Movielens - Movie Recommendation System

Donghan Lee

10/25/2021

1. Introduction

Recommendation system is of keen interest due to its applicability. Companies such as E-commence including Amazon, content streamer including Netflix, Sportify, and YouTube needs this recommendation system for their users. By providing useful contents based on recomend systym, royalty or staying as a subscriber could be achieved. The recommendation system is critical interest.

In this excise, movielens was chosen for building a recomendation system with a benchmark, root mean square error (RMSE). movielens provides rating together with user, movie, time, genres, and title.

1.1 Flow

1. Preparation of data (download, parsing). products: edx and validation data sets.
2. Data analysis with data visualization. products: table and plots.
3. Using various models, recommendation system is tested using train and test sets created for a cross-validation.
4. Apply the selected models to the Hang-out test set, validation.

2. Methods

Due to the size of data, many algorythms such as radomnforest and knn3 are not working at least my labtop.

Thus, prediction based on models was chosen for the prediction.

2.0. creating data

A file "http://files.grouplens.org/datasets/movielens/ml-10m.zip" is downloaded. Three dataset (movielens, edx, and validation) have been created based on the file. movielens contains all the data and edx and validation are for training and final hold-out test sets, respectively.

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

library(tidyverse)
library(caret)
library(data.table)
```

```
# 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")

# if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                           title = as.character(title),
#                                           genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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") # if using R 3.5 or earlier, use `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)
edx <- rbind(edx, removed)

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

2.1. Data Analysis

Dimension of movielens

```
# dimension of data
dim(movielens)
```

```
## [1] 10000054        6
```

movielens consists of *10000054* rows and *6* columns. Thus, if every ratings are unique, *~6 10^7* ratings need to exist.

However, the distribution of rating shows

```
# rating distribution
movielens %>% group_by(rating) %>% summarize(n = n())
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   94988
##  2    1    384180
##  3    1.5  118278
##  4    2    790306
##  5    2.5  370178
##  6    3   2356676
##  7    3.5  879764
##  8    4   2875850
##  9    4.5  585022
## 10    5   1544812
```

the number of ratings is *~1 10^7*.

Thus, movielens has NA ratings.

This is also evident by considering unique user and movie.

```
# finding how mamy unique userId and movieId
movielens %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

Unique users are 69878 and unique movies are 10677. Thus, *~7 10^8* ratings is possible.

The list of ratings on the top 5 movies from userId 10 to 15 shows 6 NAs. Thus, the ratings are sparse.

```
library(dplyr)
library(knitr)
```

```
# top 5 numbers of movie
keep <- movielens %>%
  count(movieId) %>%
  top_n(5) %>%
  pull(movieId)
```

```
## Selecting by n
```

```
# select userId 10-15 and show the rating
tab <- movielens %>%
  filter(userId %in% c(10:15)) %>%
  filter(movieId %in% keep) %>%
  select(userId, title, rating) %>%
```

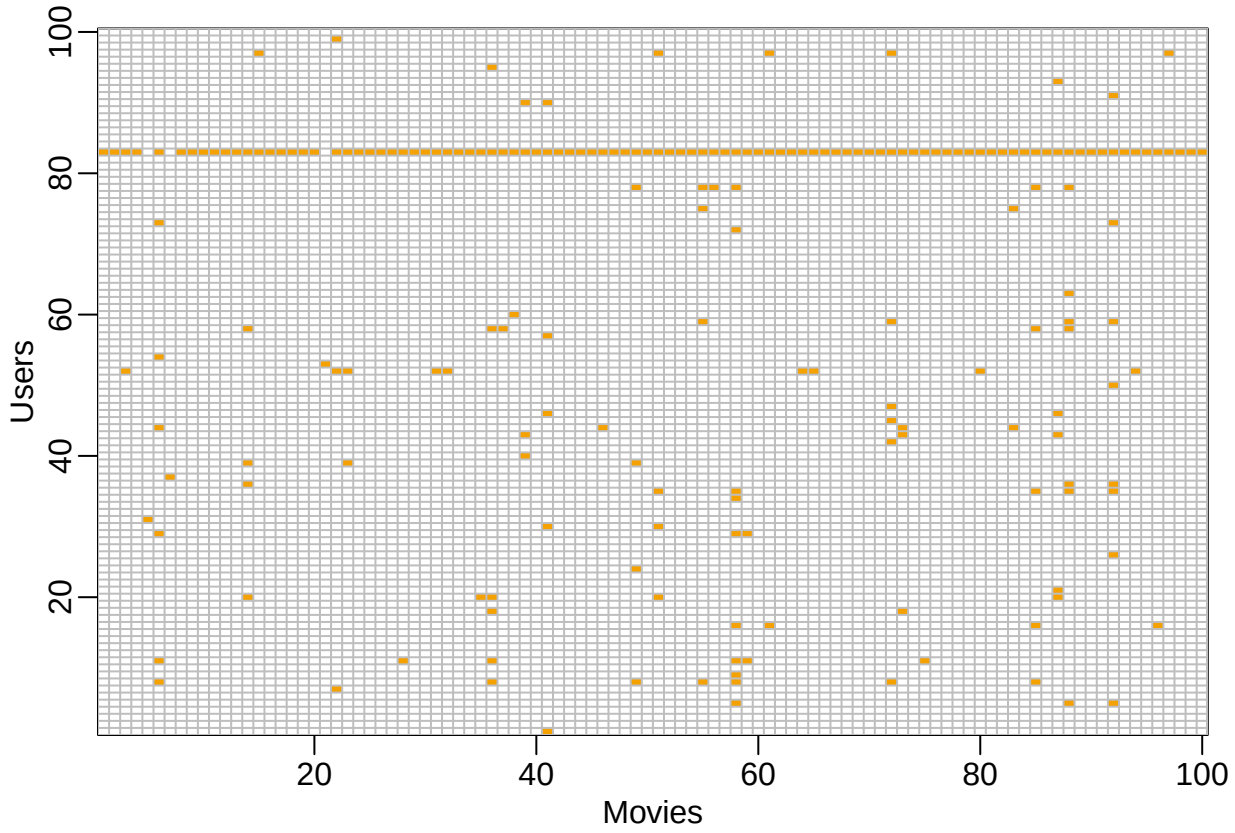
```
  spread(title, rating)

tab %>% kable()
```

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Silence of the Lambs, The (1991)
10	3	NA	2	3
11	NA	4	3	NA
13	NA	NA	4	NA

This can be illustrated with a hit-map of movies and users.

The hit-map of 100 users and 100 movies.

