MovieLens Project

Author: Gabrielmaria Scozzarro ; Date:21/05/2020

Executive summary

GroupLens is a research lab in the Department of Computer Science and Engineering at the University of Minnesota. The GroupLens team has collected and made available rating data sets from the MovieLens web site (http://movielens.org (http://movielens.org)). The data sets were collected over various periods of time and for this reason it is very suitable for the implementation of machine learning algorythm. those algorithms improve automatically through data or what we can call "experience". It is considered as a subset of AI. Machine learning algorithms are based on well known mathematical model feed with collected data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications,spanning from IT to robotics application, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks.

The goal of this project is to build a rating recommender system usign the machine learning feeded with the MovieLens data set. This report will guide you through the methods and the results of this project. For an easy and immediate comparison with other approach and for quality estimate we will use the residual mean squared error (RMSE) as the main KPI with a target value of 0.8641.

Methods and analysis

We use the MovieLens 10M dataset that consists of 10M observed ratings from almost 70K different users applied to 10K different movies. We already diveded the data set in 2 part with a proportion of 90/10 for train_set and test_set respectively.

In [51]:

```
train_set<- readRDS(file = "edx.rds")
test_set<- readRDS(file = "validation.rds")
```

To perform a preliminar analysis of the dataset we use the following packages:

In [52]:

```
library(tidyverse)
library(lubridate)
```

and options:

In [53]:

```
options(repr.plot.width=18, repr.plot.height=9)
```

The data structure is:

In [54]:

```
dim(train_set)
```

9000055 · 6

In [55]:

```
head(train_set, 10)
```

A data.frame: 10 × 6

	userId	movieId	rating	timestamp	title	genres
	<int>	<dbl>	<dbl>	<int>	<chr>	<chr>
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
8	1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
9	1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
10	1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical
11	1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy

A deeper analysis on the rating values:

In [56]:

```
summary(train_set$rating)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.500   3.000   4.000   3.512   4.000   5.000 
```

To perform an analysis of the rating behavior through the years, on the train_set we convert the time stamp in DateTime format:

In [57]:

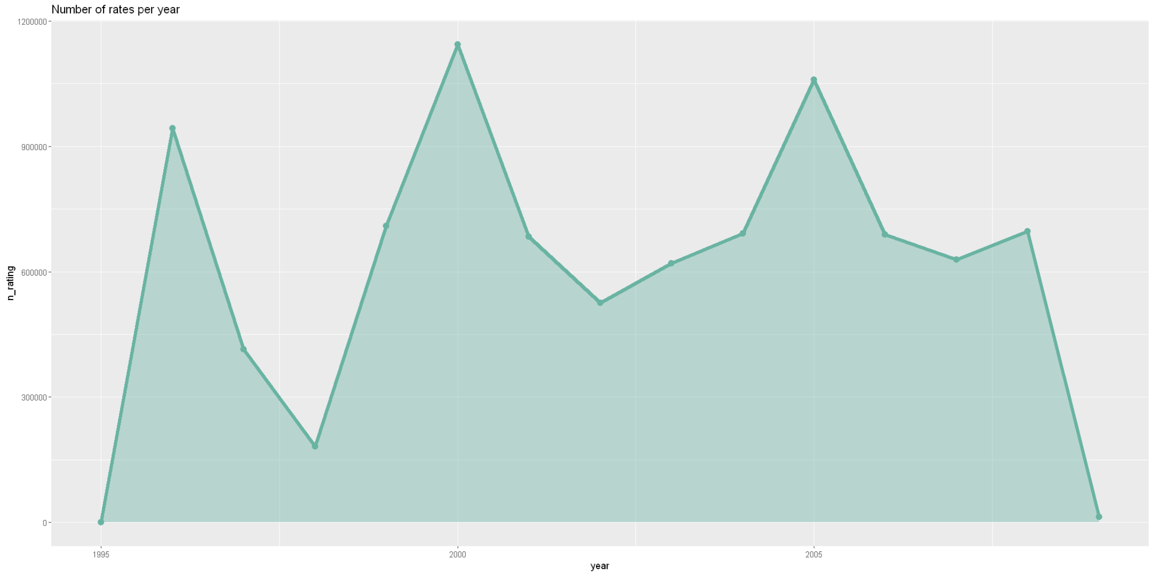
```
train_set<- train_set %>%
  mutate(timestamp = as_datetime(timestamp))

train_set %>%
  mutate(year = year(timestamp)) %>%
  group_by(year) %>%
  summarize(n_rating = n()) %>%
  print(n = 15)
```

```
# A tibble: 15 x 2
    year n_rating
   <dbl>    <int>
 1  1995        2
 2  1996   942772
 3  1997   414101
 4  1998   181634
 5  1999   709893
 6  2000  1144349
 7  2001   683355
 8  2002   524959
 9  2003   619938
10  2004   691429
11  2005  1059277
12  2006   689315
13  2007   629168
14  2008   696740
15  2009    13123
```

