Capstone Movielens Project : Predict movie rating

Swalini Mary

Introduction

A recommendation system is used to predict the "rating" or "preference" a user would give to an item. Products for which a high rating is predicted for a given user are then recommended to that user. Amazon, Google, Goodreads, etc. use recommendation systems to make predictions and recommend products to users. Netflix uses a recommendation system to predict user ratings for a specific movie.

In this Capstone project, we will predict user ratings for movies using different models. We will follow the methods described in Data Science: Machine Learning Course and finally include Matrix Factorization model, which was just introduced in this course. The data set is very long, so we use models that allow the data processing using the available RAM in the computer.

Evaluation metric:

Evaluation metric used in this project is RMSE (Root Mean Squared Error). It measures the error in the predicted ratings:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(predicted_i - actual_i)^2}$$

Here,

Predicted is the rating predicted by the model and Actual is the original rating.

N is the number of user/movie combinations and the sum occurring over all these combinations.

If a user has given a rating of 5 to a movie and we predicted the rating as 4, then RMSE is 1. Therefore, lesser the RMSE value, better the recommendations.

We write a function called RMSE() that takes two numeric vectors (one is the true movie ratings and the other is predicted movie ratings) as input, and returns the root mean squared error (RMSE) between the two as output.

```
#-----------------------------------------------------
## RMSE: compute root mean square error (RMSE)
#-----------------------------------------------------

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

#-----------------------------------------------------
```

Data:

We will work on the MovieLens dataset and try to build models to predict movie ratings. This data has been collected by the GroupLens Research Project at the University of Minnesota. The dataset can be downloaded here. This dataset consists of 69878 unique users and 10677 unique movies.

We partition Movielens data into edx and validation sets using the code provided by edx.

```
#----------------------------------------------------
## Create edx set, validation set
#----------------------------------------------------

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse

## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang

## -- Attaching packages ------------------------------------------ tidyverse 1.2.1 --

## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts --------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
```

```r
# 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")
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")

# Validation set will be 10% of MovieLens data

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
# if using R 3.5 or earlier, use `set.seed(1)` instead

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")
```

```r
edx <- rbind(edx, removed)

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

#------------------------------------------------------
```

Next, we partition edx data set into training and test data sets, so that test set can be used to evaluate results while developing models. Set.seed() command is used to reproduce same results every time. createDataPartition command from caret package is used to partition data into train and test sets from edx data set. Train set contains 90% of edx data while test set has 10% of edx data which are randomly selected. Train and test set will be used for further analyses and model development. Also, we ensure train set data contains userId and movieId which are present in test data set. We remove variables which are not required to save space.

```r
#------------------------------------------------------
## Create train and test sets from edx data set
```

```r
#---------------------------------------------------------

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
# use 10% of edx data as test set

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train 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")
```

```r
train <- rbind(train, removed)

# Remove unused data to save space

rm(removed, temp, test_index)

#---------------------------------------------------------
```

Data Exploration:

By simply typing the name of the data set we can see this table has data in tidy format and there are thousands of rows.

```r
# Exploring train data set

train %>% as_tibble()
```

```
## # A tibble: 8,100,065 x 6
##    userId movieId rating timestamp title                genres
##     <int>   <dbl>  <dbl>     <int> <chr>                <chr>
##  1      1     122      5 838985046 Boomerang (1992)     Comedy|Romance
##  2      1     292      5 838983421 Outbreak (1995)      Action|Drama|Sci-Fi~
##  3      1     316      5 838983392 Stargate (1994)      Action|Adventure|Sc~
##  4      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  5      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  6      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  7      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  8      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
##  9      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## 10      1     377      5 838983834 Speed (1994)         Action|Romance|Thri~
```

```
## # ... with 8,100,055 more rows
```

We can further obtain the distinct number of users who rated the movies and the distinct number of movies which were rated. We can obtain the highest and lowest rating given. Each row represents the rating given by one user to one movie.

```r
train %>% summarize(
  n_users=n_distinct(userId),# unique users from train data set
  n_movies=n_distinct(movieId),# unique movies from train data set
  min_rating=min(rating),  # the lowest rating
  max_rating=max(rating) # the highest rating
)
```

```
##   n_users n_movies min_rating max_rating
## 1   69878    10677        0.5          5
```

We can observe that not all users have rated all movies. So we can think of these data as a very large matrix, with rows representing users and columns representing movies and many empty cells. Here we show the matrix for 5 movies and first 10 users. Spread function is used to display data in this format. Trying for entire train data set will crash R.

```r
# Display movies and users as a matrix

movie_matrix <- train %>%
  count(movieId) %>%
  top_n(5, n) %>%
  .$movieId

final_matrix <- train %>%
  filter(movieId%in%movie_matrix) %>%
  filter(userId %in% c(1:10)) %>%
  select(userId, title, rating) %>%
  mutate(title = str_remove(title, ", The"),
         title = str_remove(title, ":.*")) %>%
  spread(title, rating)

final_matrix %>% knitr::kable()
```

