MovieLens Project

Tharawit Disyawongs

2023-10-31

Overview

This project is a part of HarvardX PH125.9x Data Science: Capstone course. The project goal is to predict movie ratings using MovieLens dataset, which contains around 10M ratings of 10k movies from 70k users. In this report, we will start with the overview of the project followed by data preparation. Then, we will perform data analysis along with developing several machine learning algorithms. The performance of each algorithm will be evaluated based on RMSE. Then, we will compare the results and come down to the conclusion, and also provide suggestions for future work.

Data preparation

In this project, we will split MovieLens dataset into two sets, **edx** and **final_holdout_test**. The former will be split further into training and test sets, which will be used for developing machine learning algorithms and comparing the results. Then, the latter, final_holdout_test, will be used for final validation with the final model.

We will start by downloading MovieLens dataset and construct movielens data object.

```r
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# 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")

library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

```r
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

Then, we will separate movielens into edx and final_holdout_test sets.

```r
# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

Now, we have two sets, edx and final_holdout_test. As we want to keep final_holdout_test only for final validation, we will create training and test sets from edx.

```r
set.seed(1)
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
tmp <- edx[test_index,]
train_set <- edx[-test_index,]
```

```r
# Make sure userId and movieId in test set are also in train set
test_set <- tmp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(tmp, test_set)
train_set <- rbind(train_set, removed)

# Remove unused objects
rm(tmp,removed,test_index)
```

Evaluation approach

The approach we use for evaluating results is the Root Mean Square Error (RMSE). RMSE measures the differences between predicted values from a model and the observed values. Basically, the algorithm that produces lower RMSE is considered to be better. In movie recommendation case, when we define $y_{u,i}$ as the rating movie i by user u and define the prediction as $\hat{y}_{u,i}$, the function that computes the RMSE can be defined as the following:

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

Data analysis and modeling approaches

In this section, we will go through several data analysis and modeling approaches. Then, we will present the result of each model.

Model 1: Mean of all ratings

We start with the simplest model - predicting the same rating for all the movies without considering any other effect. In this model, we assume that all the differences are explained by random variation. The model can be defined as

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

where μ is the real average rating for all movies and $\epsilon_{u,i}$ is an independent error sampled from the same distribution centered at 0. In this model, the estimate that minimizes the RMSE is the least squares estimate of μ, and μ can be calculated as the following:

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

```
## [1] 3.512478
```

And we can compute the RMSE of the first model as below.

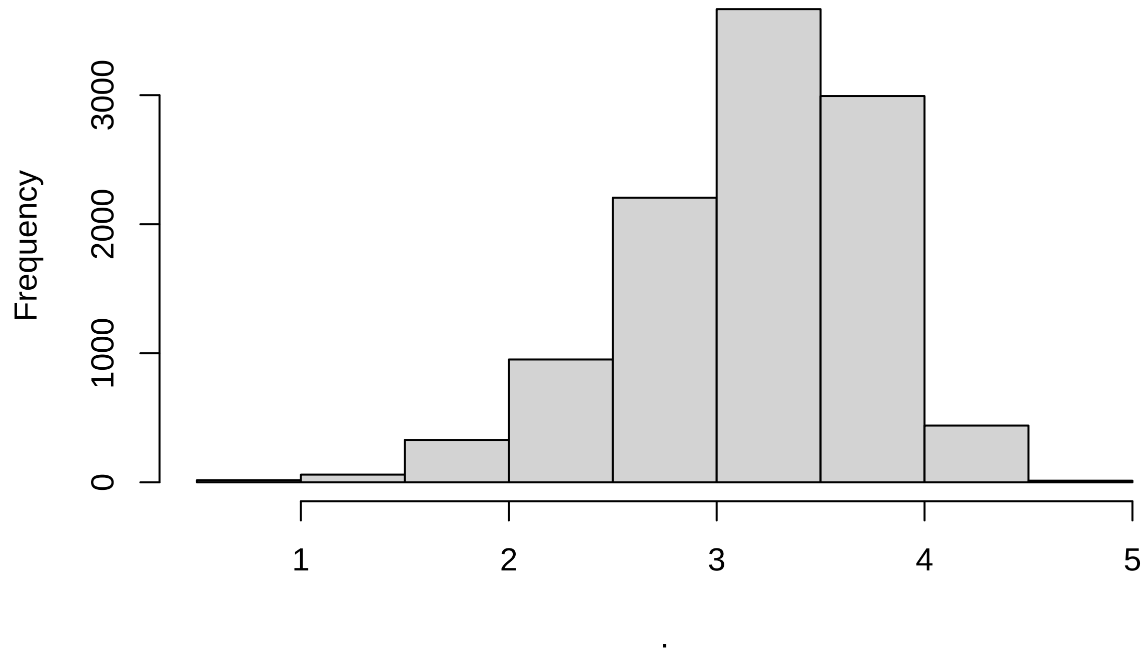
```r
model_1_rmse <- RMSE(test_set$rating, mu)
results <- data_frame(Method = "Model 1: Mean of all ratings", RMSE = round(model_1_rmse,7))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean of all ratings	1.059904