In [58]:

```
train_set %>%
  mutate(year = year(timestamp)) %>%
  group_by(year) %>%
  summarize(n_rating = n()) %>%
  ggplot(aes(year,n_rating)) +
  geom_area( fill="#69b3a2", alpha=0.4) +
  geom_line(color="#69b3a2", size=2) +
  geom_point(size=3, color="#69b3a2") +
  ggtitle("Number of rates per year")
```

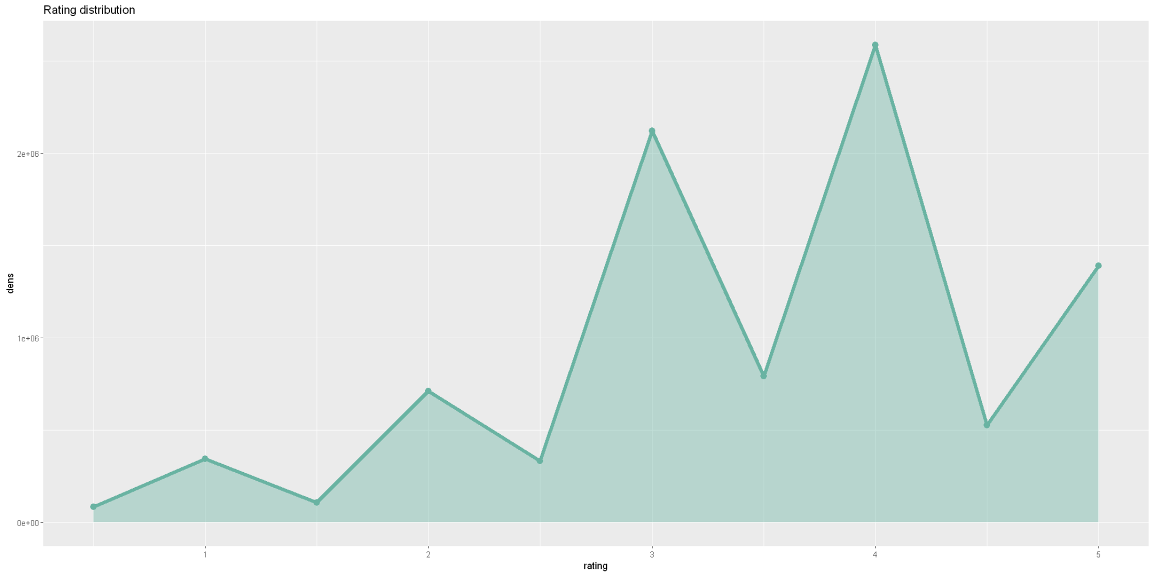


We can oberser that in the train_set we have 3 spikes 1996, 2000 and 2005.

Here’s a distribution of the user ratings in the train_set:

In [59]:

```
train_set %>%
  group_by(rating) %>%
  summarize(dens = n()) %>%
  ggplot(aes(rating,dens)) +
  geom_area( fill="#69b3a2", alpha=0.4) +
  geom_line(color="#69b3a2", size=2) +
  geom_point(size=3, color="#69b3a2") +
  ggtitle("Rating distribution")
```



It appears that users are quite generous in their ratings. In fact, as we seen in a previous analysis the mean rate is 3.51, most of the rates are on the interval 3-5 with a prevelance for round value. We can also investigate how much users are willing to rate a movie like this:

In [60]:

```
length(unique(train_set$userId))
```

69878

In [61]:

```
train_set%>%
  group_by(userId) %>%
  summarize(n_rating = n()) %>%
  summary()
```

```
     userId          n_rating
 Min.   :    1   Min.   :  10.0
 1st Qu.:17943   1st Qu.:  32.0
 Median :35799   Median :  62.0
 Mean   :35782   Mean   : 128.8
 3rd Qu.:53620   3rd Qu.: 141.0
 Max.   :71567   Max.   :6616.0
```

There are 69878 unic users. Each user rated at least 10 movies, so the distribution should not be caused just by chance variance in the quality of movies.