```
# showing sparseness of data using heat map of movieId and userId
# select unique 100 users
users <- sample(unique(movielens$userId), 100)
rafalib::mypar()

movielens %>% filter(userId %in% users) %>% # select only 100 unique users
  select(userId, movieId, rating) %>% # get userId, movieId, and rating
  mutate(rating = 1) %>% # for heat map, set rating to 1
  spread(movieId, rating) %>% # make data as row with userId and column with movieId with value of rati
  select(sample(ncol(.), 100)) %>% # choose 100 movieId
  as.matrix() %>% # make as matrix
  t(.) %>% # transpose the matrix. So, row is movieId, column is userId
  image(1:100, 1:100,. , xlab="Movies", ylab="Users") # heat map
# create grid for easy look
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```

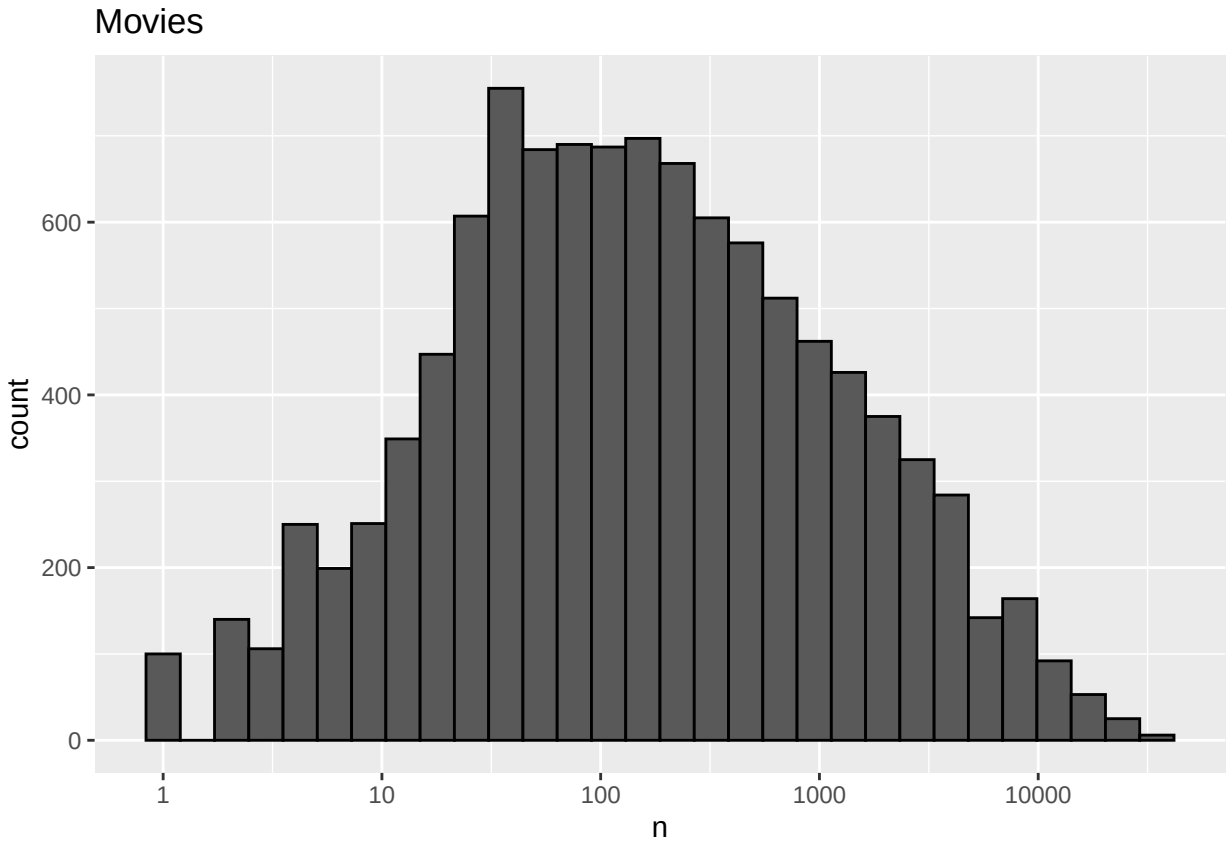


Interestingly, user 83 provides a lot of ratings compared with other users. This may have effects on a recommending system by making a weight on the rating based only a few users providing many ratings. This effect can be minimized with user effects described in 2.2.2.2 user effect.

Because rating depends on movie and user, one can model based on movie and user. In order to do that, one needs to make sure the distribution of movie and user in the data set are normal distribution.

