Capstone HarvardX - Project MovieLens

Papacosmas Nicholas

May 2019

Contents

Abstract

Recommender systems algorithms are applied into industry on various business domains. The most popular are:

-The **movie recommendations** like the one used by Netflix.

-And the **(related items recommendations)** during online purchases.

There are 2 types of recommender systems:

-**Content filtering** (based on the description of the item also called meta data or side information)

-And **collaborative Filtering**: Those techniques are calculating the similarity measures of the target ITEMS and finding the minimum (Euclidean distance, or Cosine distance, or other metric, it depends on the algorithm). This is done by filtering the interests of a user, by collecting preferences from many users **(collaborating)**.

Matrix factorization with parallel stochastic gradient descent, is an effective algorithm used to create a recommender system.The approach is to approximate the rating matrix

$R_{m \times n}$ by the product of two matrixes containing lower dimensions, $P_{k \times m}$ and $Q_{k \times n}$, in a way that

$$R \approx P'Q$$

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

A usual equation for the P and th Q is given by the below optimization problem :

$$\min_{P,Q} \sum_{(u,v)\in R} \left[f(p_u, q_v; r_{u,v}) + \mu_P ||p_u||_1 + \mu_Q ||q_v||_1 + \frac{\lambda_P}{2} ||p_u||_2^2 + \frac{\lambda_Q}{2} ||q_v||_2^2 \right]$$

where the (u, v) are the locations of the real entries in the R, $r_{u,v}$ is the real rating, f is the loss function, and $\mu_P, \mu_Q, \lambda_P, \lambda_Q$ are the usual penalization parameters used by many algorithms to avoid overfitting.

The procedure of solving the matrix P and Q is the model training, and the selection of choosing penalization parameters is the hyper parameters tuning. After obtaining the P and the Q,

we can then predict :

$\hat{R}_{u,v} = p'_u q_v$.

Many thanks to Yixuan Qiu from Carnegie Mellon University Source: link

Introduction

The purpose of this R project is to create a **rating recommender system through machine learning training.** That recommender system will be able to predict a users rating into a new movie. Or the user preference for a movie.

The most famous recommender training event was the competition launched by **Netflix with one milion dollar price.** I will use the 10M (millions) rows rating dataset named MovieLens created by the University of Minnesota. It was released at 1/2009 so our newest movies are until 2008. In order to find a pattern and behavior of the data, the data sets where "enhanced" by many new features (dimensions). As validation of the models i wil use RMSE (regression approach). During the project is given more explanations.

Many algorithms and data transformations where used in order to achieve the lowest RMSE. Such us:

-**Matrix Factorization with parallel stochastic gradient descent, H2o stacked ensembles of (GBM,GLM,DRF,NN).**

-**And H2o Auto ML models.** More details is given during the project. You can download my models from my github.

collaborative filtering underlying assumption is that if a person X has the same opinion as a person Y then the recommendation system should be based on preferences of person Y (similarity). I will enhance the collaborative filtering with the application of:

-**Matrix Factorization with parallel stochastic gradient descent algorithms**. MF is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by **decomposing the user-item interaction matrix into the product of two loir dimensionality rectangular matrices**.

This family of methods became **widely known during the Netflix prize challenge due to its effectiveness as reported by Simon Funk in his 2006 blog where he shared his findings with the research community [link](https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)

We will apply **Matrix Factorization with parallel stochastic gradient descent**. With the help of *"recosystem"* package it is an R wrapper of the LIBMF library which creates a Recommender System by Using Parallel Matrix Factorization.

-The main task of recommender system is to predict unknown entries in the rating matrix based on observed values.