We can now explore the genres feature like this:

In [62]:

```
train_set%>%
  group_by(genres) %>%
  summarize(movies_per_genre = n()) %>%
  top_n(10, movies_per_genre)
```

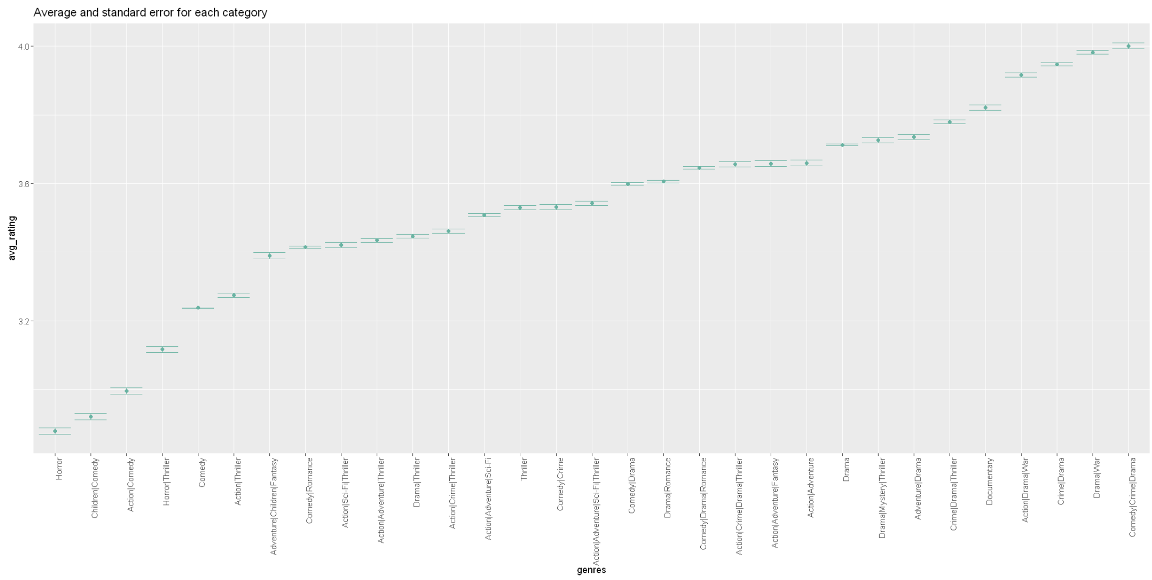
A tibble: 10 × 2

genres	movies_per_genre
<chr>	<int>
Action|Adventure|Sci-Fi	219938
Action|Adventure|Thriller	149091
Comedy	700889
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Comedy|Romance	365468
Crime|Drama	137387
Drama	733296
Drama|Romance	259355
Drama|Thriller	145373

We can see that movies can be categorized using one or more genre and which are the most present genres in the train_set. We can also compute the average and standard error for each category and find the category with the highest or lowest average rating:

In [63]:

```
train_set %>% group_by(genres) %>%
  summarize(n_rating = n(), avg_rating = mean(rating), se_rating = sd(rating)/sqrt(n
())) %>%
  filter(n_rating >= 50000) %>%
  mutate(genres = reorder(genres, avg_rating)) %>%
  ggplot(aes(x = genres, y = avg_rating, ymin = avg_rating - 2*se_rating, ymax = avg_ra
ting + 2*se_rating)) +
  geom_point(color = "#69b3a2" ) +
  geom_errorbar(color = "#69b3a2") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Average and standard error for each category")
```