The distribution of movie

```
# histogram of how the rate of the movie is provided.
movielens %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



Based on the figure, the distribution is a normal distribution,

The distribution of user

```
# histogram of how users provide rating
movielens %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```

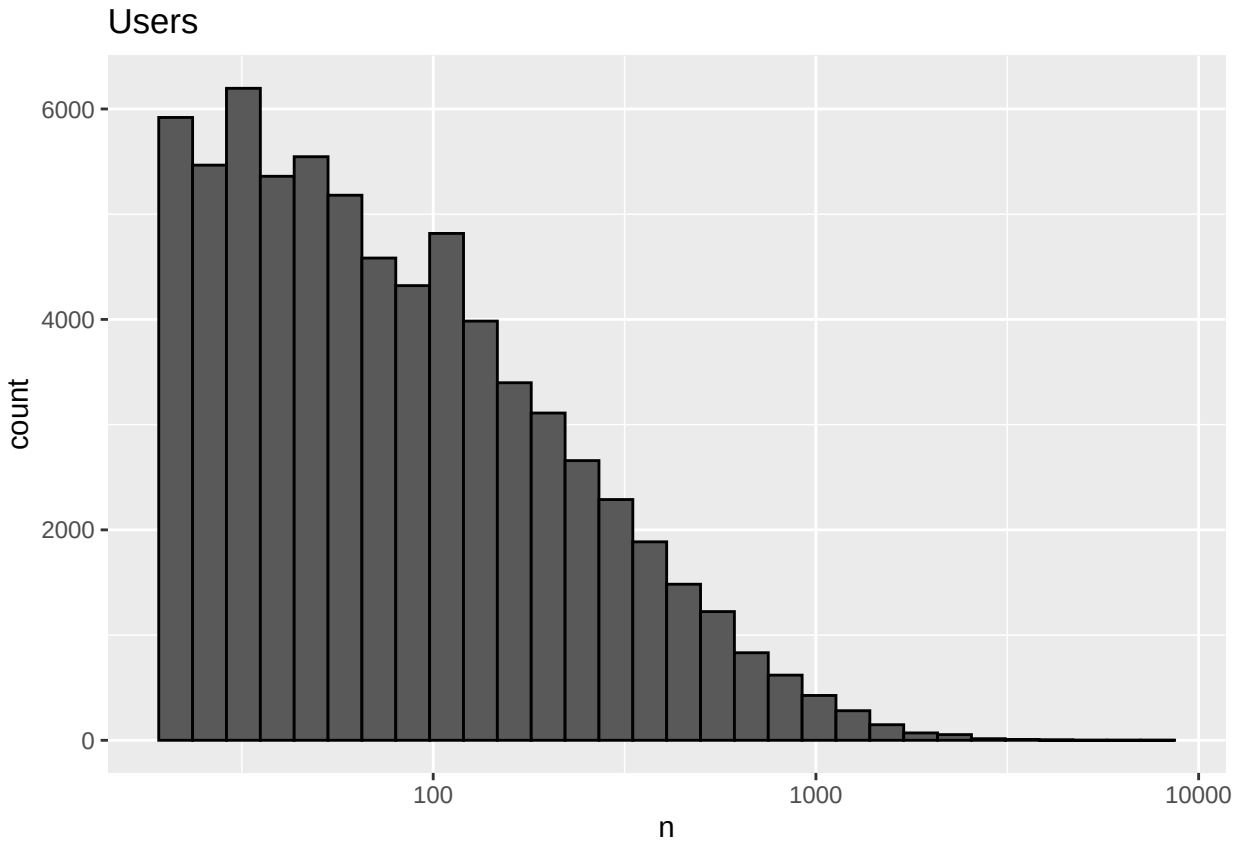


Figure looks like a half of normal distribution. Since the rating only occur positive, negative side (how much user won't rate the movie) does't present. Thus, user distribution is also a normal distribution.

rating dependence on time

```
# rating period calculation. lubridate origin is 1970-01-01.
library(lubridate)
```

```
# first date, last day, and duration of rating
tibble(first_date = date(as_datetime(min(movielens$timestamp),
                                     origin="1970-01-01")),
       last_date = date(as_datetime(max(movielens$timestamp),
                                    origin="1970-01-01"))) %>%
  mutate(duration = duration(max(movielens$timestamp)-min(movielens$timestamp)))
```

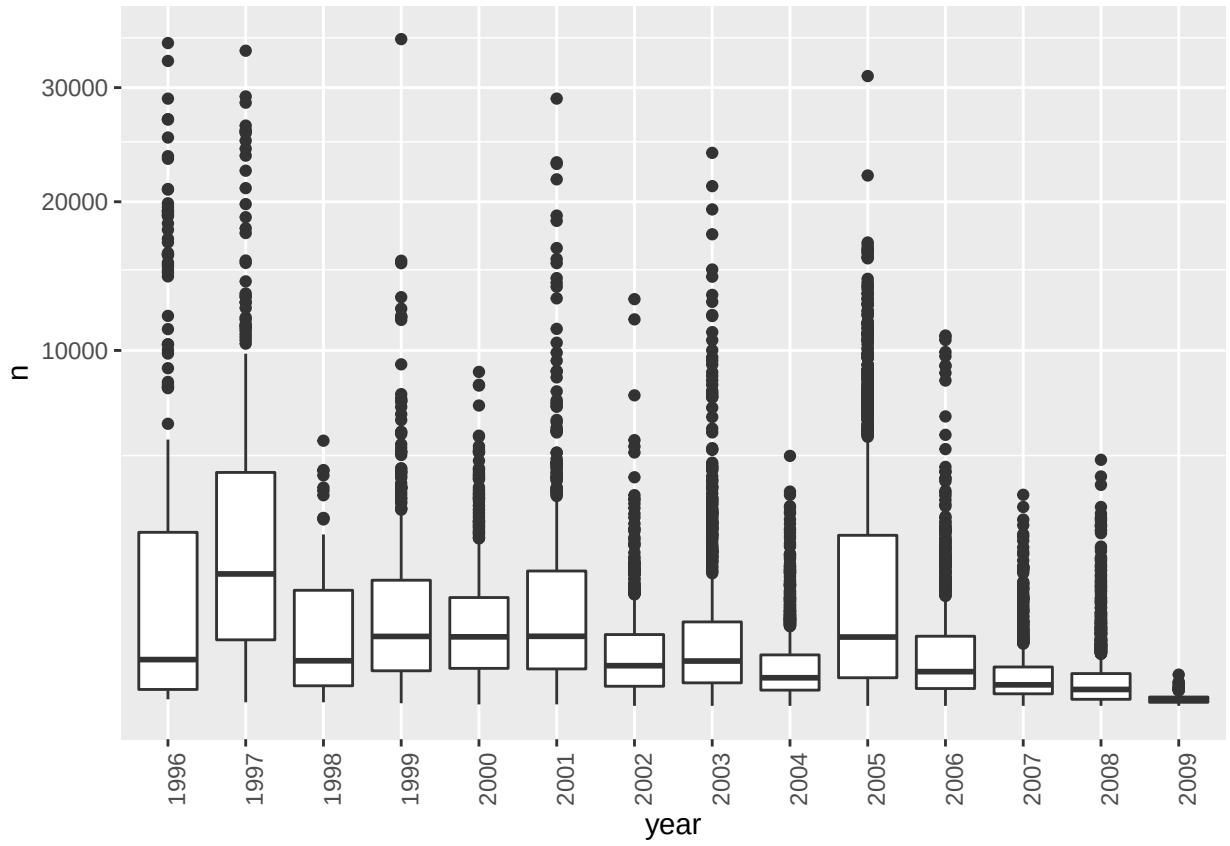
```
## # A tibble: 1 x 3
##   first_date last_date  duration
##   <date>     <date>     <Duration>
## 1 1995-01-09 2009-01-05 441479727s (~13.99 years)
```