Model 2: Mean + movie effect

In general, we know that some movies are good and some are bad, so they are rated differently, and that cause movie effect. To virtualize the movie effect, we create the histogram of the movie ratings from edx set and observe the distribution.

```
edx %>% group_by(movieId) %>% summarize(mean(rating)) %>% pull() %>% hist(main=NULL)
```

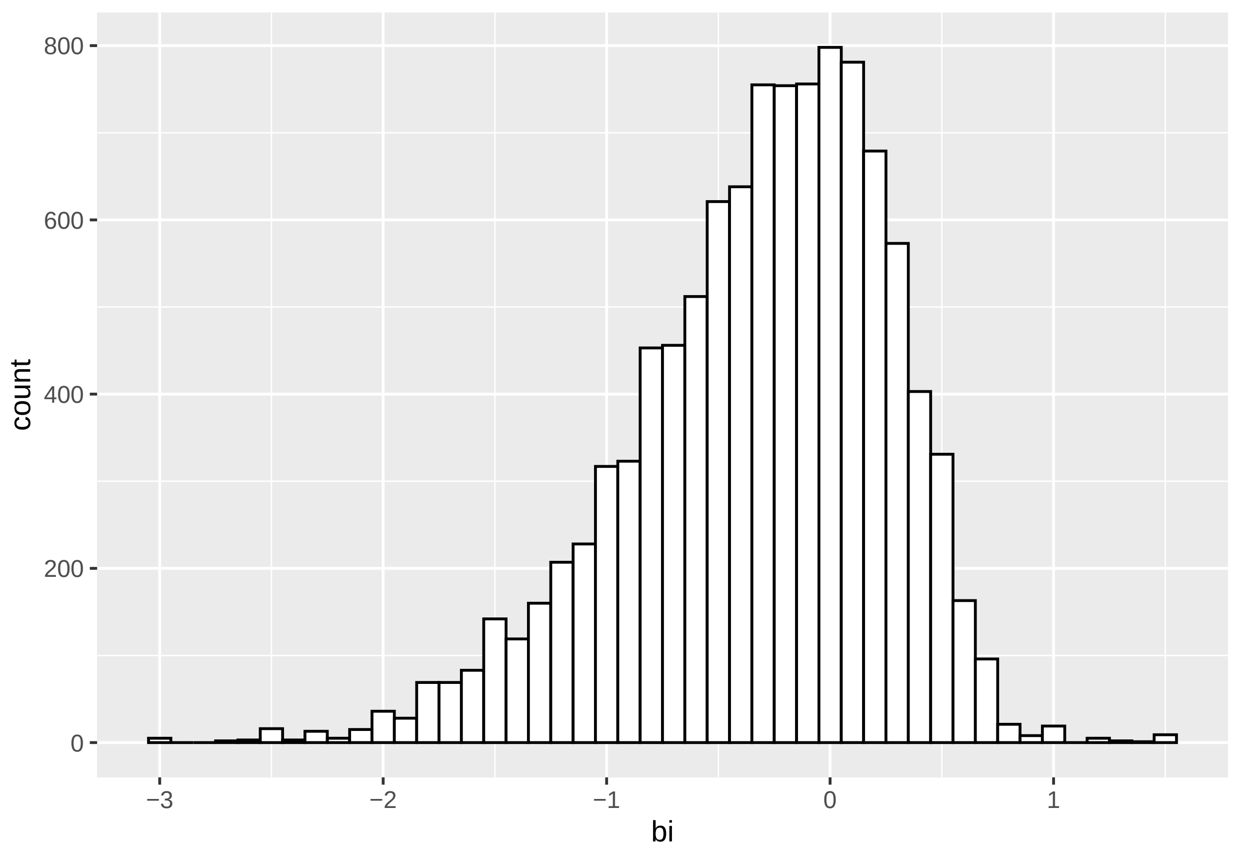


The above histogram tells us that different movies have different average ratings, so we decide to add the movie effect to our model. We use b_i to represent the effect of movie i. Now, the model is