More info on the recosystem package and the techniques link

PROJECT_GIT: (https://github.com/papacosmas/MovieLens)

CONTACT:(https://www.linkedin.com/in/niko-papacosmas-mba-pmp-mcse-695a2695/)

Data Observation

Data structure of the edx (training set)

```
Observations: 9,000,055
Variables: 6
$ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
$ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
$ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
$ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
$ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

Its class'data.frame'with : -9000047 obs (rows) -6 variables (features)

The same movie entry might belong to more than one genre. Every discrete rating is on a discrete row.

First entries of the edx (training set)

```
  userId movieId rating timestamp                         title
1      1     122      5 838985046              Boomerang (1992)
2      1     185      5 838983525               Net, The (1995)
4      1     292      5 838983421               Outbreak (1995)
5      1     316      5 838983392               Stargate (1994)
6      1     329      5 838983392 Star Trek: Generations (1994)
7      1     355      5 838984474       Flintstones, The (1994)
                         genres
1                Comedy|Romance
2         Action|Crime|Thriller
4  Action|Drama|Sci-Fi|Thriller
5       Action|Adventure|Sci-Fi
6 Action|Adventure|Drama|Sci-Fi
7       Children|Comedy|Fantasy
```

It looks we have to transform the timestamp which represents the (rating date), since the release date is inside the movie (title) column.

We will extract the release date from the movie title. And we will create a new matrix with more dimensions, containing every movie genre separately as factor.

Summary of the edx (training set)

```
    userId         movieId          rating        timestamp        
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
 1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08  
 Median :35738   Median : 1834   Median :4.000   Median :1.035e+09  
 Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09  
 3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09  
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
   title              genres         
 Length:9000055     Length:9000055    
 Class :character   Class :character  
 Mode  :character   Mode  :character  
```

The rating mean shows that users **are rating above average rating (3.512) right skewed**

-**Rating (our dependent variable y) has 10 continuous values from 0 until 5.** Its row has one given rating by one user for one movie.

-Rating is our dependent (target variable) y
-userId, movieId, timestamp (date&time) are: quantitative - Discrete unique numbers.
-Title and genres are: qualitative and not unique.

Data structure of the validation (testing set)

```
Observations: 999,999
Variables: 6
$ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5...
$ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 4...
$ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3....
$ timestamp <int> 838983392, 838983653, 838984068, 868246450, 86824564...
$ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Hom...
$ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Child...
```

Its class 'data.frame': With 999999 obs. of 6 variables: Its exactly the 10% of our training set. And has the same 6 features

First entries of the validation (testing set)

```
  userId movieId rating timestamp
1      1     231      5 838983392
2      1     480      5 838983653
3      1     586      5 838984068
4      2     151      3 868246450
5      2     858      2 868245645
6      2    1544      3 868245920
                                                      title
1                                     Dumb & Dumber (1994)
2                                     Jurassic Park (1993)
3                                        Home Alone (1990)
4                                           Rob Roy (1995)
5                                    Godfather, The (1972)
6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
                                  genres
1                                 Comedy
2       Action|Adventure|Sci-Fi|Thriller
3                        Children|Comedy
4               Action|Drama|Romance|War
5                            Crime|Drama
6 Action|Adventure|Horror|Sci-Fi|Thriller
```

Same as our training set. So we will perform the same data transformation on both training and test datasets. we mentioned earlier we could **add features in our datasets in order to analyse for** correlations,** and if they exist they would help our ML models. By adding a feature**of release year and year rated. So we will create 2 new dataframes with that features for train set and test sets.

We will transform timestamp to year rated, and extract the premier data from the movie title and add it as a separate feature.

```
[1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
[1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

