Movielens Report

Chul Hee Kim

2021 4 21

Overview

This project is designed to build a machine learning algorithm for a movie recommendation system. We will use the *Movielens* dataset provided by Grouplens, which includes about 10 million ratings along with user, movie, and date information.

Movielens has 10,000,054 observations in rows and six variables in columns: userId, movieId, rating, timestamp, title, and genre. Our goal is to find an algorithm that predicts ratings based on these variables. We will only use the linear regression model in this project and add biases in the model to make a better prediction. Root Mean Squared Error (RMSE) will be used to confirm the best model.

Analysis

- Data Preparation **movielens** will be downloaded and separated into **edx** set (90%) and **validation** set (10%). **It is very important to note that the validation set is only for the final test!** It should not be used during training and testing process in the middle of building an algorithm. **edx** set is what we use for training and testing, and only after we find the final model, the **validation** set will be used for the final test.

- Glimpse of Movielens

Table 1: First five rows of Movielens

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

Notable features

- 69,878 users
- 10,677 movies
- **timestamp** is not in an intuitive format. It will be modified in normal date format.
- Each observation has one or more **genres**, which is problematic when used to train an algorithm. It will be modified to having one genre per row.
- **movieId** and **title** gives us the same information. Only **movieId** will be considered in modeling.

Table 2: Modified version of Movielens

userId	movieId	rating	timestamp	title	genres	date
1	122	5	838985046	Boomerang (1992)	Comedy	1996-08-04
1	122	5	838985046	Boomerang (1992)	Romance	1996-08-04
1	185	5	838983525	Net, The (1995)	Action	1996-08-04
1	185	5	838983525	Net, The (1995)	Crime	1996-08-04
1	185	5	838983525	Net, The (1995)	Thriller	1996-08-04

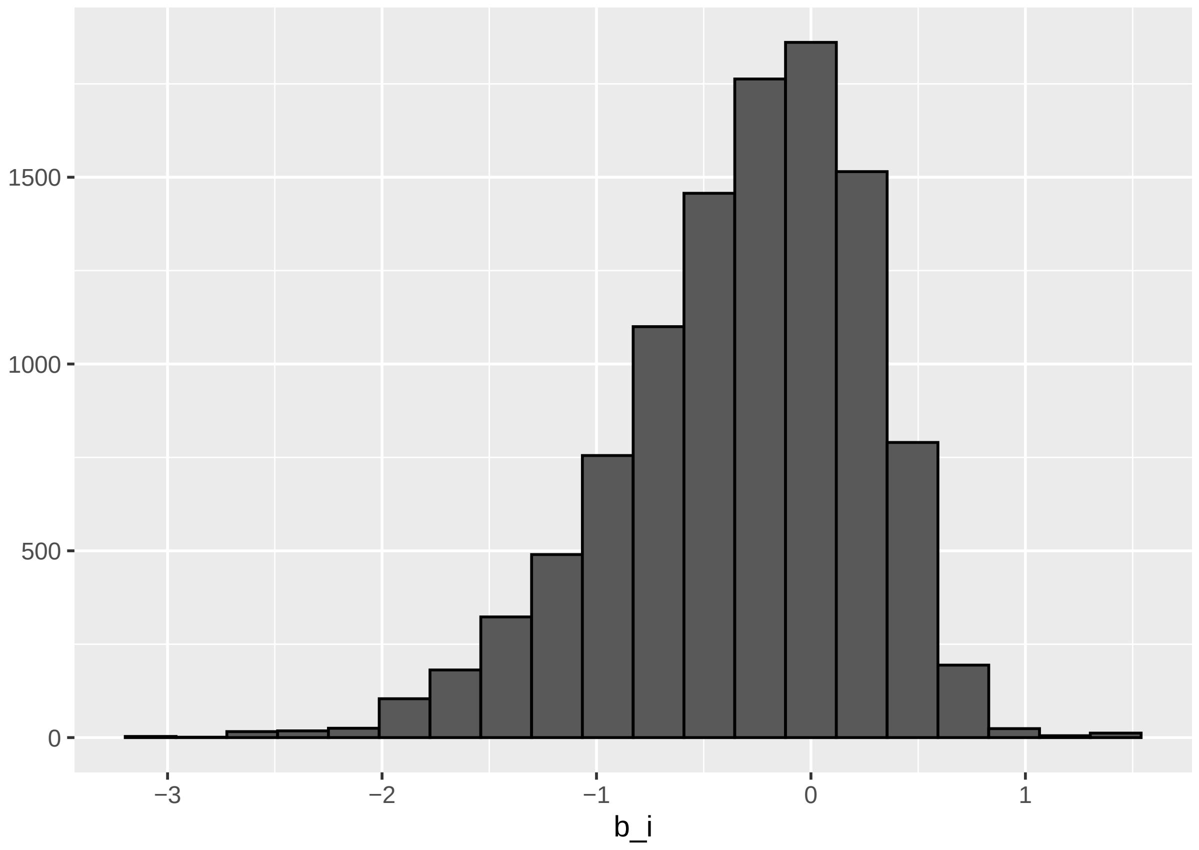
- The new dataset has a new column, **date**.
- **genres** column now has only one genre per row.
- However, because genres are separated, the dataset now has about 2.5 times more rows than the original *Movielens*, which makes the code running longer. Therefore, the original set is used for models that do not include genres and date.