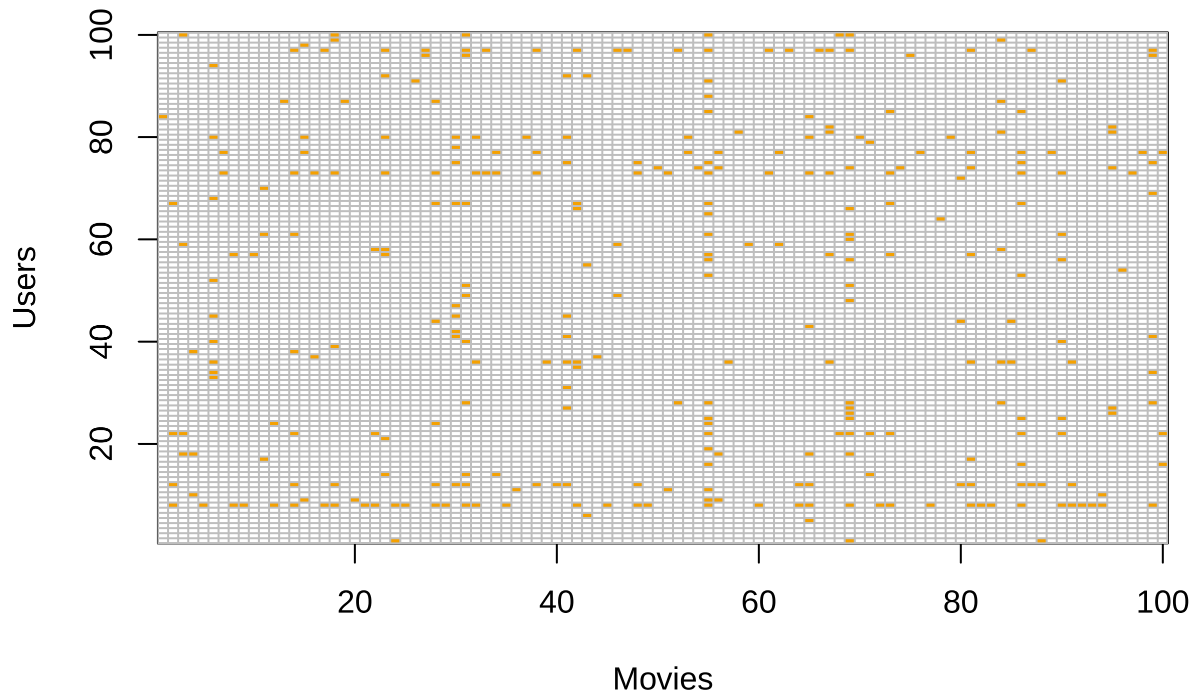
userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Silence of the Lambs (1991)
1	5	NA	NA	NA
4	NA	5	NA	NA
7	NA	NA	NA	3
8	NA	3	NA	4
10	3	NA	2	3

```r
rm(movie_matrix,final_matrix )
```

The objective of this recommendation system is to fill the NAs in the above table with the most likely rating the user would give the movie. To see how sparse the matrix is, here is the matrix for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating.