Check the data sets for consistency

```
  userId movieId rating year_rated                        title
1      1     122      5       1996             Boomerang (1992)
2      1     185      5       1996              Net, The (1995)
3      1     292      5       1996              Outbreak (1995)
4      1     316      5       1996              Stargate (1994)
5      1     329      5       1996 Star Trek: Generations (1994)
6      1     355      5       1996       Flintstones, The (1994)
                         genres release_year
1                Comedy|Romance         1992
2         Action|Crime|Thriller         1995
3  Action|Drama|Sci-Fi|Thriller         1995
4       Action|Adventure|Sci-Fi         1994
5 Action|Adventure|Drama|Sci-Fi         1994
6       Children|Comedy|Fantasy         1994
```

```
  userId movieId rating year_rated
1      1     231      5       1996
2      1     480      5       1996
3      1     586      5       1996
4      2     151      3       1997
5      2     858      2       1997
6      2    1544      3       1997
                                                   title
1                                    Dumb & Dumber (1994)
2                                    Jurassic Park (1993)
3                                       Home Alone (1990)
4                                          Rob Roy (1995)
5                                   Godfather, The (1972)
6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
                                      genres release_year
1                                     Comedy         1994
2        Action|Adventure|Sci-Fi|Thriller         1993
3                            Children|Comedy         1990
4                  Action|Drama|Romance|War         1995
5                                Crime|Drama         1972
6 Action|Adventure|Horror|Sci-Fi|Thriller         1997
```

Display the distinct number of users and distinct number of movies in our train set

```
  distinct_users distinct_movies
1          69878           10677
```

We create and display a new df with useful metrics in order to understand better our dataset and identify outliers. Packages used (tidyr),(dplyr)

```
# A tibble: 20 x 5
   genres Ratings_perGenr~ Ratings_perGenr~ Movies_perGenre~
   <chr>             <int>            <dbl>            <int>
 1 (no g~                7             3.64                1
 2 Action          2560545             3.42             1473
 3 Adven~          1908892             3.49             1025
 4 Anima~           467168             3.60              286
 5 Child~           737994             3.42              528
 6 Comedy          3540930             3.44             3703
 7 Crime           1327715             3.67             1117
 8 Docum~            93066             3.78              481
 9 Drama           3910127             3.67             5336
10 Fanta~           925637             3.50              543
11 Film-~           118541             4.01              148
12 Horror           691485             3.27             1013
13 IMAX               8181             3.77               29
14 Music~           433080             3.56              436
15 Myste~           568332             3.68              509
16 Roman~          1712100             3.55             1685
17 Sci-Fi          1341183             3.40              754
18 Thril~          2325899             3.51             1705
19 War              511147             3.78              510
20 Weste~           189394             3.56              275
# ... with 1 more variable: Users_perGenre_Sum <int>
```

We notice that the rating mean is not rounded so we will fix it. Also we identify in our new edx movies metrics df that there is one movie without genres.

We will treat it as an outlier and delete it from all our datasets, since it doesnt add any value. We also have 19 distinct genres.

Display the genres with the most movies *(not distinct movies)*

```
# A tibble: 20 x 5
   genres Ratings_perGenr~ Ratings_perGenr~ Movies_perGenre~
   <chr>             <int>            <dbl>            <int>
 1 Drama           3910127             3.67             5336
 2 Comedy          3540930             3.44             3703
 3 Thril~          2325899             3.51             1705
 4 Roman~          1712100             3.55             1685
```

```
 5 Action          2560545            3.42              1473
 6 Crime           1327715            3.67              1117
 7 Adven~          1908892            3.49              1025
 8 Horror           691485            3.27              1013
 9 Sci-Fi          1341183            3.4                754
10 Fanta~           925637            3.5                543
11 Child~           737994            3.42               528
12 War              511147            3.78               510
13 Myste~           568332            3.68               509
14 Docum~            93066            3.78               481
15 Music~           433080            3.56               436
16 Anima~           467168            3.6                286
17 Weste~           189394            3.56               275
18 Film-~           118541            4.01               148
19 IMAX               8181            3.77                29
20 (no g~                7            3.64                 1
# ... with 1 more variable: Users_perGenre_Sum <int>
```

We can observe that most movies are in the above genres

(Reminder) those are not distinct movies. Because as we observed earlier one movie might belong to more than one genre

