Capstone Project Report

Building a recommender system using 10M version of Movielens dataset

Endri Raco

06/11/2019

List of Tables

List of Figures

Introduction

Recommender systems are information filtering tools that aspire to predict the rating for users and items, predominantly from big data to recommend their likes. Movie recommendation systems provide a mechanism to assist users in classifying users with similar interests. This makes recommender systems essentially a central part of websites and e-commerce applications. This project focuses on building a movie recommendation system using data from 10M version of movielens dataset. Several Machine Learning techniques such as Matrix Factorization, Regularization etc will be used to produce evaluation metrics such as root mean square error (RMSE) for the movie recommender system.

Importing data

MovieLense dataset contains the ratings that the users give to movies. Code used in "Importing data" section was previously provided by edX.

Let's start by checking if needed R packages for this project are installed. If not, code below will install them.

```
# required packages for our project
if(!require(kableExtra)) install.packages("kableExtra",
repos = "http://cran.us.r-project.org")
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")
```

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

Now we are ready for data downloading:

```r
# 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)
```

Some adjustments of downloaded data to have **movielens** dataframe as a result

```r
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")
```

When developing an algorithm, we usually have a dataset for which we know the outcomes, as we do with the heights: we know the sex of every student in our dataset. Therefore, to mimic the ultimate evaluation process, we typically split the data into two parts and act as if we don't know the outcome for one of these. We stop pretending we don't know the outcome to evaluate the algorithm, but only after we are done constructing it. We refer to the group for which we know the outcome, and use to develop the algorithm, as the training set. We refer to the group for which we pretend we don't know the outcome as the test set. A standard way of generating the training and test sets is by randomly splitting the data. The caret package includes the function **createDataPartition** that helps us generates indexes for randomly splitting the data into training and test sets:

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating,
times = 1, p = 0.1, list = FALSE)
```

We use the result of the **createDataPartition** function call to define the training and test sets like this:

```
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

And some final adjustments before cleaning environment from unused elements.

```
# 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, movielens, removed)
```

Now, we can save the result of steps above (**edx** and **validation** dataframes) as R objects, so we can reload the final version of the data into the session for further analysis without repeating the process.

```
# Save our data as R objects
save(edx, file = "edx.RData")
save(validation, file = "validation.RData")
```

Describing Data

We stored the data for our project in two data frames. Let's access these datasets using the **load** function:

```
# Load data
load("edx.RData")
load("validation.RData")
```

First, we make a check if our data format is indeed **data frame**:

```
# Check format
class(edx)
```

```
[1] "data.frame"
```

```
class(validation)
```

```
[1] "data.frame"
```

Now let's take a look in our data. We start by finding out more about the structure of our **edx**:

```
as_tibble(edx) %>%
slice(1:5) %>%
style()
```

userId	movieId	rating	rate_year	title	premier_year	genres
1	122	5	1996	Boomerang	1992	Comedy|Romance
1	185	5	1996	Net, The	1995	Action|Crime|Thriller
1	292	5	1996	Outbreak	1995	Action|Drama|Sci-Fi|Thriller
1	316	5	1996	Stargate	1994	Action|Adventure|Sci-Fi
1	329	5	1996	Star Trek: Generations	1994	Action|Adventure|Drama|Sci-Fi

Now for **validation**:

```
as_tibble(validation) %>%
slice(1:5) %>%
style()
```

userId	movieId	rating	rate_year	title	premier_year	genres
1	231	5	1996	Dumb & Dumber	1994	Comedy
1	480	5	1996	Jurassic Park	1993	Action|Adventure|Sci-Fi|Thriller
1	586	5	1996	Home Alone	1990	Children|Comedy
2	151	3	1997	Rob Roy	1995	Action|Drama|Romance|War
2	858	2	1997	Godfather, The	1972	Crime|Drama

We see that **edx** data frame has 9000055 rows and 7 variables, while **validation** data frame has 999999 rows and 7.

Now let's print features of both data frames **edx** and **validation together** to reassure ourselves that both contain the same features.

```
library(dataCompareR)
comp_edx_val <- rCompare(edx, validation)
comp_summ <- summary(comp_edx_val)
comp_summ[c("datasetSummary", "ncolInAOnly", "ncolInBOnly", "ncolCommon", "rowsInAOnly",
```

```
$datasetSummary
  Dataset Name Number of Rows Number of Columns
1          edx        9000055                 7
2   validation         999999                 7

$ncolInAOnly
[1] 0

$ncolInBOnly
[1] 0

$ncolCommon
[1] 7

$rowsInAOnly
  indices_removed
1         2808866
2         7235076
3         4043383
4         2159766
5         3413160

$rowsInBOnly
[1] indices_removed
<0 rows> (or 0-length row.names)
```

```
$nrowCommon
[1] 999999
```

It is a good idea to check for dublicates so to create a general idea about number of distinct users, movies and genres.

```
# Distinct users, movies, genres
edx %>%
summarize(distinct_users = n_distinct(userId),
            distinct_movies = n_distinct(movieId),
            distinct_genres = n_distinct(genres)) %>%
  style()
```

distinct_users	distinct_movies	distinct_genres
69878	10677	797

Data Wrangling

When we printed **edx** and **validation** data frames as tibbles we noticed that we can make some arrangements in **title**, **timestamp** and **genres** columns to bring our data in a tidy format.