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

As the following, b_i is calculated by the average of the difference between the observed rating and μ. After the computation is done, we can see its distribution via histogram.

```
bi <- train_set %>% group_by(movieId) %>% summarize(bi=mean(rating-mu))
bi %>% ggplot(aes(x=bi)) + geom_histogram(binwidth=0.1,color="black", fill="white")
```



The movie effect histogram is left-skewed, indicating that most movies are penalized, which means they should have average ratings lower than the mean (μ).

Adding movie effect seems to reduce RMSE, as we can see in the following:

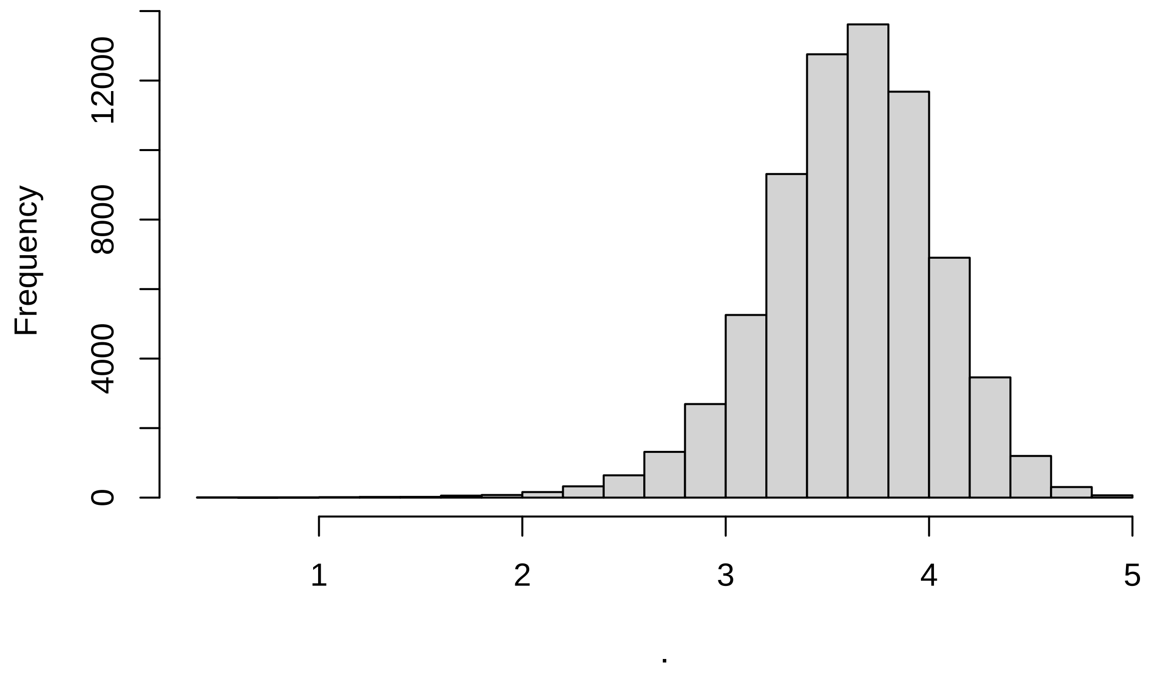
```r
model_2_rmse <- test_set %>% left_join(bi, by = "movieId") %>% mutate(pred = mu+bi) %>%
                summarize(rmse = RMSE(rating, pred)) %>% pull(rmse)
results <- rbind(results,c("Model 2: Mean + movie effect", round(model_2_rmse,7)))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean of all ratings	1.0599043
Model 2: Mean + movie effect	0.9437429

Model 3: Mean + movie effect + user effect

We know that each individual has different taste and preference, and that could impact the way he/she rates movies. Some users might be tough graders and generally give low movie ratings while some users might be generous and usually provide high ratings, and we call that “user effect”. To virtualize user effect, we compute the average rating each user gave, and observe data distribution via histogram as below.