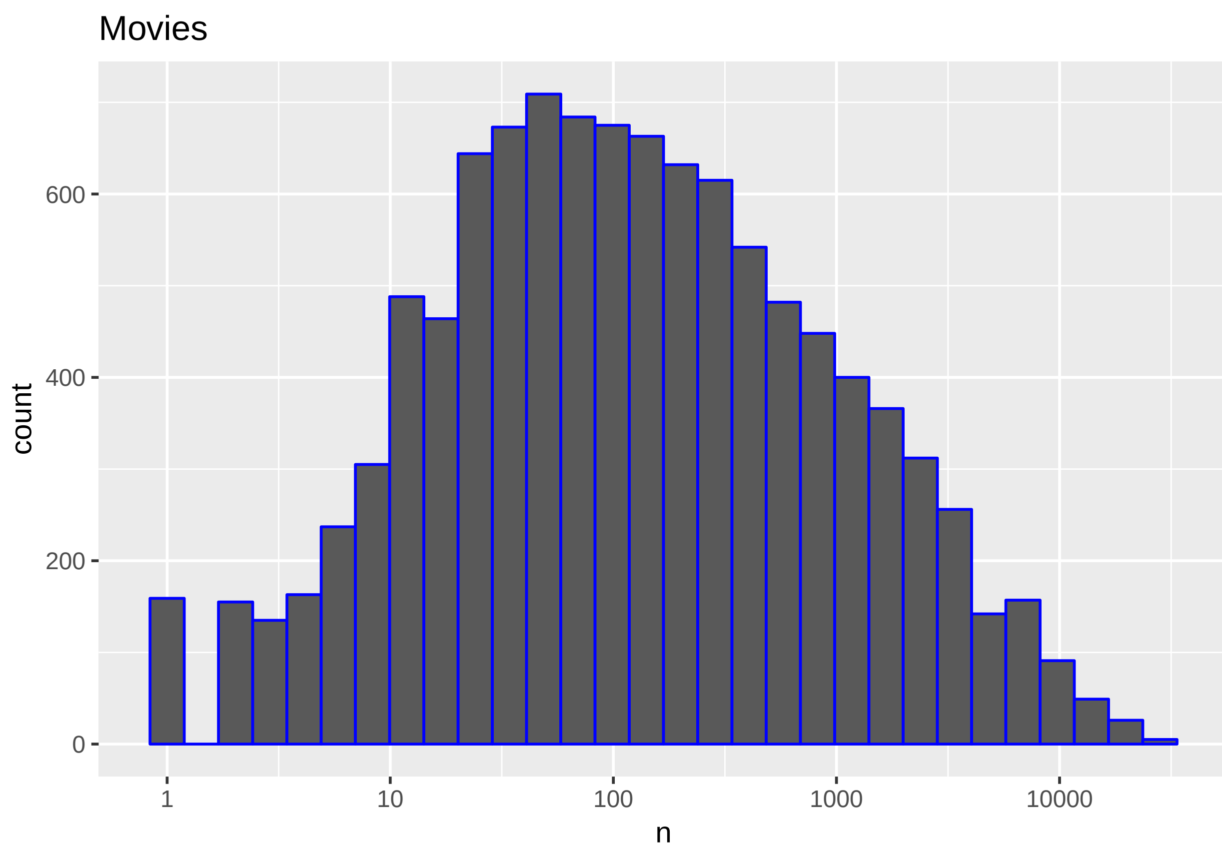
```r
# This matrix displays a random sample of 100 movies and 100 users with yellow
# indicating a user/movie combination for which we have a rating.

users <- sample(unique(train$userId), 100)
train %>% filter(userId %in% users) %>%
```

```
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```

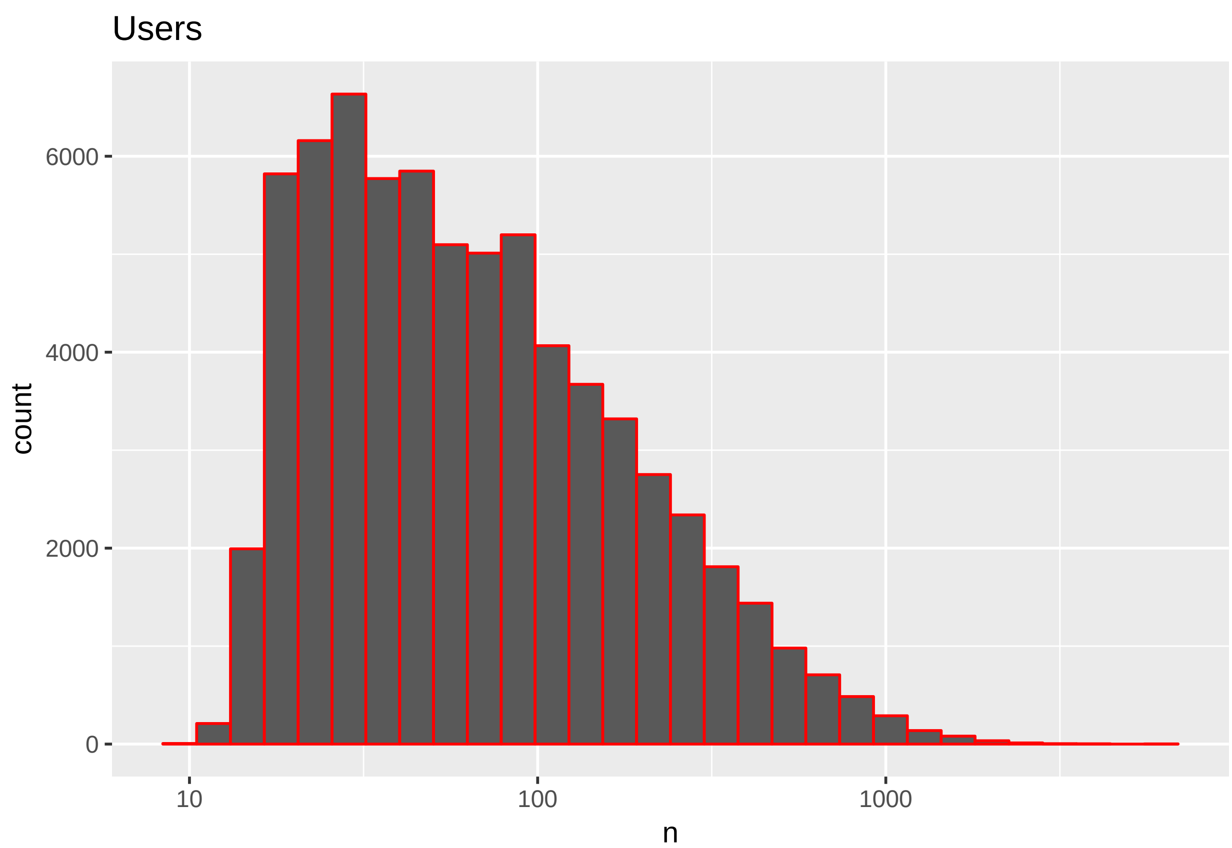


Below is the distribution of movies and users. We can observe that some movies are rated more compared to other movies. Similarly some users have rated more movies than other users.