We are going to perform these tasks:

- Most of the movies have their **premier year** added to their **titles**. We will extract debut years in a separate column.
- Column **genres** has to be categorized. We will change the class of **genres** to **factor**
- **Timestamp** needs to be converted to **rate_year**.

```
tidydf <- function(df){
  df$genres <- as.factor(df$genres) #Convert genres to factor
  df$timestamp <- as.Date(as.POSIXct(df$timestamp, origin="1970-01-01"))
  #Convert timestamp
  names(df)[names(df) == "timestamp"] <- "rate_year" # Rename column timestamp to rate_
  df <- df %>%
```

```r
    mutate(title = str_trim(title), rate_year = year(rate_year)) %>%  #Mutate title and
    extract(title, c("title", "premier_year"), regex = "(.*)\\s\\((\\d+)\\)", convert =
  return(df)
}
# Transform our dataframes
edx <- tidydf(edx)
validation <- tidydf(validation)
```

Now our data frames look like this:

```r
as_tibble(edx)
```

```
# A tibble: 9,000,055 x 7
   userId movieId rating rate_year title          premier_year genres
    <int>   <dbl>  <dbl>     <dbl> <chr>                 <int> <fct>
 1      1     122      5      1996 Boomerang              1992 Comedy|Romance
 2      1     185      5      1996 Net, The               1995 Action|Crime|Thr~
 3      1     292      5      1996 Outbreak               1995 Action|Drama|Sci~
 4      1     316      5      1996 Stargate               1994 Action|Adventure~
 5      1     329      5      1996 Star Trek: Ge~         1994 Action|Adventure~
 6      1     355      5      1996 Flintstones, ~         1994 Children|Comedy|~
 7      1     356      5      1996 Forrest Gump           1994 Comedy|Drama|Rom~
 8      1     362      5      1996 Jungle Book, ~         1994 Adventure|Childr~
 9      1     364      5      1996 Lion King, The         1994 Adventure|Animat~
10      1     370      5      1996 Naked Gun 33 ~         1994 Action|Comedy
# ... with 9,000,045 more rows
```

```r
as_tibble(validation)
```

```
# A tibble: 999,999 x 7
   userId movieId rating rate_year title             premier_year genres
    <int>   <dbl>  <dbl>     <dbl> <chr>                    <int> <fct>
 1      1     231    5        1996 Dumb & Dumber             1994 Comedy
 2      1     480    5        1996 Jurassic Park             1993 Action|Advent~
 3      1     586    5        1996 Home Alone                1990 Children|Come~
 4      2     151    3        1997 Rob Roy                   1995 Action|Drama|~
 5      2     858    2        1997 Godfather, The            1972 Crime|Drama
 6      2    1544    3        1997 Lost World: Jura~         1997 Action|Advent~
 7      3     590    3.5      2006 Dances with Wolv~         1990 Adventure|Dra~
 8      3    4995    4.5      2005 Beautiful Mind, A         2001 Drama|Mystery~
 9      4      34    5        1996 Babe                      1995 Children|Come~
10      4     432    3        1996 City Slickers II~         1994 Adventure|Com~
# ... with 999,989 more rows
```

Probably is a good idea in this step to check for NA values:

```r
# Check edx dataframe for NA values
edx_na <- edx %>%
  filter(is.na(title) | is.na(year))
glimpse(edx_na)
```

```
Observations: 0
Variables: 7
$ userId       <int>
$ movieId      <dbl>
$ rating       <dbl>
$ rate_year    <dbl>
$ title        <chr>
$ premier_year <int>
$ genres       <fct>
```

```r
# Check validation dataframe for NA values
validation_na <- validation %>%
  filter(is.na(title) | is.na(year))
glimpse(validation_na)
```

```
Observations: 0
Variables: 7
$ userId       <int>
$ movieId      <dbl>
$ rating       <dbl>
$ rate_year    <dbl>
$ title        <chr>
$ premier_year <int>
$ genres       <fct>
```

Exploring Data

Ratings frequency

From this step to the development of our algorithm we will continue using **edx** dataframe. We will come back to **validation** dataframe to perform a final test of our algorithm, predict movie ratings in the validation set as if they were unknown.