```r
edx %>% group_by(userId) %>% summarize(mean(rating)) %>% pull() %>% hist(main=NULL)
```



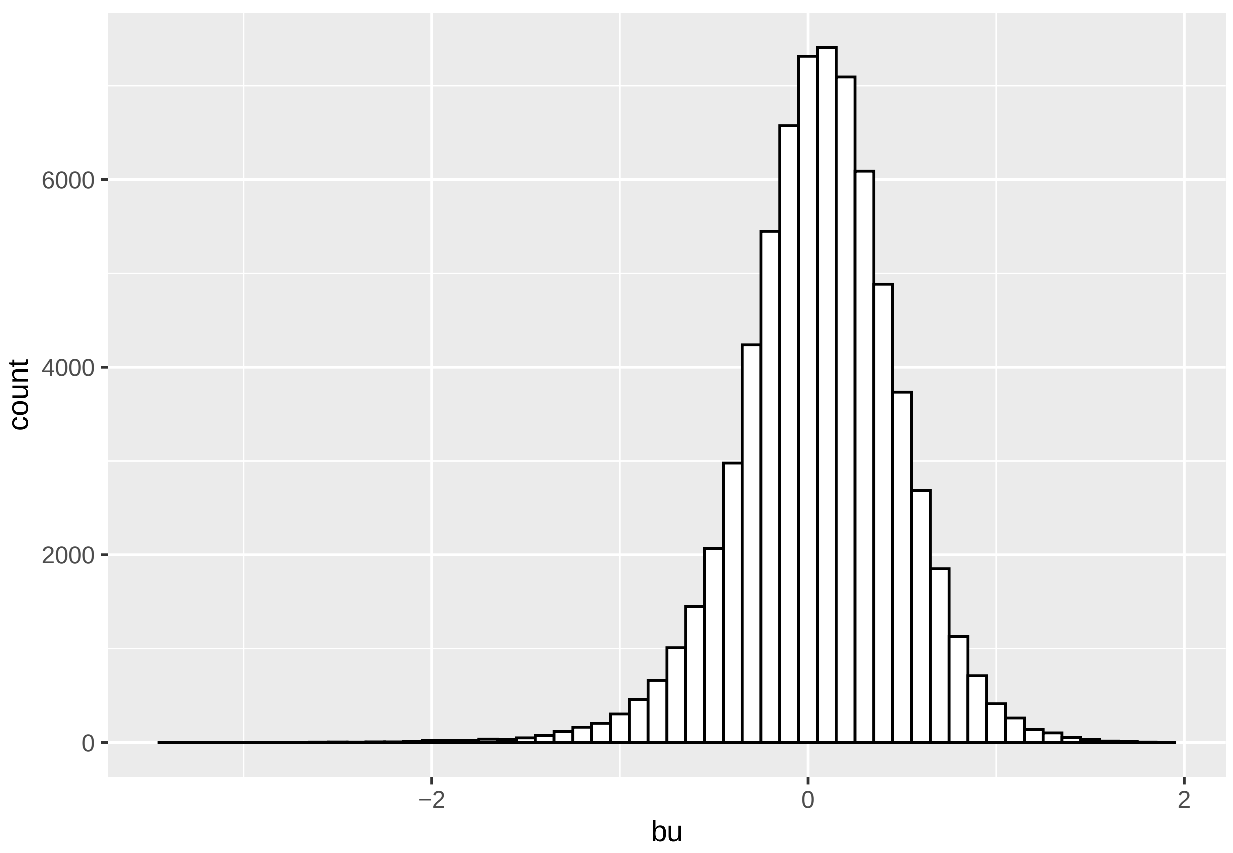
The above histogram seems to indicate user effect, as the average ratings from users are varied. So, we take user effect into consideration. Now, the model is

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

where b_u is user effect.

We construct b_u as below. After the computation is done, we also observe its distribution via histogram, and it looks close to normal distribution.

```r
bu <- train_set %>% left_join(bi,by="movieId") %>% group_by(userId) %>%
      summarize(bu = mean(rating-mu-bi))
bu %>% ggplot( aes(x=bu)) + geom_histogram(binwidth=0.1,color="black", fill="white")
```



Then, we calculate RMSE for this model, and the result is as the following:

```
model_3_rmse <- test_set %>% left_join(bi, by = "movieId") %>% left_join(bu, by = "userId")  %>%
              mutate(pred=mu+bi+bu) %>% summarize(rmse = RMSE(rating, pred)) %>% pull(rmse)
results <- rbind(results,c("Model 3: Mean + movie effect + user effect", round(model_3_rmse,7)))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean of all ratings	1.0599043
Model 2: Mean + movie effect	0.9437429
Model 3: Mean + movie effect + user effect	0.8659319

Model 4: Regularization on movie and user effects

After exploring the edx set, we found that some movies got very high ratings. However, when looking closer, those are unpopular movies rated by very few people.