The data was collected between 1995-01-09 2009-01-05 and period of ~14 years.

It is possible thqt the longer available, the more rating.

```
## how many movies came out in various years. lubridate origin is 1970-01-01.
movielens %>%
```

```
  mutate(year = year(as_datetime(timestamp, origin="1970-01-01"))) %>%
  group_by(movieId) %>%
  summarize(n = n(), year = as.character(first(year))) %>%
  qplot(year, n, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

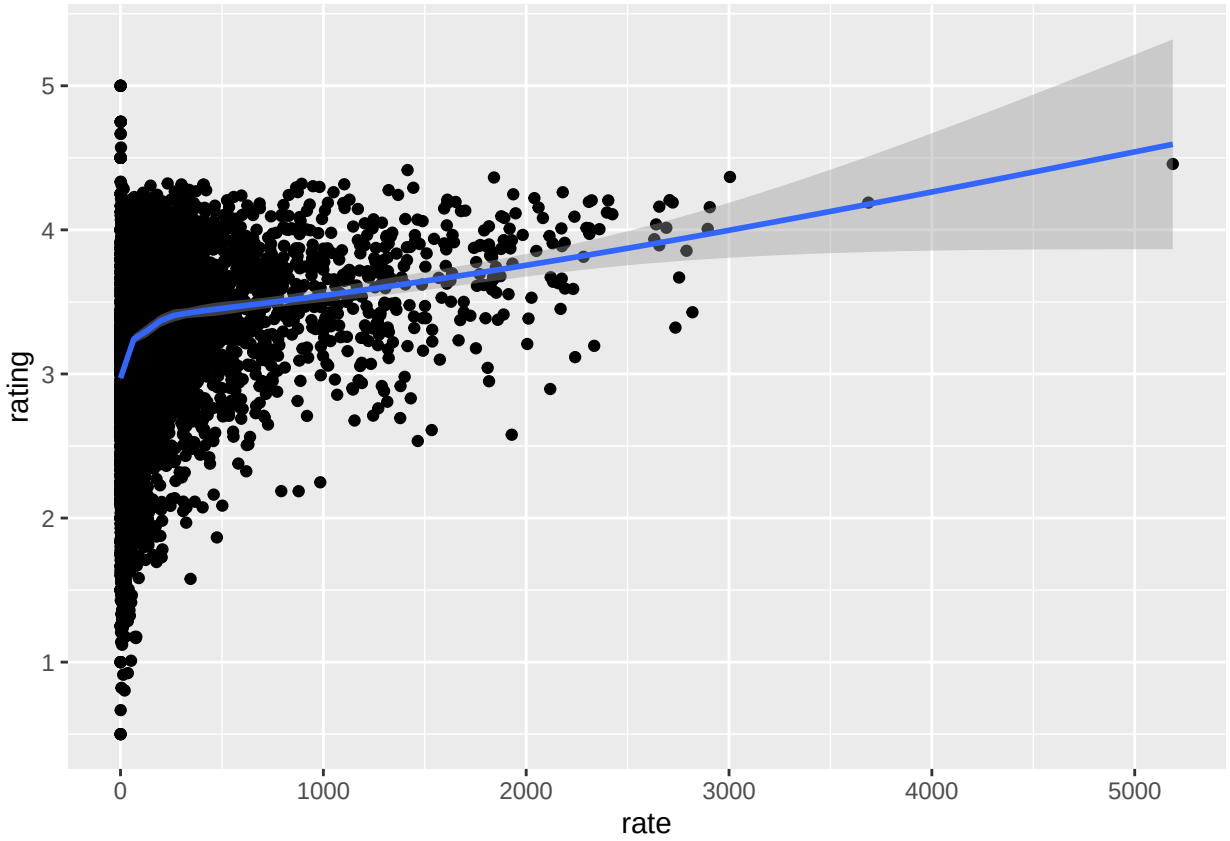


Indeed, the longer availability leads more ratings. To include this fact, dependency between frequency of rating on movie (The longer time, the more ratings) and rating.

** Relationship between rate of rating on movie and rating**