```
# This plot displays rating count by movie

train %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue") +
  scale_x_log10() +
  ggtitle("Movies")
```




```r
# This plot displays rating count by users

train %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "red") +
  scale_x_log10() +
  ggtitle("Users")
```



Data Modelling:

We start by building the basic model which is assigning the average of all ratings to all movies irrespective of the users. The model can be represented as shown below:

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

where $\varepsilon_{u,i}$ is independent errors and μ the "true" rating for all movies. The average rating gives the least RMSE whereas any other number gives a higher RMSE.

```
# Calulate the average movie rating mu

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

```
## [1] 3.512456
```

```
# Compute RMSE for just the average

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

```
## [1] 1.060054
```

We will be comparing the results from different models. So we will tabulate the results in a table.

```r
# Create a results table to record RMSE for all models

rmse_results <- tibble(Method = "Just the average", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.060054

We can definitely do better.

Modeling Movie effect

During data exploration we observed that some movies are generally rated more than other movies. We can add a term b_i to the equation of the previous basic model to account for this variation in movie rating.

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

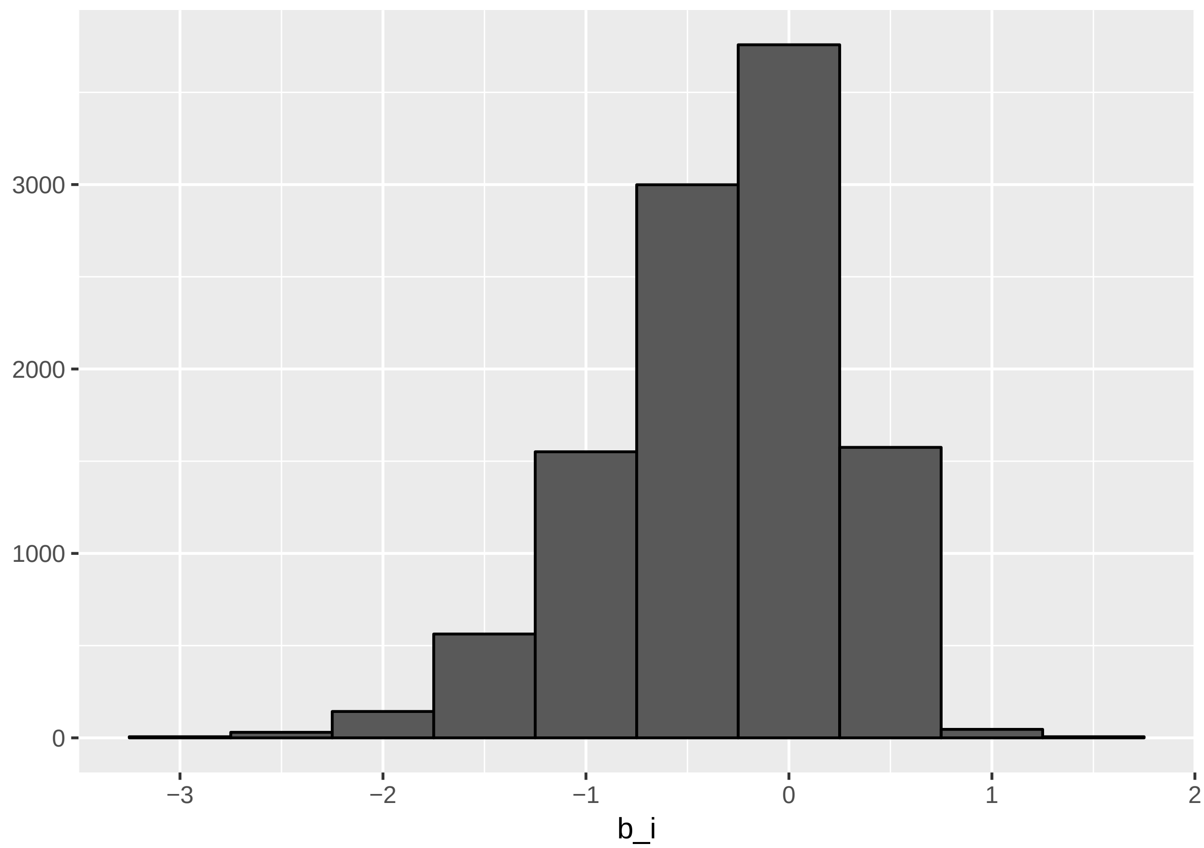
We can use least squares to estimate the b_i in the following way,

```r
# Fitting using least squares estimates will take a long time

fit <- lm(rating ~ as.factor(movieId), data = train)
```

but because there are thousands of b_i as each movie gets one, the lm() function will be very slow here if not impossible to run.