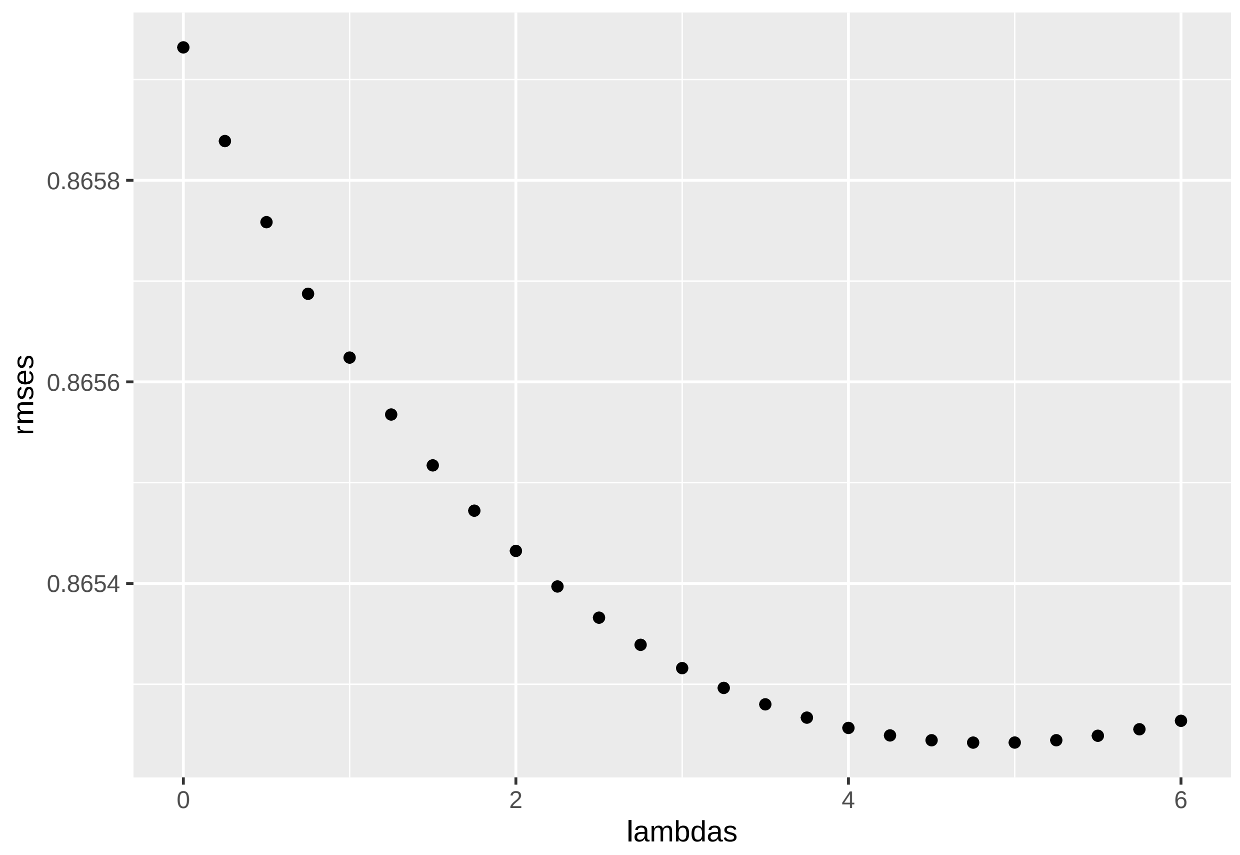
```
edx %>% group_by(movieId) %>% summarize(avg_rating=mean(rating),n=n()) %>%
        filter(avg_rating>4.5 ) %>% left_join(edx, by = "movieId") %>%
        select(movieId,title, avg_rating, n) %>% distinct() %>% arrange(desc(avg_rating))
```

```
## # A tibble: 13 x 4
##    movieId title                                                        avg_rating     n
##      <int> <chr>                                                   <dbl> <int>
##  1    3226 Hellhounds on My Trail (1999)                            5         1
##  2   33264 Satan's Tango (Sátántangó) (1994)                        5         2
##  3   42783 Shadows of Forgotten Ancestors (1964)                    5         1
##  4   51209 Fighting Elegy (Kenka erejii) (1966)                     5         1
##  5   53355 Sun Alley (Sonnenallee) (1999)                           5         1
##  6   64275 Blue Light, The (Das Blaue Licht) (1932)                 5         1
##  7    5194 Who's Singin' Over There? (a.k.a. Who Sings Over Th~     4.75      4
##  8   26048 Human Condition II, The (Ningen no joken II) (1959)      4.75      4
##  9   26073 Human Condition III, The (Ningen no joken III) (196~     4.75      4
## 10   65001 Constantine's Sword (2007)                               4.75      2
## 11    4454 More (1998)                                              4.71      7
## 12    5849 I'm Starting From Three (Ricomincio da Tre) (1981)       4.67      3
## 13   63808 Class, The (Entre les Murs) (2008)                       4.67      3
```