Display of the genres - with the most distinct ratings

```
# A tibble: 20 x 5
   genres Ratings_perGenr~ Ratings_perGenr~ Movies_perGenre~
   <chr>             <int>            <dbl>            <int>
 1 Drama           3910127             3.67             5336
 2 Comedy          3540930             3.44             3703
 3 Action          2560545             3.42             1473
 4 Thril~          2325899             3.51             1705
 5 Adven~          1908892             3.49             1025
 6 Roman~          1712100             3.55             1685
 7 Sci-Fi          1341183             3.4               754
 8 Crime           1327715             3.67             1117
 9 Fanta~           925637             3.5               543
10 Child~           737994             3.42              528
11 Horror           691485             3.27             1013
12 Myste~           568332             3.68              509
13 War              511147             3.78              510
14 Anima~           467168             3.6               286
15 Music~           433080             3.56              436
16 Weste~           189394             3.56              275
17 Film-~           118541             4.01              148
```

```
18 Docum~          93066              3.78              481
19 IMAX             8181              3.77               29
20 (no g~              7              3.64                1
# ... with 1 more variable: Users_perGenre_Sum <int>
```

Here we observed that the top 3 genres with the most ratings are

-Drama -Comedy and -Action

Some genres have exponential low sum of ratings so probably they will be also treated as outliers in the data frame that we will create with all genres as factors

Display of ratings mean - per genre.

```
# A tibble: 20 x 5
   genres Ratings_perGenr~ Ratings_perGenr~ Movies_perGenre~
   <chr>             <int>            <dbl>            <int>
 1 Film-~           118541             4.01              148
 2 Docum~            93066             3.78              481
 3 War              511147             3.78              510
 4 IMAX               8181             3.77               29
 5 Myste~           568332             3.68              509
 6 Crime           1327715             3.67             1117
 7 Drama           3910127             3.67             5336
 8 (no g~                7             3.64                1
 9 Anima~           467168             3.6               286
10 Music~           433080             3.56              436
11 Weste~           189394             3.56              275
12 Roman~          1712100             3.55             1685
13 Thril~          2325899             3.51             1705
14 Fanta~           925637             3.5               543
15 Adven~          1908892             3.49             1025
16 Comedy          3540930             3.44             3703
17 Action          2560545             3.42             1473
18 Child~           737994             3.42              528
19 Sci-Fi          1341183             3.4               754
20 Horror           691485             3.27             1013
# ... with 1 more variable: Users_perGenre_Sum <int>
```

Here we can observe that genres with low sum of ratings have higher rating mean. This is one more indicator that should be treated as outliers.

Also movies with low sum of ratings will be removed from the training set for the same reasons

We create and display the same df with the metrics for the validation dataset (test data) to analyse if it is "representative" of our training data

```
# A tibble: 19 x 5
   genres Ratings_perGenr~ Ratings_perGenr~ Movies_perGenre~
   <chr>             <int>            <dbl>            <int>
 1 Drama            434071             3.67             4835
 2 Comedy           393138             3.44             3468
 3 Thril~           258536             3.5              1615
 4 Roman~           189783             3.55             1586
 5 Action           284804             3.42             1404
 6 Crime            147242             3.66             1044
 7 Adven~           212182             3.49              974
 8 Horror            76740             3.26              949
 9 Sci-Fi           149306             3.4               713
10 Fanta~           102845             3.5               523
11 Child~            82155             3.42              507
12 Myste~            62612             3.68              481
13 War               56916             3.77              453
14 Music~            48094             3.56              407
15 Docum~            10388             3.78              406
16 Anima~            51944             3.59              275
17 Weste~            21065             3.55              244
18 Film-~            13051             4.02              137
19 IMAX                899             3.74               27
# ... with 1 more variable: Users_perGenre_Sum <int>
```

```
[1] "We  see that our validation set is representative to our training set."
```

