Recommendation System, the MovieLens

Introduction

The goal of this project is to develop a recommendation system. Movielens 10M dataset is used in this project and we would like to know the rating that user U gives to movie M. To start with, the following code is given in the course to generate the datasets.

```
############################################################
# Create edx set, validation set (final hold-out test 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
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1
```

```
## -- 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 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
```

```
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)`
```

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

```
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")
```

```
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The edx dataset consists of 9000055 rows(observations) and 6 columns (variables) and it looks as follows:

```
glimpse(edx)
```

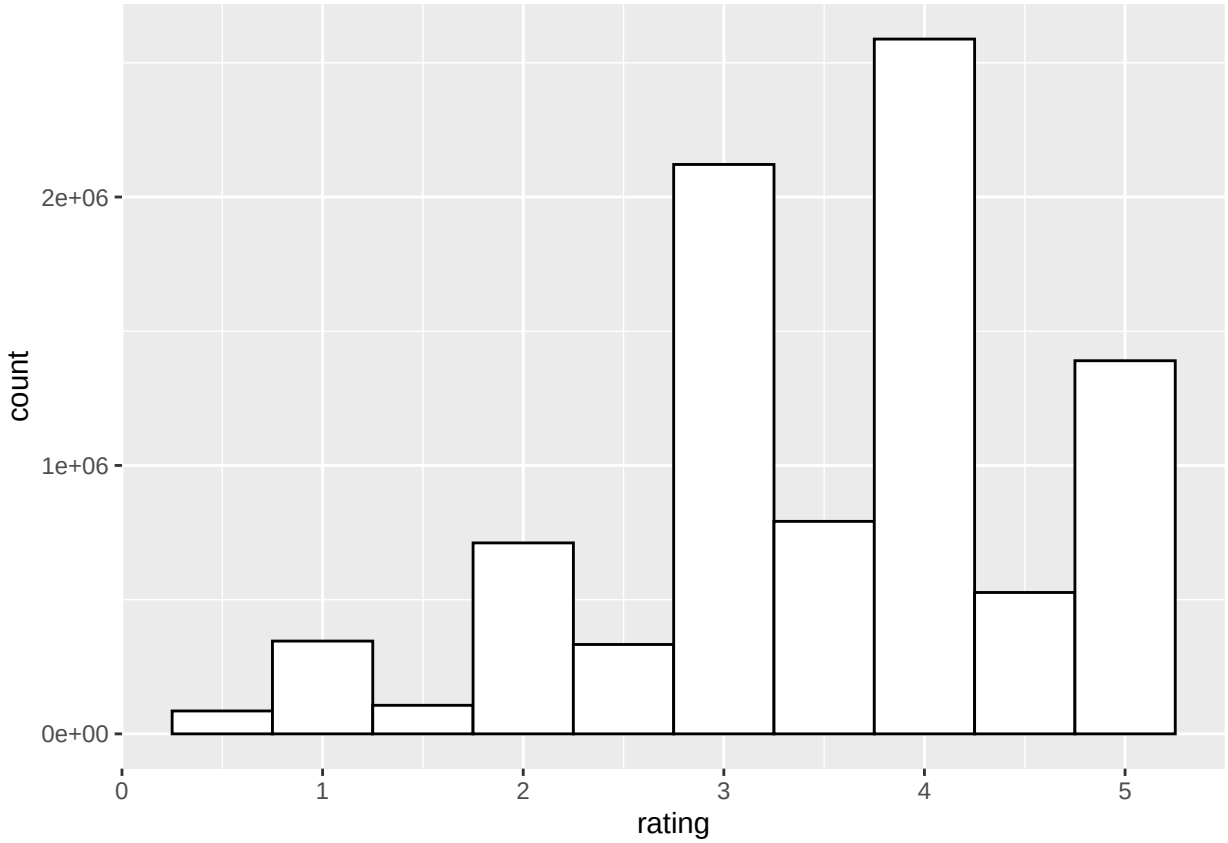
```
## Rows: 9,000,055
## Columns: 6
```

```
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

The dataset shows the rating which each users have given to some movies. In addition the titles and genres of the movies are provided in the dataset. We are more interested in the first three columns (the users, the movies and the ratings). Here we try to get more insight on the dataset. Following figures shows the distributions of the ratings.

```
ggplot(edx, aes(x=rating)) + geom_histogram() +
geom_histogram(color="black", fill="white", binwidth = .5)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



As we see number of ratings above 3 is more than number of ratings below 3. Which generally speaking, means these movies are satisfying for the users in the data set. Another implication of this is that the given data is slightly unbalanced.

Following code shows that, there is 10677 movies and 69878 users in the dataset.