In this particular situation, we know that the least square estimate b_i is just the average of Y_{u,i} - mu for each movie i. So we can compute them this way:

```r
# We will use the fact that b_i = Y(u,i) - mu

mu <- mean(train$rating)
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



Let's evaluate the improvement in RMSE using this model:

```
# Calulate RMSE for model with movie effect

predicted_ratings <- mu + test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

movie_effect <- RMSE(test$rating, predicted_ratings)

# Add this result to the results table

rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Model with Movie Effect",
                                 RMSE = movie_effect ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600537
Model with Movie Effect	0.9429615

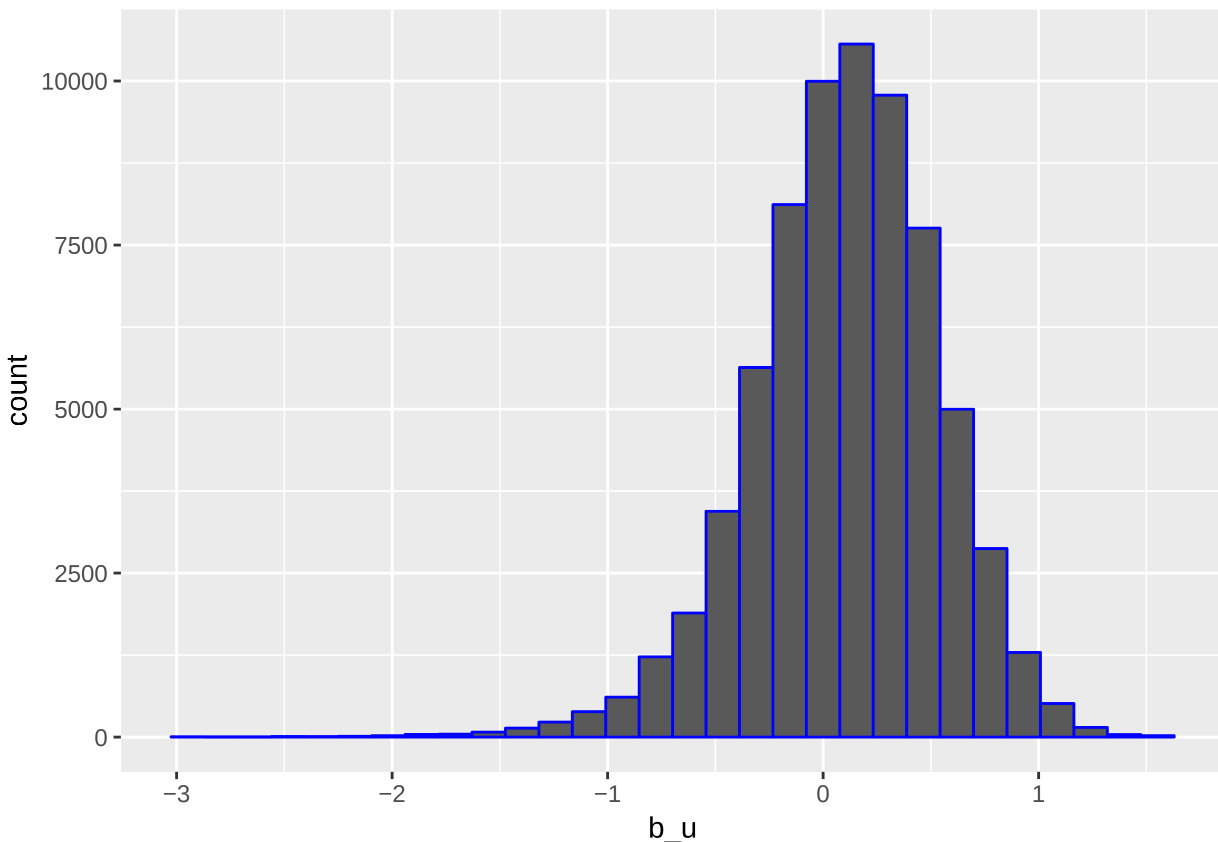
This is an improvement compared to the previous model. Maybe we can still do better.

Modeling movie and user effects

Similarly, during data exploration we observed that some users generally rate more movies than others. We can add a term b_u to the equation of the movie effect model to account for this variation in movie rating.

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

```r
# The plot below shows us there is variability across users

train %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "blue")
```



Fitting with least squares estimates again will be very slow, so we the below method as in movie effect model.

```r
# Fitting using least squares estimates will take a long time

fit <- lm(rating ~ as.factor(movieId) + as.factor(userId))
```

```r
# We will use the fact that b_u = Y(u,i) - mu - b_i

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

Let's evaluate the improvement in RMSE using this model:

```r
# Calulate RMSE for model with movie and user effect

predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

user_effect <- RMSE(predicted_ratings, test$rating)

# Add this result to the results table

rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Model with Movie + User Effects",
                                 RMSE = user_effect ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600537
Model with Movie Effect	0.9429615
Model with Movie + User Effects	0.8646843

This is a good improvement over the movie effect model. Lets check regularization technique.