```
# relationship between rate of rating on movie and rating
library(gam)
```

```
movielens %>%
  mutate(year = year(as_datetime(timestamp, origin="1970-01-01"))) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = 2011 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  ggplot(aes(rate, rating)) +
  geom_point() +
  geom_smooth()
```

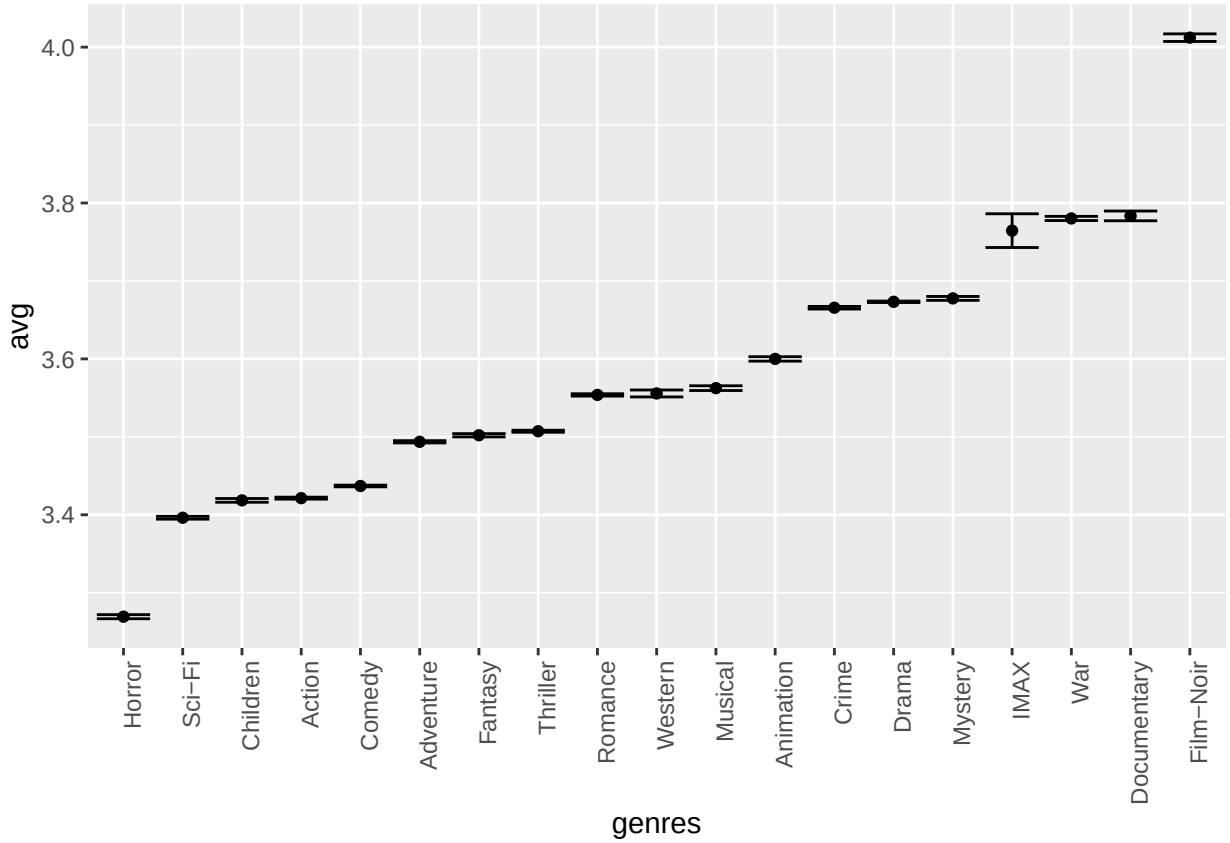


Based on this figure, model acounting this needs to be $f(d_{u,i})$ with f a smooth function of $d_{u,i}$ and $d_{u,i}$ the day of user u's rating of movie i.

The remaining object in movielens is possibly to related with rating is genres.

Relationship between genres and rating

```
# how the rating is related with genres
# considering multiple genres in a movie
movielens %>% separate_rows(genres, sep = "\\|") %>% # genres could be multiple. First separate
  group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  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))
```



Based on the figure, the contribution of genres on rating is linear (slope of β_k depending on k genres). Thus, the effect can be modeled as $\sum_{k=1}^{K} x_{u,i}^k \beta_k$ with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k, $g_{u,i}$ is the genre for user u's rating of movie i, and K is the total genres.

2.2. Models

2.2.1. Navie estimate A simplest estimation is that the average value (μ) of ratings would predict the rating ($Y_{u,i} with user(\text{u}) and movie(\text{i\$})$.

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

where $\epsilon_{u,i}$ are erros.

2.2.2. Movie effect From the experience, one knows that movie affects on rating. Simply good movie gets better rating than bad one. This effect (b_i) can be added to the previous model (2.2.1. Naive estimate)

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

2.2.3. User effect From the experience, one knows that user affects on rating. Simply user generous for rating gets better rating than cracky one. This effect (b_u) can be added to the previous model (2.2.2. Movie effect)

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

2.2.4. year effect From the data analysis, there is a relationship between time and rating, $d_{u,i}$ as the day of user u's rating of movie i. This can be added to the previous models.

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \epsilon_{u,i}$$

with f a smooth function of $d_{u,i}$

2.2.5. genres effect From the data analysis, there is a relationship between genres and rating. Thus, the effect can be added to the prevous models.

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \sum_{k=1}^{K} x_{u,i}^k \beta_k + \epsilon_{u,i}$$

with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k

2.3. Regularization For the movie effect case, lower number of rating on movie would be over-valued compared to large number of rating. It is also true for the user effect. Thus, the regularization commensates difference between small number of rating and large number of rating on movie and user effects.

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

where

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

and

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

2.4. Matrix Factorizing If one describes a model with movie and user effect,

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

one can describe residual ($r_{u,i}$) as

$$r_{u,i} = y_{u,i} - b_i - b_u$$

This can be expressed as

$$r_{u,i} \sim p_u q_i$$

This is the matrix factorizing. Marix, $r_{u,i}$, is decomposed into p_u and q_i.

Thus,

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

3. Results

3.1. Cross Validation

Using edx data set, data is divided into train (90%) and test (10%) sets for the cross-validation follwed by k-fold cross-validation

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`

# create train index p = 0.9
train_index <- createDataPartition(y = edx$rating, times = 1,
                                   p = 0.9, list = FALSE)

train <- edx[train_index,]
temp <- edx[-train_index,]

# Make sure userId and movieId in validation set are also in edx 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)

# clean up the workspace
rm(train_index, temp, removed)
```

Benchmark RMSE function

Root Mean Square Error is defined as follows:

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

where $\hat{y_{u,i}}$ and $y_{u,i}$ are predicted and actul values. This RMSE is used as the Benchmark.