With a filter that keeps only genres with more than 50K ratings we can appreciate that Horror movies has the lowest average ratings and Comedy|Crime|Drama the highest. Now we can investigate which movie title has the best ratings like this:

In [64]:

```
length(unique(train_set$movieId))
```

10677

In [65]:

```
train_set %>% group_by(movieId, title) %>%
    summarize(count = n()) %>%
    arrange(desc(count)) %>%
    head(10)
```

A grouped_df: 10 × 3

movieId	title	count
<dbl>	<chr>	<int>
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212
457	Fugitive, The (1993)	25998
589	Terminator 2: Judgment Day (1991)	25984
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
150	Apollo 13 (1995)	24284

Among the 10677 different movies rated from 1995 to 2019 these are the top 10 most rated.

Recommendation system

For a recommendation system we need to identify the most important features that can contain helpful information to predict the rating any given user will assign to any movie. We start defining the RMSE formula needed for the quality assestment of the procces we are going to build. We choose to apply a collaborative filtering approach with a Matrix Factorization to increment the accuracy of the prediction.

As know collaborative filtering is a method for automatic predictions of users interests by collecting preferences or taste information from many other users. The reasoning behind this approach is that if a user A has the same sentiment as a user B on an matter, A is more likely to have B's opinion on another matter compared to a randomly chosen person. Note that the most fascinating thing about this approach is that these predictions are specific to the user, but use information is harvested from many users.

To accomplish our goal we will make use of the recosystem package which is an R wrapper of the LIBMF, an open source library for recommender system using parallel matrix factorization. In addition this packages allow us to speedup all processes using parallel computation on many thread.

In [78]:

```
library(recosystem)
```

Defining RMSE:

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

In [67]:

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

Now we can convert the 2 data set in the recosystem desired imput format:

In [68]:

```
train_data <-  with(train_set, data_memory(user_index = userId, item_index = movieId, r
ating     = rating))

test_data  <-  with(test_set,  data_memory(user_index = userId, item_index = movieId, r
ating     = rating))
```

We reate a model object or you can call it a Reference Class object in R by calling the function Reco() from the recosystem package.

In [70]:

```
r <-  recosystem::Reco()
```

Recosystem allows to optimize train parameters calling the $tune option:

In [71]:

```
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = 6, niter = 10))
```

Finalyy we can train the algorithm with the train_data we prepared before:

In [72]:

```
r$train(train_data, opts = c(opts$min, nthread = 6, niter = 20))
```

```
iter      tr_rmse          obj
   0       0.9684   1.1944e+07
   1       0.8729   9.8876e+06
   2       0.8380   9.1670e+06
   3       0.8153   8.7386e+06
   4       0.8002   8.4605e+06
   5       0.7888   8.2675e+06
   6       0.7794   8.1199e+06
   7       0.7713   7.9989e+06
   8       0.7645   7.9034e+06
   9       0.7584   7.8195e+06
  10       0.7533   7.7557e+06
  11       0.7486   7.6977e+06
  12       0.7444   7.6465e+06
  13       0.7406   7.6035e+06
  14       0.7371   7.5634e+06
  15       0.7341   7.5306e+06
  16       0.7311   7.4978e+06
  17       0.7284   7.4705e+06
  18       0.7259   7.4436e+06
  19       0.7236   7.4204e+06
```

Results

Now that the algorythm is trained we can compute the predicted value using the test_data:

In [74]:

```
y_hat <-  r$predict(test_data, out_memory())
```

And verify the RMSE for comparison:

In [76]:

```
RMSE(test_set$rating, y_hat)
```

0.782329091466384

The **RMSE** is therefore **0.78** wich is far below the target of **0.86**

As espected the algorythm built with the recosystem package perform very well for the goal we set up at the beginning. In fact, we have been successful in reduce the target RMSE by **0.08**.

Conclusion

The easiest try to improve our recommendation system could be enrich the data set used with new and different data for example lenght of the movie or box offices.

An advance improvment can be obtained combining both content based and collaborative filtering methods:

Rating matrix can be also compressed by a neural network. The so called autoencoder is very similar to the matrix factorization. Deep autoencoders, with multiple hidden layers and nonlinearities are more powerful but harder to train. Neural net can be also used to preprocess item attributes so we can combine content based and collaborative approaches.