Those movie ratings have very small sample sizes, and could yield a lot of prediction errors. To make our prediction more accurate, we should give a penalty to those with small samples, and this penalty should be gradually reduced once the sample size becomes larger.

In the following code, we use lambda (λ) to penalize b_i and b_u in case of a small sample size. In order to find the optimal lambda, we use cross validation to find the one that minimizes RMSE.

```r
lambdas <- seq(0, 6, 0.25)
rmses <- sapply(lambdas,function(x){
  bi <- train_set %>% group_by(movieId) %>% summarize(bi=sum(rating-mu)/(n()+x))
  bu <- train_set %>% left_join(bi,by="movieId") %>% group_by(userId)  %>%
        summarize(bu=sum(rating-mu-bi)/(n()+x))
  pred_ratings <- test_set %>% left_join(bi, by = "movieId") %>% left_join(bu, by = "userId") %>%
                mutate(pred=mu+bi+bu) %>% pull(pred)
  return(RMSE(test_set$rating, pred_ratings))
})
qplot(lambdas, rmses)
```



As the above plot, the optimal lambda is the following:

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

```
## [1] 4.75
```

And RMSE associated to that lambda is as below:

```r
model_4_rmse <- min(rmses)
results <- rbind(results,c("Model 4: Regularization on movie and user effects", round(model_4_rmse,7)))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean of all ratings	1.0599043
Model 2: Mean + movie effect	0.9437429
Model 3: Mean + movie effect + user effect	0.8659319
Model 4: Regularization on movie and user effects	0.8652421

Model 5: Matrix Factorization

Matrix factorization is a popular approach for recommendation systems which uses historical data to predict the ratings. The main idea is to decompose the user-movie matrix into the product of smaller matrices and find the relationship between users and items (movies). Considering the efficiency of the computation process, **recosystem packgage** https://cran.r-project.org/web/packages/recosystem/ will be used to build the model.

In this model, we will use recosystem in the simplest way without any tuning parameter and observe the result.

First, we start by converting train and test sets into recosystem input format. Then, we use the train object (train_reco) for training the model.

```
library(recosystem)
set.seed(1)
train_reco <- with(train_set, data_memory(user_index = userId, item_index = movieId, rating = rating))
test_reco <- with(test_set, data_memory(user_index = userId, item_index = movieId, rating = rating))
reco <- Reco()
reco$train(train_reco)
```

```
## iter      tr_rmse          obj
##    0       0.9717   1.1953e+07
##    1       0.8828   1.0694e+07
##    2       0.8627   1.0538e+07
##    3       0.8480   1.0377e+07
##    4       0.8423   1.0324e+07
##    5       0.8377   1.0291e+07
##    6       0.8331   1.0260e+07
##    7       0.8293   1.0233e+07
##    8       0.8266   1.0214e+07
##    9       0.8247   1.0202e+07
##   10       0.8234   1.0190e+07
##   11       0.8223   1.0183e+07
##   12       0.8215   1.0176e+07
##   13       0.8209   1.0172e+07
##   14       0.8203   1.0167e+07
##   15       0.8198   1.0162e+07
##   16       0.8195   1.0161e+07
##   17       0.8191   1.0156e+07
##   18       0.8188   1.0154e+07
##   19       0.8186   1.0151e+07
```

After that, we compute RMSE of this model, and the result seems to be much better than the previous ones.

