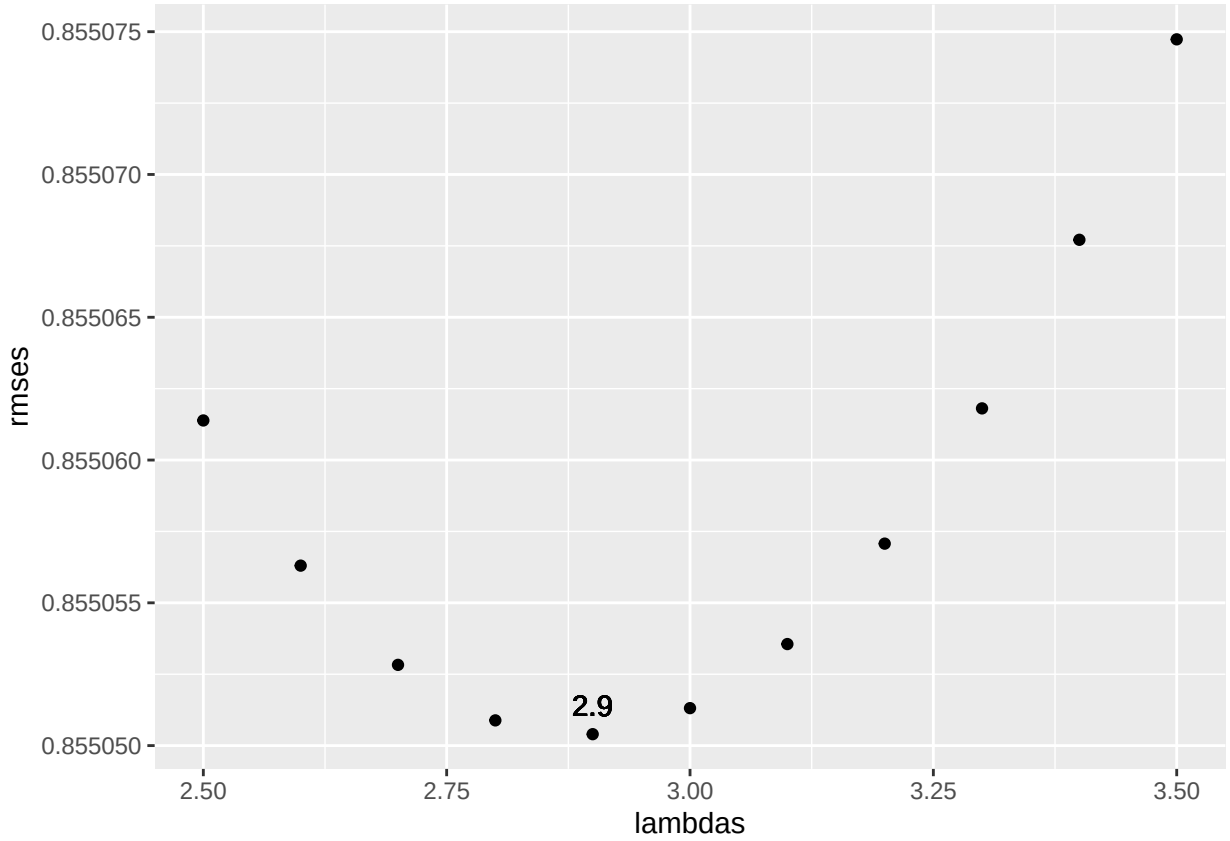
summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + l))

```
## [1] "min RMSE"
```

```
## [1] 0.8550504
```

```
## [1] "Lambdas of min RMSE"
```

```
## [1] 2.9
```



Prediction Restraint and NA handling for Movie / User outside of Train Set

Since our target is to predict the movie rating which is only from 0 to 5. It is not meaningful to predict a rating less than 0 or greater than 5. As a result, prediction is restrained to 0 to 5 with below code.

```
mutate(pred = ifelse(pred <0, 0, ifelse(pred >5 , 5, pred)))
```

It is possible to encounter a movie or a user in validation set that does not appear in the train set. Under this scenario, rating of mu is predicted.

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

removed1 <- validation %>% anti_join(temp, by = "movieId")

removed2 <- validation %>% anti_join(temp, by = "userId")

temp <- rbind(temp, removed1, removed2)

predicted_ratings <- c(predicted_ratings, rep(mu, nrow(removed1) + nrow(removed2)))
```

Result

Both RMSE of test set and validation set are in the tier of RMSE < 0.86490.

```
## [1] "Test Set"
```

```
## [1] 0.8643338
```

```
## [1] "Validation Set"
```

```
## [1] 0.8647539
```

The most time-wasting process is regularization since it involves lambda section with selected values. Wider range and wider increment are applied first and then narrow the range to 1 and increment to 0.1.

lambdas <- seq(1, 10, 1)

lambdas <- seq(2, 5, 0.25)

lambdas <- seq(2.5, 3.5, 0.1)

The second time-wasting process is data cleaning specially on the split of title and year from the original title column. The main reason is the separator using Regex to detect the pattern and handle different scenarios.

Conclusion

This report is using the approach of Normalization of Global Effects and Regularization to capture the main effects in the data. Result RMSE is quite significant in the tier of RMSE < 0.86490 with handling the 4 effects of Movie / User / Genre / Rate per Year. We can find that these baseline effects have clear impact on the rating distribution under data exploration.

In addition to the baseline effects, more sophisticated models can be applied, like Neighborhood Model and Matrix factorization. Neighborhood Model (Movie-Movie approach / User-User approach) can identify similar movie and user that are similar to each other. Their ratings are closed to each other. Matrix factorization (SVD / PCA) can identify the latent factors like coefficient of different genres or relevant impact on big name serial movies.