```
edx$movieId %>% unique() %>% length()
```

```
## [1] 10677
```

```
edx$userId %>% unique() %>% length()
```

```
## [1] 69878
```

We can create a matrix which every row represents a user and every column represents a movie by following piece of code.

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

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

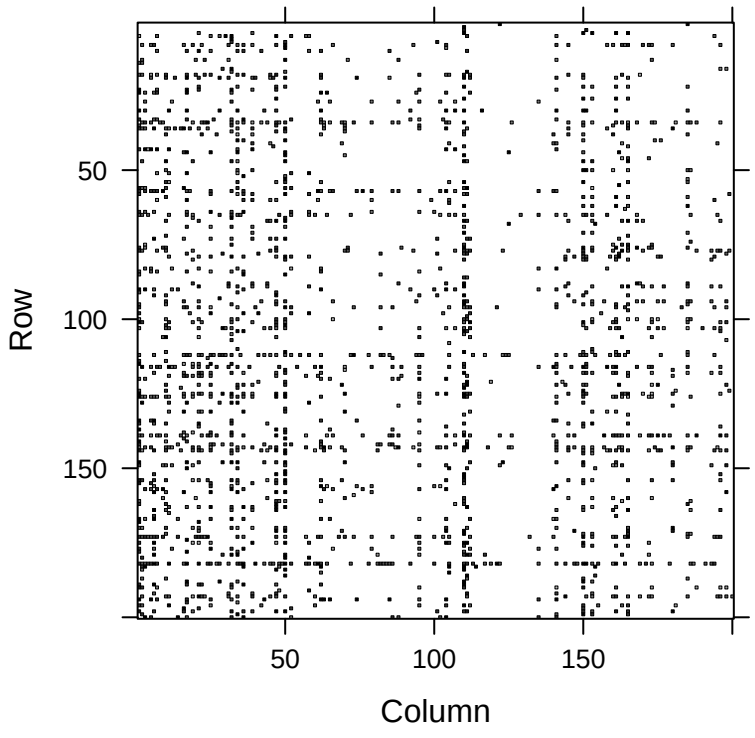
```
library(Matrix)

ratings_matrix <- sparseMatrix(i = edx$userId, j = edx$movieId , x = edx$rating)
dim(ratings_matrix)
```

```
## [1] 71567 65133
```

Each entry in this matrix is the rating given by a user to a movie. The matrix is very sparse. We can show this by visualizing the first 200 rows and columns.

```
userId <- 1:200
movieId <- 1:200
image(ratings_matrix[1:200,1:200])
```



The white areas shows the elements without any rating (N/A). The whole idea is to estimate the rating for the white areas.

Method

Method 1 The easiest way is to take an average from all given ratings and assume that all the missing ratings are equal to this average value.

```
avg_value <- mean(edx$rating)
avg_value
```

```
## [1] 3.512465
```

Doing this we can calculate the rsme for this simple model as follows.

```
actual <- validation$rating
predicted <- rep(avg_value,nrow(validation))
Metrics::rmse(actual,predicted)
```

```
## [1] 1.061202
```

Method 2 Another approach is matrix factorization which is a collaborative filtering method. It decomposes user-movie matrix into the product of two matrices which are called latent factors. These two matrices have lower dimensionality which is computationally beneficial. Here we use *recosystem* library. We first need to tune the model. Following codes shows a suggestion. The tuner goes through different combinations of the parameters and finds the best combination.

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

```
## Loading required package: recosystem
```

```
library(recosystem)
train_reco <- with(edx, data_memory(user_index = userId, item_index = movieId, rating = rating))
test_reco <- with(validation, data_memory(user_index = userId, item_index = movieId, rating = rating))
reco <- Reco()

para_reco <- reco$tune(train_reco, opts = list(dim = c(20, 30),
                                               costp_l1 = 0,
                                               costq_l1 = 0,
                                               costp_l2 = c(0.01, 0.1),
                                               costq_l2 = c(0.01, 0.1),
                                               lrate = c(0.01, 0.1)))
```

Note that *dim* in the *opts* determines the number of latent factors. More information about the settings are given in the following link:

https://rdrr.io/cran/recosystem/man/tune.html

The result of tuning is presented in the following table:

```
print(para_reco)
```

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

```
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.7922573
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   20        0     0.01        0     0.01  0.01 0.8676665
## 2   30        0     0.01        0     0.01  0.01 0.8657039
## 3   20        0     0.10        0     0.01  0.01 0.8701999
## 4   30        0     0.10        0     0.01  0.01 0.8672400
## 5   20        0     0.01        0     0.10  0.01 0.8808520
## 6   30        0     0.01        0     0.10  0.01 0.8780285
## 7   20        0     0.10        0     0.10  0.01 0.8822925
## 8   30        0     0.10        0     0.10  0.01 0.8804835
## 9   20        0     0.01        0     0.01  0.10 0.8092317
## 10  30        0     0.01        0     0.01  0.10 0.8205792
## 11  20        0     0.10        0     0.01  0.10 0.8002434
## 12  30        0     0.10        0     0.01  0.10 0.8016765
## 13  20        0     0.01        0     0.10  0.10 0.7949162
## 14  30        0     0.01        0     0.10  0.10 0.7922573
## 15  20        0     0.10        0     0.10  0.10 0.8220457
## 16  30        0     0.10        0     0.10  0.10 0.8197574
```

As we can see the best result is achieved by following settings:

dim = 30

costp_l1 = 0

costp_l2 = 0.01

costq_l1 = 0

costq_l2 = 0.1

lrate = 0.1

Now we can train the model by these settings as follows:

```
reco$train(train_reco, opts = c(para_reco$min, nthread = 4, niter = 50))
```

```
## iter      tr_rmse          obj
##    0       0.9703   1.1992e+07
##    1       0.8727   9.8843e+06
##    2       0.8385   9.1726e+06
##    3       0.8163   8.7542e+06
##    4       0.8005   8.4660e+06
##    5       0.7888   8.2679e+06
##    6       0.7795   8.1254e+06
##    7       0.7717   8.0027e+06
```

```
##    8       0.7650   7.9088e+06
##    9       0.7594   7.8282e+06
##   10       0.7542   7.7628e+06
##   11       0.7498   7.7046e+06
##   12       0.7457   7.6565e+06
##   13       0.7420   7.6113e+06
##   14       0.7387   7.5731e+06
##   15       0.7357   7.5419e+06
##   16       0.7328   7.5093e+06
##   17       0.7302   7.4834e+06
##   18       0.7277   7.4569e+06
##   19       0.7254   7.4331e+06
##   20       0.7233   7.4140e+06
##   21       0.7213   7.3936e+06
##   22       0.7193   7.3773e+06
##   23       0.7176   7.3597e+06
##   24       0.7159   7.3439e+06
##   25       0.7143   7.3287e+06
##   26       0.7129   7.3171e+06
##   27       0.7115   7.3048e+06
##   28       0.7101   7.2931e+06
##   29       0.7089   7.2814e+06
##   30       0.7077   7.2721e+06
##   31       0.7066   7.2624e+06
##   32       0.7056   7.2524e+06
##   33       0.7046   7.2447e+06
##   34       0.7036   7.2355e+06
##   35       0.7027   7.2287e+06
##   36       0.7018   7.2215e+06
##   37       0.7009   7.2141e+06
##   38       0.7002   7.2096e+06
##   39       0.6994   7.2018e+06
##   40       0.6987   7.1962e+06
##   41       0.6980   7.1905e+06
##   42       0.6973   7.1854e+06
##   43       0.6968   7.1814e+06
##   44       0.6961   7.1771e+06
##   45       0.6955   7.1705e+06
##   46       0.6950   7.1676e+06
##   47       0.6944   7.1629e+06
##   48       0.6939   7.1595e+06
##   49       0.6934   7.1552e+06
```

```
predicted <- reco$predict(test_reco, out_memory())
Metrics::rmse(actual,predicted)
```

```
## [1] 0.7806023
```

Here we used 50 iterations achieved *rmse* equal to 0.6929

Conclusion

As mentioned the goal is to fill the missing values in the sparse user-movie matrix with the best estimated values. One simple approach was to assume that all missing ratings are the same and equal to average rating. In this case the *rmse* is equal to 1.061.

As we can see the matrix factorization yields much better results. In this case, in the best results that we achieved, the *rmse* is equal to 0.6929