Regularization

Regularization is used to penalize large estimates that are formed using small sample sizes.

These are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so we would rather be conservative when unsure.

Let's look at the top 10 worst and best movies based on b_i. First, let's create a database that connects movieId to movie title:

```r
# Create a database that connects movieId to movie title

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

Here are the 10 best movies according to our estimate:

```r
# Top 10 best movies based on b_i

movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(3:12) %>%
  knitr::kable()
```

title	b_i
Shadows of Forgotten Ancestors (1964)	1.487544
Fighting Elegy (Kenka erejii) (1966)	1.487544
Sun Alley (Sonnenallee) (1999)	1.487544

title	b_i
Blue Light, The (Das Blaue Licht) (1932)	1.487544
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544
Human Condition II, The (Ningen no joken II) (1959)	1.237544
Human Condition III, The (Ningen no joken III) (1961)	1.237544
Constantine's Sword (2007)	1.237544
More (1998)	1.154211

And here are the 10 worst:

```r
# Top 10 worse movies based on b_i

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

title	b_i
Besotted (2001)	-3.012456
Hi-Line, The (1999)	-3.012456
Accused (Anklaget) (2005)	-3.012456
Confessions of a Superhero (2007)	-3.012456
War of the Worlds 2: The Next Wave (2008)	-3.012456
SuperBabies: Baby Geniuses 2 (2004)	-2.767775
Disaster Movie (2008)	-2.745789
From Justin to Kelly (2003)	-2.638139
Hip Hop Witch, Da (2000)	-2.603365
Criminals (1996)	-2.512456

How often the best and worse are rated:

```r
# To find how often the best obscure movies are rated

train %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(3:12) %>%
  knitr::kable()
```

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

title	b_i	n
Shadows of Forgotten Ancestors (1964)	1.487544	1
Fighting Elegy (Kenka erejii) (1966)	1.487544	1
Sun Alley (Sonnenallee) (1999)	1.487544	1
Blue Light, The (Das Blaue Licht) (1932)	1.487544	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544	4
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544	2

title	b_i	n
Human Condition II, The (Ningen no joken II) (1959)	1.237544	4
Human Condition III, The (Ningen no joken III) (1961)	1.237544	4
Constantine's Sword (2007)	1.237544	2
More (1998)	1.154211	6

```
# To find how often the worse obscure movies are rated

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

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

title	b_i	n
Besotted (2001)	-3.012456	1
Hi-Line, The (1999)	-3.012456	1
Accused (Anklaget) (2005)	-3.012456	1
Confessions of a Superhero (2007)	-3.012456	1
War of the Worlds 2: The Next Wave (2008)	-3.012456	2
SuperBabies: Baby Geniuses 2 (2004)	-2.767775	47
Disaster Movie (2008)	-2.745789	30
From Justin to Kelly (2003)	-2.638139	183
Hip Hop Witch, Da (2000)	-2.603365	11
Criminals (1996)	-2.512456	1

The supposed "best" and "worst" movies were rated by very few users, in most cases just 1. These movies were mostly obscure ones. This is because with just a few users, we have more uncertainty. Therefore, larger estimates of b_i, negative or positive, are more likely.

Choosing the penalty terms

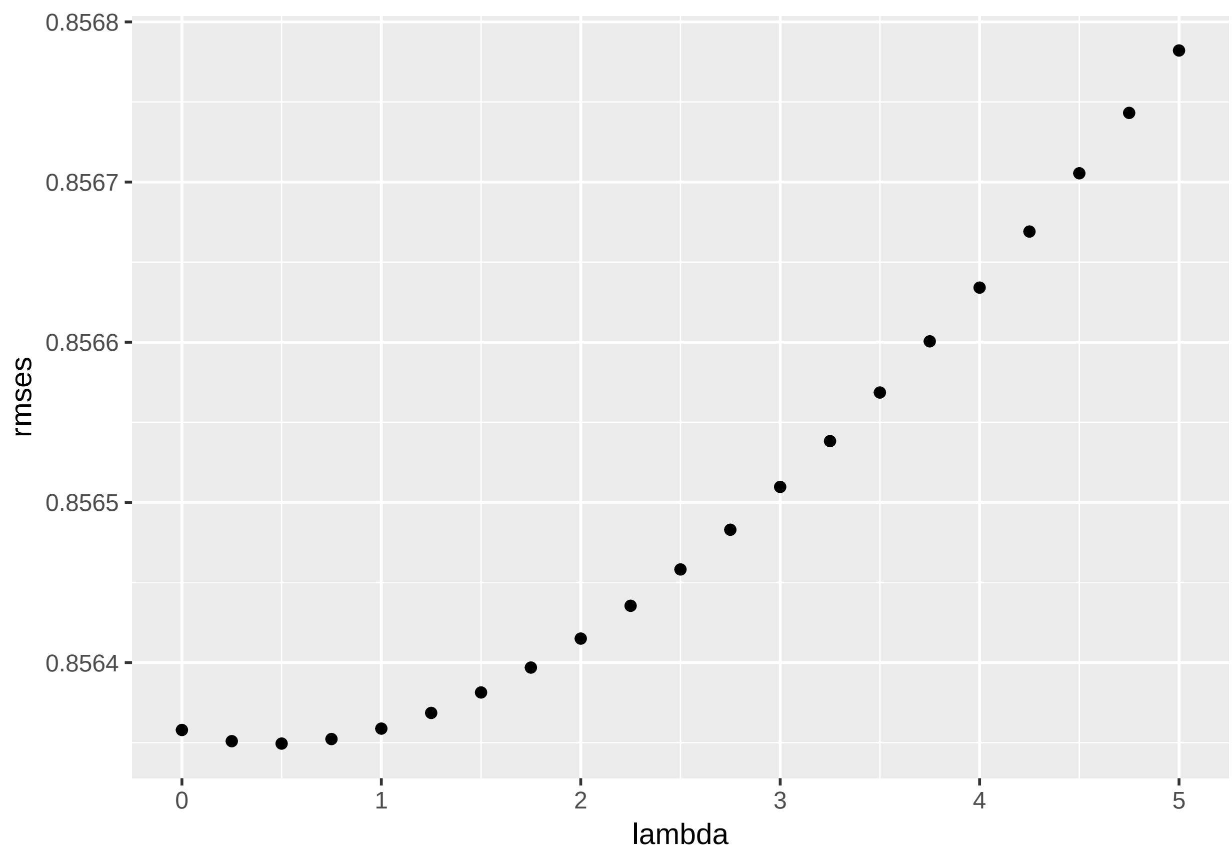
We use regularization for the estimate of both movie and user effects. We are minimizing:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

Here we use cross-validation to pick a λ:

```
# use cross-validation to pick the penalty term lambda:

lambda <- seq(0, 5, 0.25)

rmses <- sapply(lambda, function(l){
  mu <- mean(train$rating)

  b_i <- train %>%
```

```r
    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))

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

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

qplot(lambda, rmses)
```




```r
# pick lambda with minimun rmse

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

# print lambda
```

```
lambda
```

```
## [1] 0.5
```

We use the test set for the final assessment of this model

```
# compute movie effect with regularization on train set

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

# compute user effect with regularization on train set

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

# compute predicted values on test set

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

# create a results table with this and previous approaches

model_regularization <- RMSE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Model with Regularized Movie and User Effect",
                                 RMSE = model_regularization))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600537
Model with Movie Effect	0.9429615
Model with Movie + User Effects	0.8646843
Model with Regularized Movie and User Effect	0.8645518

In this case, regularization does not produce significant improvement in performance using RMSE as the metric.

Matrix factorization

A popular technique to solve recommender system problem is the matrix factorization method. The idea is to approximate the whole rating matrix R_{mXn} by the product of two matrices of lower dimensions, P_{kxm} and $Q_{k X n}, such that

$$R \approx P'Q$$

Let p_u be the u-th column of P, and q_v be the v-th column of Q, then the rating given by user u on item v would be predicted as $p'_u q_v$.

The process of solving the matrices P and Q is referred to as model training, and the selection of penalty parameters is called parameter tuning.

full description of package is available here.

Data Format

The data file for training set needs to be arranged in sparse matrix triplet form, i.e., each line in the file contains three numbers

user_index item_index rating User index and item index may start with either 0 or 1, and this can be specified by the index1 parameter in data_file() and data_memory(). For example, with index1 = FALSE, the training data file for the rating matrix in the beginning of this article may look like

0 0 2 0 1 3 1 1 4 1 2 3 2 0 3 2 1 2 . . .

Testing data file is similar to training data, but since the ratings in testing data are usually unknown, the rating entry in testing data file can be omitted, or can be replaced by any placeholder such as 0 or ?.

The testing data file for the same rating matrix would be

0 2 1 0 2 2 . . . Example data files are contained in the /dat (or /inst/dat, for source package) directory.

Usage of recosystem

The usage of recosystem is quite simple, mainly consisting of the following steps:

1. Create a model object (a Reference Class object in R) by calling Reco().
2. (Optionally) call the $tune() method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the $train() method. A number of parameters can be set inside the function, possibly coming from the result of $tune().
4. (Optionally) export the model via $output(), i.e. write the factorization matrices P and Q into files or return them as R objects.
5. Use the $predict() method to compute predicted values.