Display of the ratings distribution

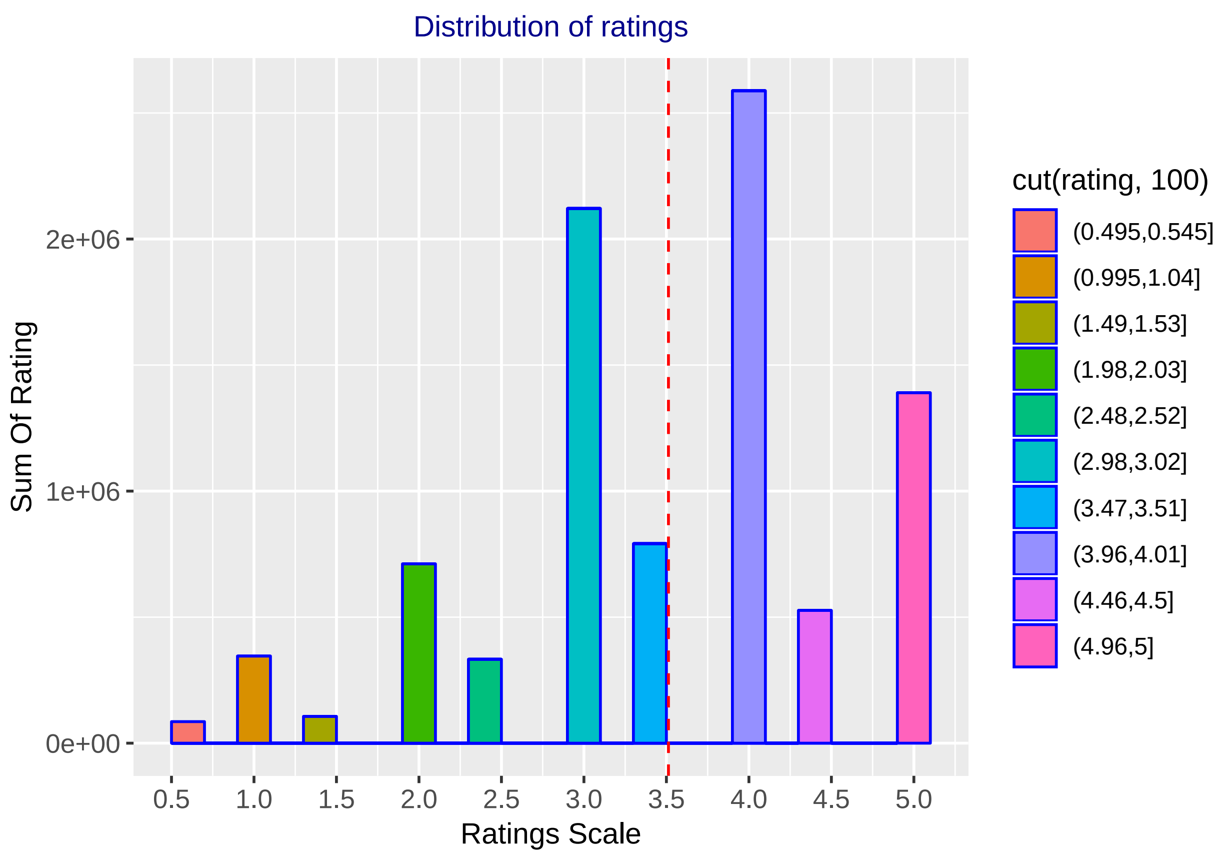
```
# A tibble: 10 x 2
   rating ratings_distribution_sum
    <dbl>                    <int>
 1    4                    2588430
 2    3                    2121240
 3    5                    1390114
 4    3.5                   791624
 5    2                     711422
 6    4.5                   526736
 7    1                     345679
 8    2.5                   333010
 9    1.5                   106426
10    0.5                    85374
```

We observe that the highest sum of ratings are on rating 4. So audience tend not to rate very strict. We also noticed that there is not a movie with rating 0

rating	**ratings_distribution_sum**
4.0	**2588430**
3.0	**2121240**
5.0	**1390114**
3.5	**791624**
2.0	**711422**
4.5	**526736**
1.0	**345679**
2.5	**333010**
1.5	**106426**
0.5	**85374**

Note: Ratings distribution

Histogram plot of the ratings distribution.