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

3.2.1 Models

3.2.1.1 Naive estimate using the following Naive estimate

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

RMSE is calculated.

```
# average (mu) of rating in train set
mu <- mean(train$rating)

# prediction; rating is expected to be average for all.
predictions <- rep(mu, nrow(test))

# rmse from average
naive_rmse <- RMSE(predictions, test$rating)

# storing the rmse to rmse_results
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

```
## Warning: ‘data_frame()‘ was deprecated in tibble 1.1.0.
## Please use ‘tibble()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was generated.
```

```
# print table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.059802

3.2.1.2 Movie effect using the following model

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

RMSE is calculated.

```
b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# prediction
predictions <- test %>%
  left_join(b_i, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

# rmse calculation
model_bi_rmse <- RMSE(predictions, test$rating)

# storing the rmse to rmse_results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect",
                                     RMSE = model_bi_rmse ))
# print table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798

3.2.1.3. User effect Using the following model

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

RMSE is calculated.

rational: The taste of user should be related to rating

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

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

# rmse calculation
model_bi_bu_rmse <- RMSE(predictions, test$rating)

# storing the rmse to rmse_results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_bi_bu_rmse ))
# print table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798
Movie + User Effects Model	0.8649803

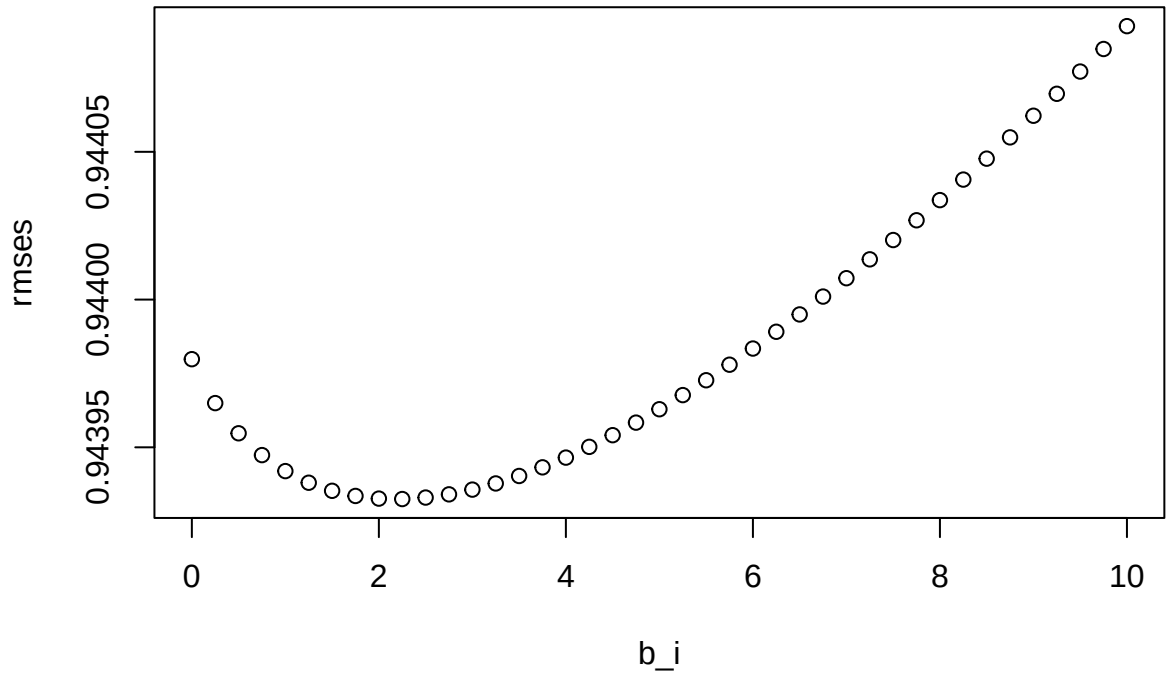
3.2.2. Regularization using the following model,

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

λ is optimzed.

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

rmses <- sapply(lambdas, function(l){
  b_i <- train %>%
```

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

  predictions <- test %>%
    left_join(b_i, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predictions, test$rating))
})

plot(lambdas,rmses, xlab = "b_i")
```



based on the plot, λ that produces the lowest RMSE is chosen.