- Modeling Approach (Part 1) Throughout the modeling process, Root Mean Squared Error (RMSE) will be used to assess how closely a model estimated the ratings to the actual observations. RMSE is basically the average of the differences between each prediction and observation. The lower the RMSE, the better.

edx dataset is now separated into **train_set** and **test_set**. We will train multiple models using the **train_set** and make predictions for each model using variables in the **test_set**. Lastly, RMSE is calculated by finding the mean differences between the predictions and the ratings in the **test_set**.

1. Movie (M) Model

Let's start with the movie factor. Some movies are generally rated higher than others. To see it visually, we can group the dataset by **movieId** and find the differences (**b_i**) between each rating and "average of all ratings" (**mu_hat**) in the **train_set**.



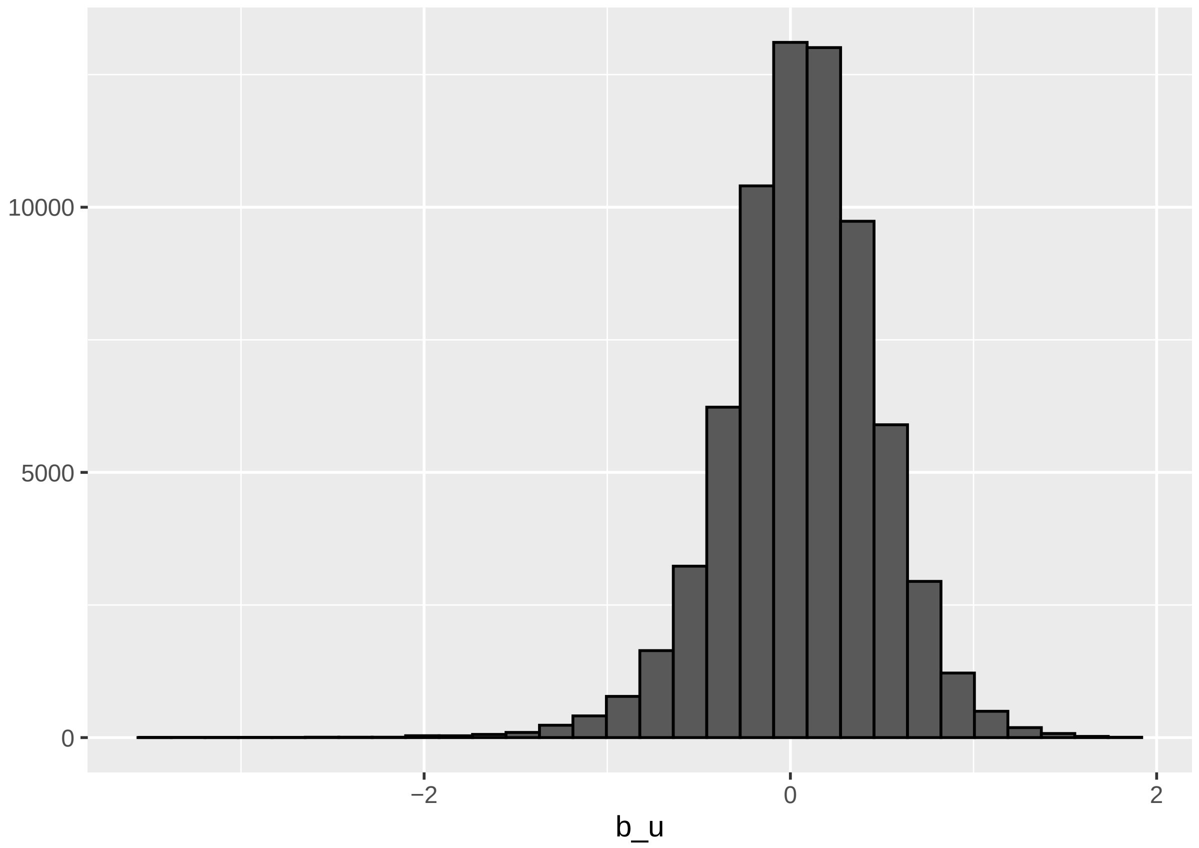
This plot shows that there is significant movie-to-movie variation. In other words, **b_i** explains this variation and helps us make predictions. By adding **b_i** to **test_set**, we get our predictions for this first model by calculating **mu_hat + b_i**.

```
mu_hat <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(pred = mu_hat + b_i) %>%
  .$pred
M <- RMSE(test_set$rating, predicted_ratings)
```