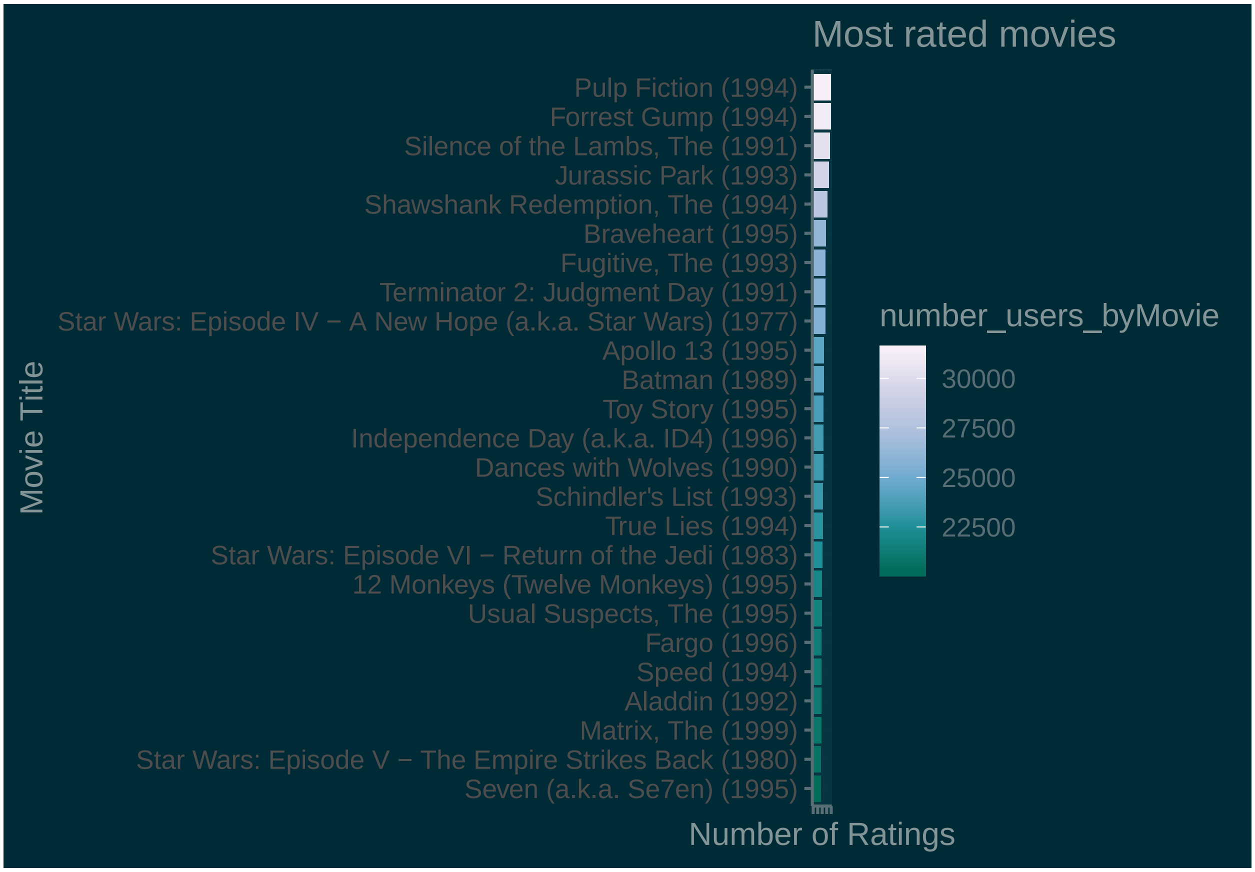


For the training of our ML algorithms we want to **penalized movies rated by low number of users.So in order to put more weight on movies that have been rated by more people, we will add 2 more features in our data sets.**