```
# get lambda for the lowest rmse
lambdas[which.min(rmses)]
```

```
## [1] 2.25
```

```
model_reg_bi_rmse <- min(rmses)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regular Movie Effects Model",
                                     RMSE = model_reg_bi_rmse ))
# print table for RMSE
rmse_results %>% kable()
```

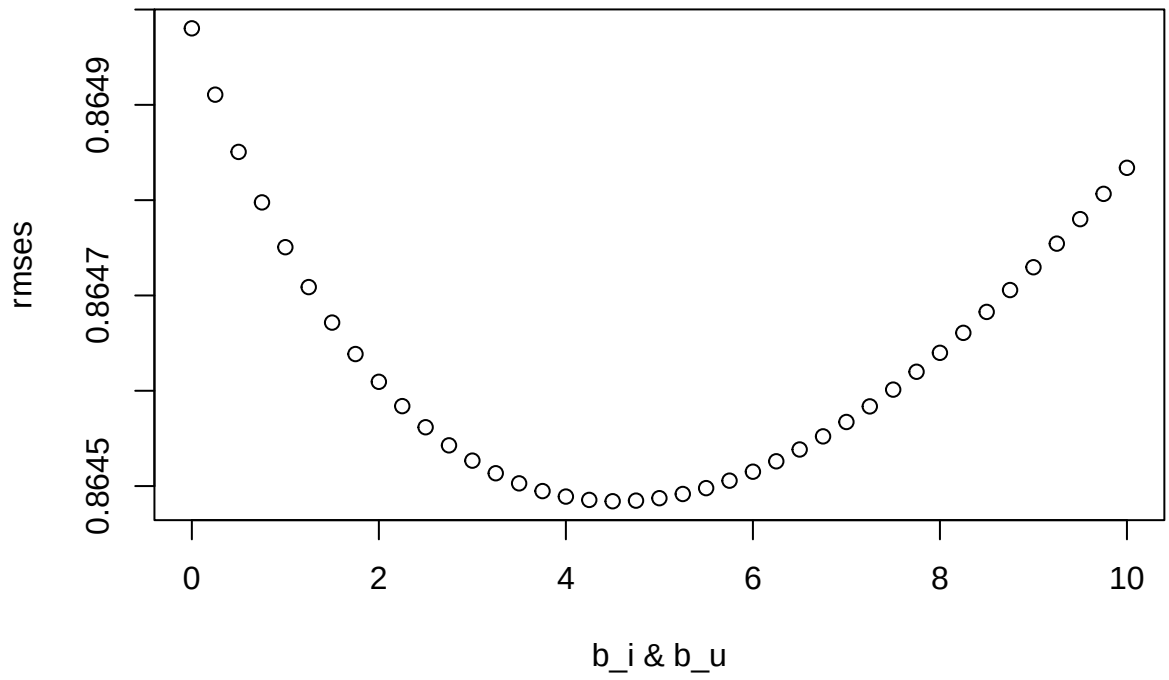
method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798
Movie + User Effects Model	0.8649803
Regular Movie Effects Model	0.9439325

using the following model

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

λ is optimzed.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predictions <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predictions, test$rating))
})

plot(lambdas,rmses, xlab = "b_i & b_u")
```



based on optimization. RMSE is obtained.

```
# set lambda for the lowest rmse
lambdas[which.min(rmses)]
```

```
## [1] 4.5
```

```
model_reg_bi_bu_rmse <- min(rmses)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie User Effect ",
                                     RMSE = model_reg_bi_bu_rmse ))

# print table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798
Movie + User Effects Model	0.8649803
Regular Movie Effects Model	0.9439325
Regularized Movie User Effect	0.8644841

3.2.3. Matrix Factorizing

recosystem This library is doing non-negative matrix factorizing.

Model is as follows:

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

RMSE is calculated.

```
library(recosystem)
set.seed(1, sample.kind = "Rounding") # This is a randomized algorithm

##########################################################################
# recosystem read only three columns.
# user_index needs to be user index, integer. userId for this data
# item_index needs to be item index, integer. movieId
# rating needs to be numeric. rating
# using data_memory function to include data
train_reco <- data_memory(user_index = train$userId,
                          item_index = train$movieId,
                          rating = train$rating)

test_reco <- data_memory(user_index = test$userId,
                         item_index = test$movieId,
                         rating = test$rating)

# Reco() function creates data files
rec <- Reco()

set.seed(1, sample.kind = "Rounding") # This is a randomized algorithm

# Actual calculation
set.seed(1, sample.kind = "Rounding") # This is a randomized algorithm
rec$train(train_reco, opts = c(nthread  = 4))
```

```
## iter      tr_rmse          obj
##    0       0.9623   1.3279e+07
##    1       0.8804   1.2025e+07
##    2       0.8573   1.1785e+07
##    3       0.8465   1.1651e+07
##    4       0.8427   1.1604e+07
##    5       0.8405   1.1581e+07
##    6       0.8388   1.1567e+07
##    7       0.8370   1.1551e+07
##    8       0.8349   1.1539e+07
##    9       0.8321   1.1519e+07
##   10       0.8294   1.1505e+07
##   11       0.8270   1.1485e+07
##   12       0.8253   1.1470e+07
##   13       0.8240   1.1462e+07
##   14       0.8230   1.1455e+07
##   15       0.8222   1.1450e+07
##   16       0.8217   1.1444e+07
##   17       0.8211   1.1439e+07
```

```
##   18       0.8207   1.1435e+07
##   19       0.8203   1.1432e+07
```

```
# predition from recosystem. For the return, out_memory() should be used.
predictions <- rec$predict(test_reco, out_memory())

# calculate RMSE
model_mat_fac_rmse <- RMSE(predictions, test$rating)

# stroring RMSE
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix Factorized",
                                     RMSE = model_mat_fac_rmse ))

# print table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798
Movie + User Effects Model	0.8649803
Regular Movie Effects Model	0.9439325
Regularized Movie User Effect	0.8644841
Matrix Factorized	0.8329753

```
## Warning in rm(train, test, b_i, b_u, predictions, opts, lambda, lambdas, :
## object 'opts' not found
```

```
## Warning in rm(train, test, b_i, b_u, predictions, opts, lambda, lambdas, :
## object 'lambda' not found
```