RMSE for the 'M' model is 0.94374. Let's see if we can lower this by adding other factors.

2. Movie + User (MU) Model

We now take into consideration that some users generally give high ratings and some are opposite. Following similar approaches taken in 'M' model, we group the **train_set** by **userId** and find the differences (**b_u**) between each rating and **mu_hat - b_i**.



This plot shows that there is significant user-to-user variation. **b__u** explains this variation and helps us make predictions. By adding **b__u** to **test__set**, we get our predictions for 'MU' model by calculating **mu__hat + b__i + b__u**.

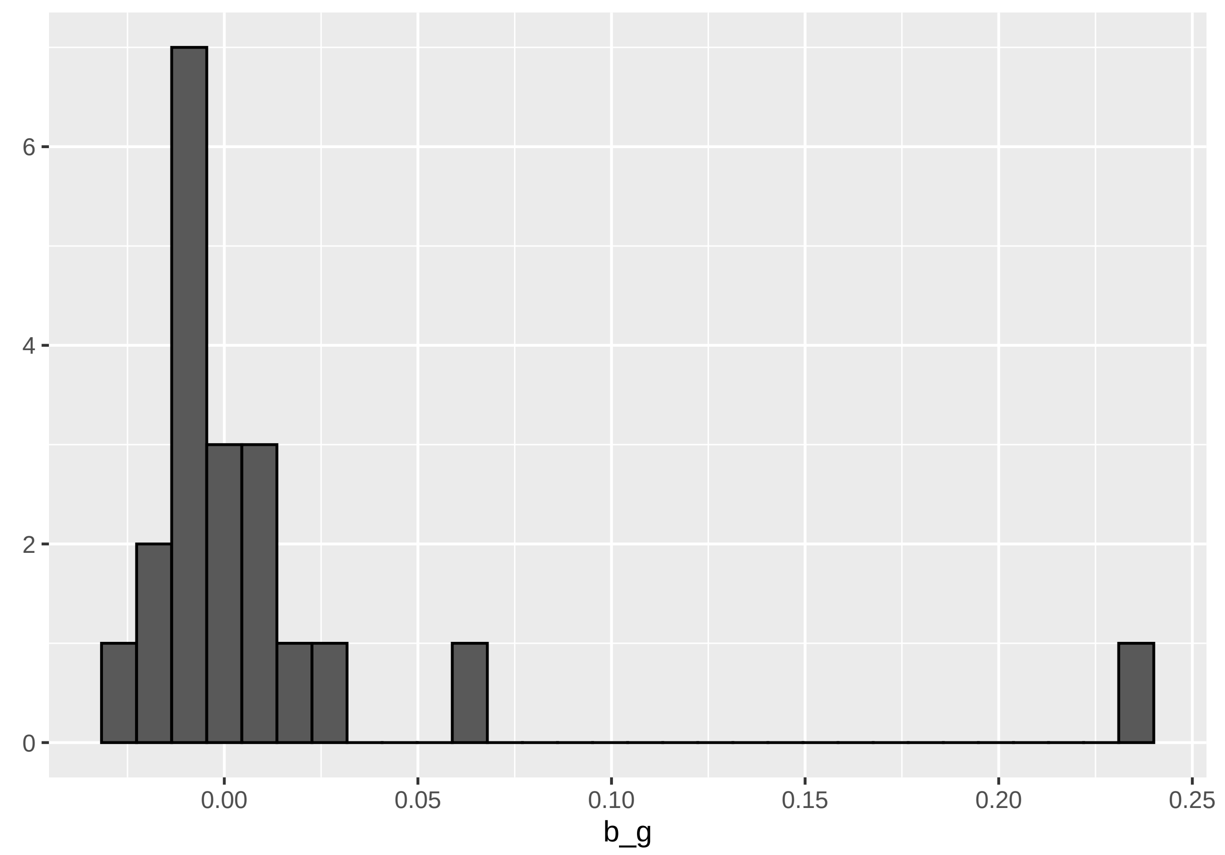
```r
movie_user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(movie_user_avgs, by = "userId") %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred
MU <- RMSE(test_set$rating, predicted_ratings)
```

RMSE for 'MU' model is 0.86593, which is significant improvement from the previous model. Let's now add **genres** to our model.