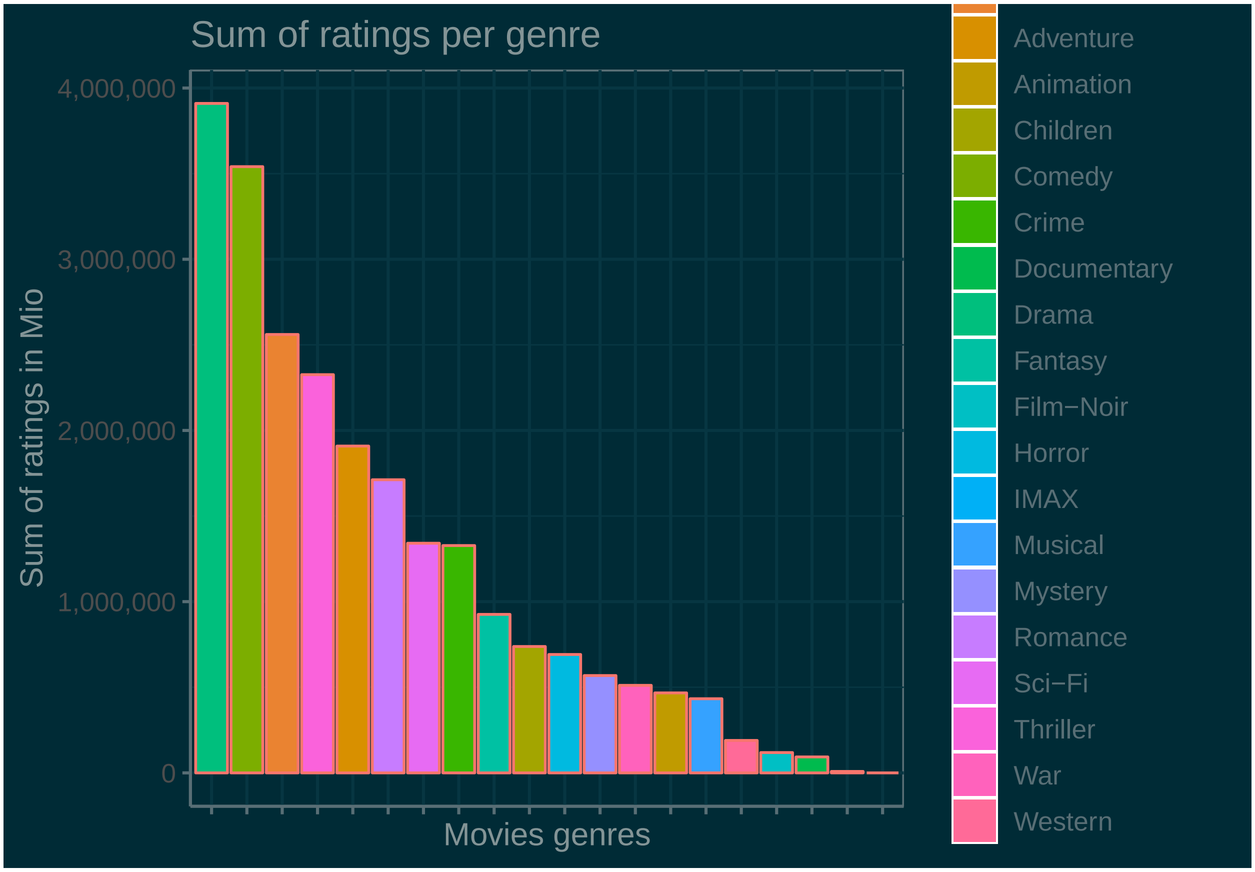
-Number of users per movie and
-Number of movies per user(how many movies had that user rated). So movies and users that have not many rates will be penalized during the ML training

Plot of the most rated movies

(Only those that have been rated over 20.000 times)