Following the steps mentioned above, the code for matrix factorization is as below:

```r
# Install/Load recosystem

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

```
## Loading required package: recosystem

## Warning: package 'recosystem' was built under R version 3.6.1
```

```r
# The data file for training and test set needs to be arranged in sparse matrix
# triplet form, i.e., each line in the file contains three numbers
# user_index item_index rating

train_matrix <- data_memory(user_index = train$userId, item_index = train$movieId,
                            rating = train$rating, index1 = T)

test_matrix <- data_memory(user_index = test$userId, item_index = test$movieId, index1 = T)

# Create a model object (a Reference Class object in R) by calling Reco()

rec <- Reco()
```

We use cross validation to select the best tuning parameters.

```r
# Call the $tune() method to select best tuning parameters

opts = rec$tune(train_matrix, opts = list(dim = c(10, 20, 30), lrate = c(0.05, 0.1, 0.2),
                                          costp_l1 = 0, costq_l1 = 0,
                                          nthread = 2))

# Display best tuning parameters

print(opts$min)
```

```
## $dim
## [1] 30
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.7970839
```

The following code trains a recommender model. It will read from a training data source and create a model file at the specified location. The model file contains necessary information for prediction. And the output is stored in memory. The resulting RMSE is compared with RMSE from other models.

```r
#Train the model by calling the $train() method. A number of parameters can be set
#inside the function, possibly coming from the result of $tune()

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
rec$train(train_matrix, opts = c(dim = 30, costp_l1 = 0, costp_l2 = 0.01,
                                 costq_l1 = 0,costq_l2 = 0.1, lrate = 0.1,
                                 verbose = FALSE))

# Use the $predict() method to compute predicted values

predicted_ratings <- rec$predict(test_matrix, out_memory())

# Create a results table with matrix factorization

factorization <- RMSE(test$rating, predicted_ratings)
```

```r
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Model with Matrix Factorization",
                                 RMSE = factorization))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600537
Model with Movie Effect	0.9429615
Model with Movie + User Effects	0.8646843
Model with Regularized Movie and User Effect	0.8645518
Model with Matrix Factorization	0.7856736

```r
#---------------------------------------------------------------------------
```

Now that we have developed matrix factorization for train and test set, we finally calculate RMSE on edx and validation set.

```r
# The data file for edx and validation set needs to be arranged in sparse matrix
# triplet form, i.e., each line in the file contains three numbers
# user_index item_index rating

edx_matrix <- data_memory(user_index = edx$userId, item_index = edx$movieId,
                          rating = edx$rating, index1 = T)

validation_matrix <- data_memory(user_index = validation$userId, item_index = validation$movieId, index

# Create a model object (a Reference Class object in R) by calling Reco()

rec_final <- Reco()

# Call the $tune() method to select best tuning parameters along a set of candidate values

opts = rec_final$tune(edx_matrix, opts = list(dim = c(10, 20, 30), lrate = c(0.05, 0.1, 0.2),
                                             costp_l1 = 0, costq_l1 = 0,
                                             nthread = 2))

# Display best tuning parameters

print(opts$min)
```

```
## $dim
## [1] 30
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
```

```
## [1] 0.1
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.7920613
```

```r
#Train the model by calling the $train() method. A number of parameters can be set
#inside the function, possibly coming from the result of $tune()

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
rec_final$train(edx_matrix, opts = c(dim = 30, costp_l1 = 0, costp_l2 = 0.01,
                                     costq_l1 = 0,costq_l2 = 0.1, lrate = 0.1,
                                     verbose = FALSE))

# Use the $predict() method to compute predicted values

predicted_ratings <- rec_final$predict(validation_matrix, out_memory())

# Create a results table with matrix factorization

factorization_final <- RMSE(validation$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Model with Matrix Factorization on validation test",
                                 RMSE = factorization_final))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0600537
Model with Movie Effect	0.9429615
Model with Movie + User Effects	0.8646843
Model with Regularized Movie and User Effect	0.8645518
Model with Matrix Factorization	0.7856736
Model with Matrix Factorization on validation test	0.7828578

Conclusion

1. Huge data set does not allow us to train various models, so we train models that can process data with available RAM in the computer.
2. We see there is an improvement from basic model to model with movie effect and further improvement in model with movie and user effect.
3. There is not much improvement in RMSE with model using Regularization due to huge data set.
4. Matrix Factorization method alone gives a huge improvement in RMSE.

Advantages of Matrix Factoriszation using recosystem package:

recosystem is an R wrapper of the LIBMF library developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin (http://www.csie.ntu.edu.tw/~cjlin/libmf/), an open source library for recommender system using parallel marix factorization. (Chin, Yuan, et al. 2015)

recosystem is a wrapper of LIBMF, hence it inherits most of the features of LIBMF, and additionally provides a number of user-friendly R functions to simplify data processing and model building. Also, unlike most other R packages for statistical modeling that store the whole dataset and model object in memory, LIBMF (and hence recosystem) can significantly reduce memory use, for instance the constructed model that contains information for prediction can be stored in the hard disk, and output result can also be directly written into a file rather than be kept in memory.