3. Movie + User + Genre (MUG) Model

Similarly, we now group by **genres** and find the differences (**b__g**) between each rating and **mu__hat - b__i - b__u**.

- This is when we have to modify our dataset and separate genres as explained in Table 2.



This plot shows that there is some, but not much genre-to-genre variation. Let's check if our model makes improvement regardless. By adding **b_g** to **test_set**, we get our predictions for 'MUG' model by calculating **mu_hat + b_i + b_u + b_g**.

```r
genre_avgs <- train_set_separated %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(movie_user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))

predicted_ratings <- test_set_separated %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(movie_user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred

MUG <- RMSE(test_set_separated$rating, predicted_ratings)
```

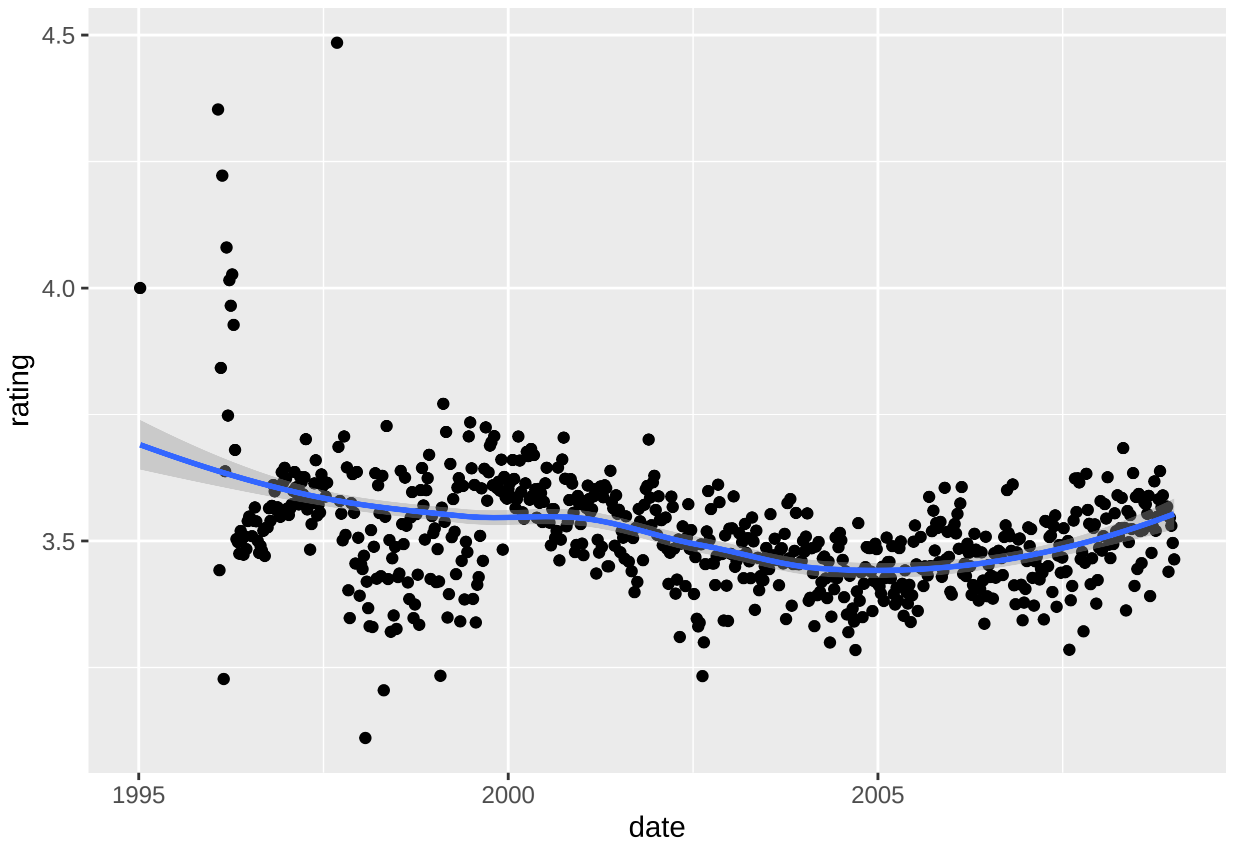
RMSE is now 0.86419. As expected, we only have minor improvement from the 'MU' model.