```
results_reco <- reco$predict(test_reco, out_memory())
model_5_rmse <- RMSE(results_reco, test_set$rating)
results <- rbind(results,c("Model 5: Matrix factorization",round(model_5_rmse,7)))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean of all ratings	1.0599043
Model 2: Mean + movie effect	0.9437429
Model 3: Mean + movie effect + user effect	0.8659319
Model 4: Regularization on movie and user effects	0.8652421
Model 5: Matrix factorization	0.834126

Model 6: Matrix Factorization with tuning parameters

From the previous model, the RMSE looks impressive, but can we do it even better? In recosystem packgage, it has regularization parameters, so we will try that and see if they could provide a better result. The tuning parameters that we use are **costp_l2** for user regularization and **costq_l2** for item (movie) regularization.

Please be noted that, for better performance, **nthread** parameter can be tuned so that we get the benefit of better speed from parallel computation when building a model (i.e, we can set it to 4 for a quad-core CPU machine). However, the drawback is that it will not guarantee reproducible results even with set.seed() function used. So, that tuning is not used in the following code.

```
set.seed(1)
opts_tune <- reco$tune(train_reco, opts = list(costp_l2 = c(0.01, 0.1), # user regularization
                                              costq_l2 = c(0.01, 0.1), # movie regularization
                                              nthread = 1))

reco$train(train_reco, opts = opts_tune$min)
```

```
## iter      tr_rmse          obj
##    0       0.9874   9.9376e+06
##    1       0.8803   8.0896e+06
##    2       0.8501   7.5438e+06
##    3       0.8290   7.1959e+06
##    4       0.8129   6.9516e+06
##    5       0.8006   6.7738e+06
##    6       0.7910   6.6411e+06
##    7       0.7833   6.5389e+06
##    8       0.7767   6.4524e+06
##    9       0.7711   6.3840e+06
##   10       0.7664   6.3250e+06
##   11       0.7621   6.2729e+06
##   12       0.7584   6.2312e+06
##   13       0.7551   6.1949e+06
##   14       0.7522   6.1617e+06
##   15       0.7496   6.1336e+06
##   16       0.7472   6.1071e+06
##   17       0.7450   6.0827e+06
##   18       0.7430   6.0627e+06
##   19       0.7412   6.0429e+06
```

```
results_reco <- reco$predict(test_reco, out_memory())
```

After using tuning parameters, we calculate RMSE, and this model seems to generate the best result by far.

```
model_6_rmse <- RMSE(results_reco, test_set$rating)
results <- rbind(results,c("Model 6: Matrix factorization with tuning parameters",
                          round(model_6_rmse,7)))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean of all ratings	1.0599043

Method	RMSE
Model 2: Mean + movie effect	0.9437429
Model 3: Mean + movie effect + user effect	0.8659319
Model 4: Regularization on movie and user effects	0.8652421
Model 5: Matrix factorization	0.834126
Model 6: Matrix factorization with tuning parameters	0.7952232

Final Result

Now, we will perform a final validation by running our final model using final_holdout_test data set.

```r
final_holdout_reco <- with(final_holdout_test, data_memory(user_index = userId,
                                                            item_index = movieId,
                                                            rating = rating))
pred_reco <- reco$predict(final_holdout_reco, out_memory())
final_rmse <- RMSE(final_holdout_test$rating, pred_reco)
results <- rbind(results,c("Final validation: model 6 on final_holdout_test", round(final_rmse,7)))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Mean of all ratings	1.0599043
Model 2: Mean + movie effect	0.9437429
Model 3: Mean + movie effect + user effect	0.8659319
Model 4: Regularization on movie and user effects	0.8652421
Model 5: Matrix factorization	0.834126
Model 6: Matrix factorization with tuning parameters	0.7952232
Final validation: model 6 on final_holdout_test	0.7948455

From the above result, the model of matrix factorization with tuning parameters seems to generate very low RMSE on final validation set.

Conclusion

In this project, we build and test several models, and optimize them along the way. For linear models, using movie and user effects along with regularization provides the minimum RMSE. However, matrix factorization (via recosystem) seems to be a much better approach. Especially, when we use it along with user and movie regularization via tuning parameters, the RMSE can go below 0.8 on validation (final_holdout_test) set, which is quite impressive.

The drawback of matrix factorization with tuning parameters is that it consumes a lot of CPU, memory resource and time. Given that, it would be great if we could find a way to build a model with less resource consumption while not losing its accuracy. So, for future work, there could be some improvements in efficiency and accuracy that we can explore.