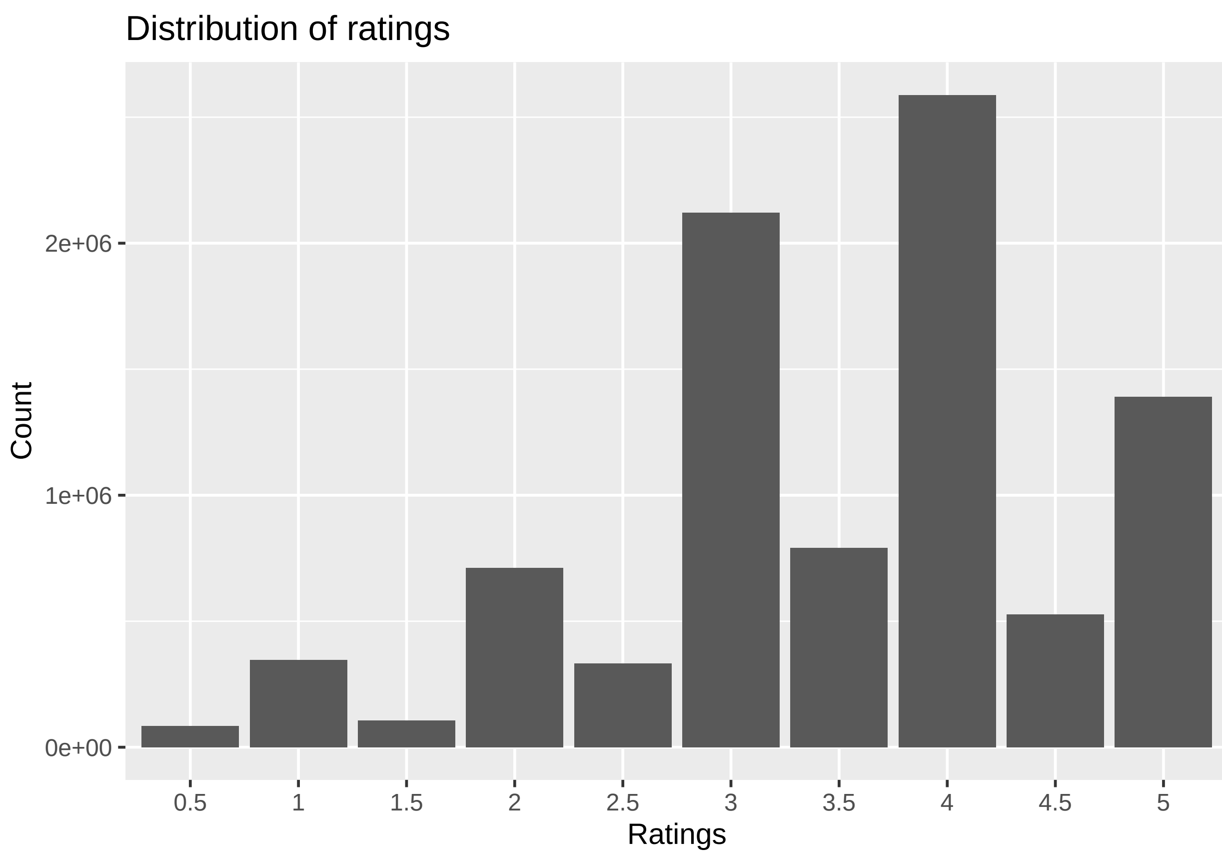
Now let's begin our exploration by looking at **rating** variable.

```
# Check frequencies of ratings unique values
table_rating <- as.data.frame(table(edx$rating))
colnames(table_rating) <- c("Rating", "Frequencies")
table_rating
```

```
   Rating Frequencies
1     0.5       85374
2       1      345679
3     1.5      106426
4       2      711422
5     2.5      333010
6       3     2121240
7     3.5      791624
8       4     2588430
9     4.5      526736
10      5     1390114
```

Now, we will build a frequency plot of the ratings using **ggplot2**. For this step we need to convert **ratings** into categories:

```
# Frequency plot of the ratings
table_rating %>% ggplot(aes(Rating, Frequencies)) +
geom_bar(stat = "identity") +
labs(x="Ratings", y="Count") +
ggtitle("Distribution of ratings")
```



We notice from the figure that most of the ratings are above 2, and the most common is 4.

Most viewed movies

Now let's check 10 most viewed movies:

```
# Top movies by number of views
tmovies <- edx %>% select(title) %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  arrange(desc(count)) %>% head(10)
# Print top_movies
style(tmovies)
```

title	count
Pulp Fiction	31362
Forrest Gump	31079
Silence of the Lambs, The	30382
Jurassic Park	29360
Shawshank Redemption, The	28015
Braveheart	26212
Fugitive, The	26020
Terminator 2: Judgment Day	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)	25672
Batman	24585

Results

Conclusion