4. Movie + User + Genre + Time (MUGT) Model

We now group by **date** and find the differences (**b_t**) between each rating and **mu_hat - b_i - b_u - b_g**.

- This is when we have to modify timestamp in the dataset as explained in Table 2.

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



This plot shows that there is some, but not very strong trend. Let's check if our model makes improvement regardless. By adding **b_t** to **test_set**, we get our predictions for 'MUGT' model by calculating **mu_hat + b_i + b_u + b_g + b_t**.

```
time_avgs <- train_set_date %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(movie_user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu_hat - b_i - b_u - b_g))

predicted_ratings <- test_set_date %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(movie_user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(time_avgs, by = "date") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_t) %>%
  .$pred

MUGT <- RMSE(test_set_separated$rating, predicted_ratings)
```

RMSE is now 0.86408. Again as expected, we only have minor improvement from the 'MUG' model.

- Modeling Approach (Part 2) We now introduce *Regularization*.

Table 3: Number of ratings given for top and worst rated movies

	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10
top10	1	3	2	1	1	1	1	1	1	6
worst10	1	1	2	40	168	4	28	11	1	1

This table shows that most of these top and worst movies have too small sample sizes. These are untrustworthy and can potentially increase uncertainty in our models, so we need to make some adjustments. Rather than removing all observations with small samples, we penalize them by regularization. We won't go into details here, but simply speaking, the movies with small samples get penalized by adding a term called **lambda**, which has almost no effect when added to ones with large samples.

1. Regularized Movie + User (RMU) Model

Before we take similar approaches as we did in the previous models, regularization requires us to find the optimal lambda. Cross-validation enables us to find which lambda minimizes RMSE. We inspect lambdas between 0 and 10, incremented by 0.25. General process is the same as 'MU' model, but now we now average residuals by **N + lambda**, rather than just **N**.

```r
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(d){

  mu_hat <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat) / (n() + d))

  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_i) / (n() + d))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    .$pred

  return(RMSE(test_set$rating, predicted_ratings))
})

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

RMSE is 0.86524 with lambda of 4.75. 'RMU' model made some improvement from 'MU' model.

2. Regularized Movie + User + Genre (RMUG) Model

General approach is the same as 'MUG' model but is regularized. We do not show the cross-validation below, but the lambda was actually derived by cross-validating.

```
lambda <- 4.75

b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat) / (n() + lambda))

b_u <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i) / (n() + lambda))

b_g <- train_set_separated %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_hat - b_i - b_u) / (n() + lambda))

predicted_ratings <- test_set_separated %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred

RMUG <- RMSE(test_set_separated$rating, predicted_ratings)
```

RMSE is 0.86355, and 'RMUG' has slight improvement over 'MUG' model.

3. Regularized Movie + User + Genre + Time (RMUGT) Model

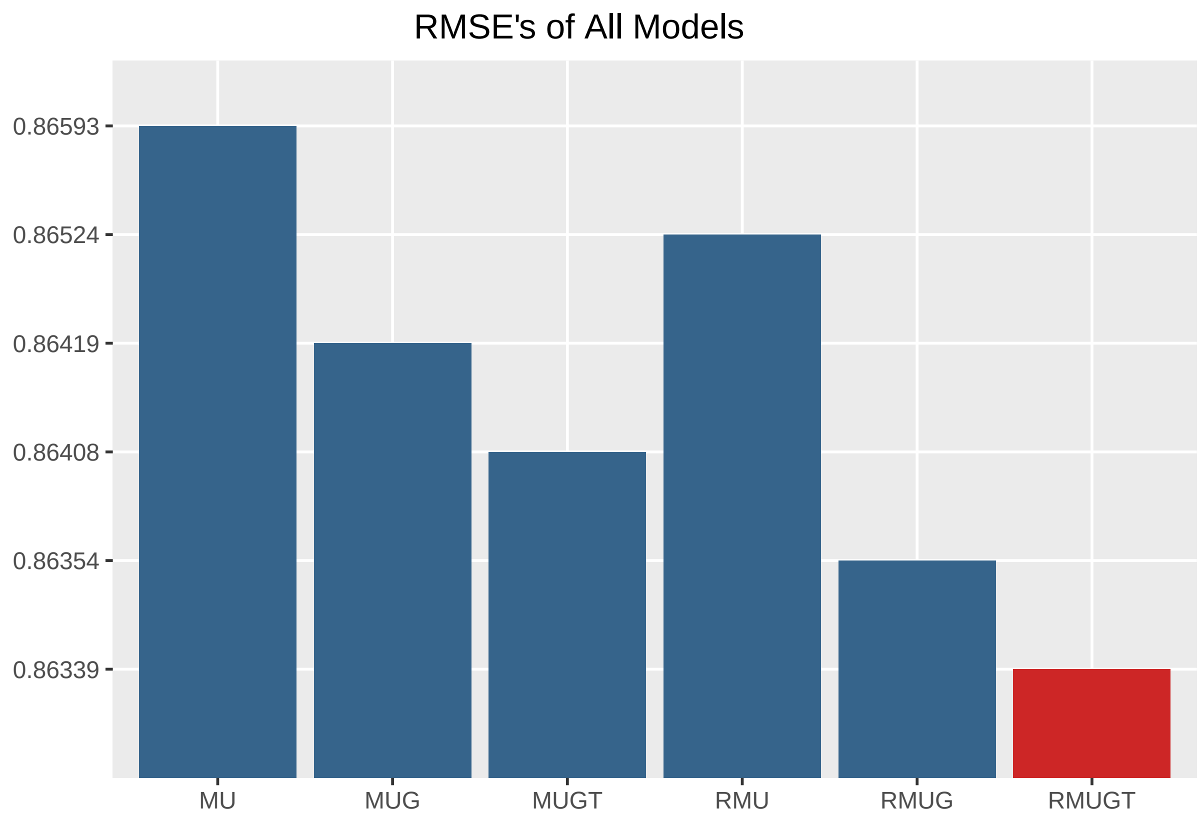
Again, approach is sames as 'MUGT' model but is regularized. We again do not show the cross-validation below, but the lambda was actually derived by cross-validating.

```
lambda <- 5

b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat) / (n() + lambda))

b_u <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i) / (n() + lambda))

b_g <- train_set_separated %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_hat - b_i - b_u) / (n() + lambda))

b_t <- train_set_date %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(date) %>%
  summarize(b_t = sum(rating - mu_hat - b_i - b_u - b_g) / (n() + lambda))

predicted_ratings <- test_set_date %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_t, by = "date") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_t) %>%
  .$pred

RMUGT <- RMSE(test_set_separated$rating, predicted_ratings)
```

RMSE is 0.86339, the lowest of all models!

Results

The plot shown below summarizes all RMSE's from models we have developed.



It is fairly obvious to choose 'RMUGT' model, which is the last one we developed as it generates the lowest RMSE.

Now, it is time to use the **validation** set to test our algorithm. But, before that, we need to train our 'RMUGT' model on the entire **edx** set because so far, we have trained our models on a part of **edx** set. Although not shown here, we should modify **timestamp** and **genres** column in the **edx** and **validation** set as we did in Table 2.

```r
lambda <- 5

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

b_u <- edx %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i) / (n() + lambda))

b_g <- edx_separated %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_hat - b_i - b_u) / (n() + lambda))

b_t <- edx_date %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(date) %>%
  summarize(b_t = sum(rating - mu_hat - b_i - b_u - b_g) / (n() + lambda))

predicted_ratings <- validation_separated %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_t, by = "date") %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_t) %>%
  .$pred

RMSE(validation_separated$rating, predicted_ratings) #> 0.86249
```

The final RMSE using 'RMUGT' model on the **validation** set is 0.86249.

Conclusion

Although we were able to reduce the RMSE with each addition of new variable or regularization, please be reminded that our final result is based only on linear regression model. Also, when we incorporated the time effect, we grouped the date by week, but it might render a better result when grouped by month or by year.

In terms of efficiency of the modeling process, there were frequent instances that we had to modify the dataset and make already large set even larger, resulting in a much longer processing time when running the code. This was the case especially with incorporating **genre** effect, and we could work on finding more efficient ways to cleaning the dataset.

In conclusion, the 'Regularized Movie + User + Genre + Time' (RMUGT) model is what this project derived as the final algorithm for movie recommendation system. Though all four variables are taken into consideration for this model, it is evident that movie and user effect is much greater and related than genre and time effect.