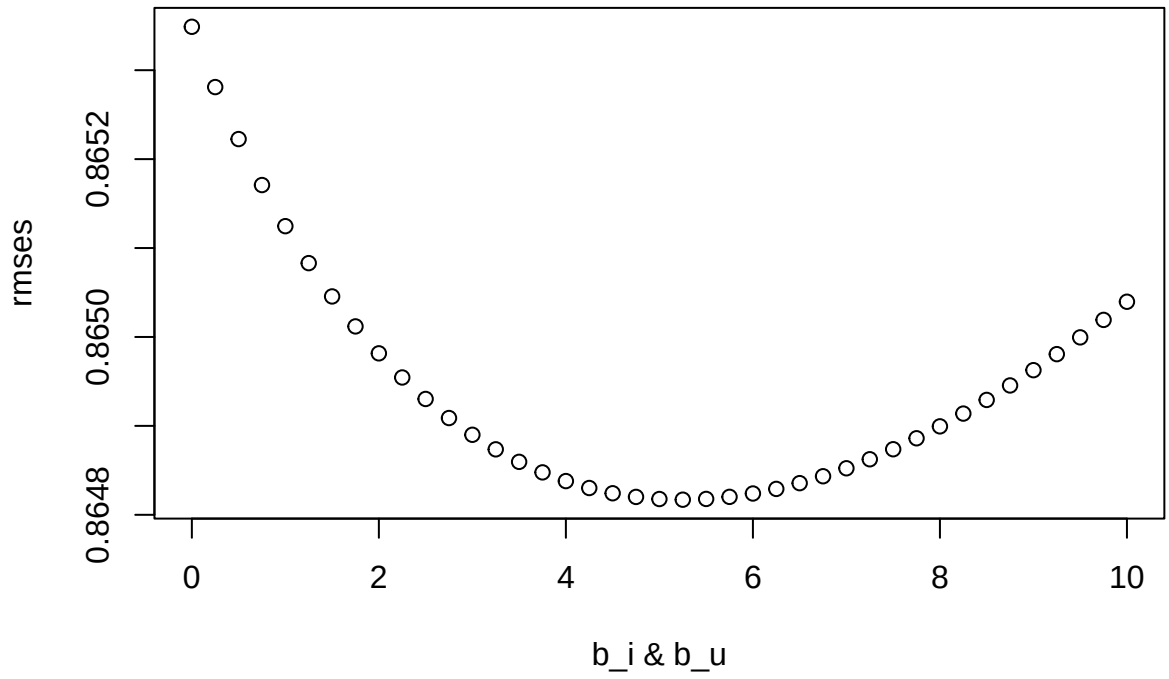
3.2. Final Hold-out Test

using edx and validation sets

3.2.1. Regularization with Movie and User effects b_i regularization (1) and b_u = sum(rating - b_i - mu)/(n()+l) getting lambda for the lowest rmse for regularized bi and bu

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

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
  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 - b_i - mu)/(n()+l))

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

  return(RMSE(predictions, validation$rating))
})

plot(lambdas,rmses, xlab = "b_i & b_u")
```



based on the optimization

```
# set lambda for the lowest rmse
lambdas[which.min(rmses)]
```

```
## [1] 5.25
```

```
model_reg_bi_bu_rmse <- min(rmses)

rmse_results <- bind_rows(rmse_results,
```

```
                          data_frame(method="Final Regularized Movie User Effect ",
                                     RMSE = model_reg_bi_bu_rmse ))

# print table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798
Movie + User Effects Model	0.8649803
Regular Movie Effects Model	0.9439325
Regularized Movie User Effect	0.8644841
Matrix Factorized	0.8329753
Final Regularized Movie User Effect	0.8648170

RMSE is calculated

3.2.2. Matrix Factorizing recosystem this library is doing non-negative matrix factorizing model: Y = P*Q

```
library(recosystem)
set.seed(1, sample.kind = "Rounding") # This is a randomized algorithm

# data_memory function to store in recosystem
edx_reco <- data_memory(user_index = edx$userId,
                        item_index = edx$movieId,
                        rating = edx$rating)

validation_reco <- data_memory(user_index = validation$userId,
                               item_index = validation$movieId,
                               rating = validation$rating)

# Reco() function creates data files
rec <- Reco()

# Actual calculation
set.seed(1, sample.kind = "Rounding") # This is a randomized algorithm
rec$train(edx_reco, opts = c(nthread  = 4))
```

```
## iter      tr_rmse          obj
##    0       0.9546   1.4604e+07
##    1       0.8811   1.3316e+07
##    2       0.8604   1.3142e+07
##    3       0.8446   1.2953e+07
##    4       0.8368   1.2875e+07
##    5       0.8318   1.2827e+07
##    6       0.8287   1.2794e+07
##    7       0.8266   1.2775e+07
##    8       0.8253   1.2759e+07
##    9       0.8243   1.2751e+07
```

```
##   10       0.8235   1.2742e+07
##   11       0.8228   1.2738e+07
##   12       0.8223   1.2729e+07
##   13       0.8219   1.2728e+07
##   14       0.8216   1.2723e+07
##   15       0.8213   1.2720e+07
##   16       0.8210   1.2712e+07
##   17       0.8207   1.2714e+07
##   18       0.8205   1.2711e+07
##   19       0.8203   1.2705e+07
```

```
# predition from recosystem. For the return, out_memory() should be used.
predictions <- rec$predict(validation_reco, out_memory())

# store predictions for the movie recommendation
MF_pred <- predictions

# calculate RMSE
model_mat_fac_rmse <- RMSE(predictions, validation$rating)

# stroring RMSE
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Final Matrix Factorized",
                                     RMSE = model_mat_fac_rmse ))

# print table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798
Movie + User Effects Model	0.8649803
Regular Movie Effects Model	0.9439325
Regularized Movie User Effect	0.8644841
Matrix Factorized	0.8329753
Final Regularized Movie User Effect	0.8648170
Final Matrix Factorized	0.8325267

4. Conclusion

Summary Table

```
# Summary table for RMSE
rmse_results %>% kable()
```

method	RMSE
Just the average	1.0598022
Movie Effect	0.9439798
Movie + User Effects Model	0.8649803

method	RMSE
Regular Movie Effects Model	0.9439325
Regularized Movie User Effect	0.8644841
Matrix Factorized	0.8329753
Final Regularized Movie User Effect	0.8648170
Final Matrix Factorized	0.8325267

RMSE using matrix factorization (recosystem) is 0.8325373 below the target 0.8649.

Limitation & Furture Works

All the calculation is based on the emperical models with movie effect, user effect, and matrix factorization. Other effects such as time and genres effect can be included in the analysis. But, the computational issue using a small labtop is hindering. Furthermore, using other methods such as randomnforest and knn, which was not used due to the computational issues, the prediction might be better. Thus, these possibilities might be pursued to imporve the recommendation system.