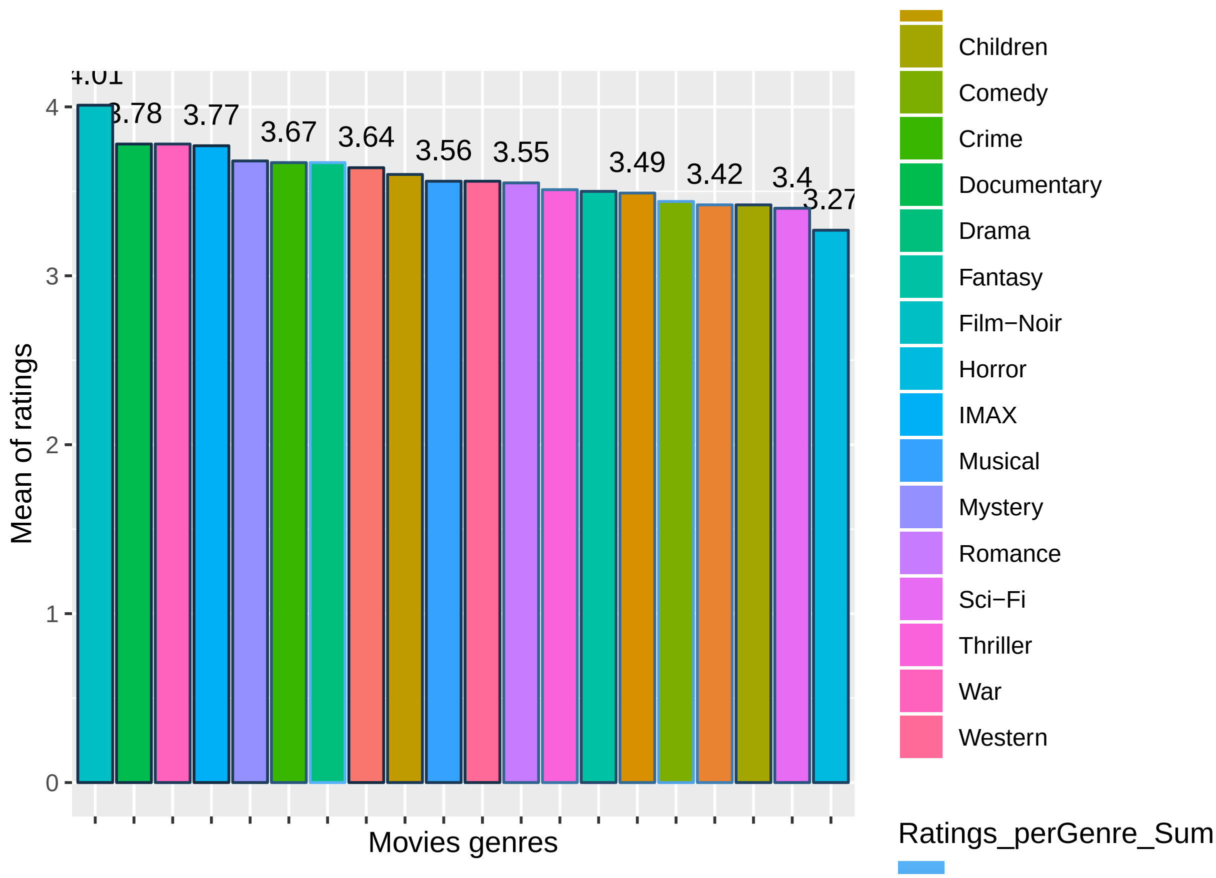
Bar graph display: in order to understand the distribution of the sum of ratings per genres,and movies per genres



In the bar graph is also clear that the **distribution is not equal in all genres. After the fantasy genre, we can see the exponential growth in the number of ratings. That is an indication that probably we wil penalize the genres with low number of ratings, in our model**

Word count cloud plot - with the most rated genres

Bar graph plot to observe - the distribution of ratings mean - per genres




```
[1] 9
```

```
[1] 9
```

Also instead of having the year released we will create a new feature with how old is every movie and drop the year released.

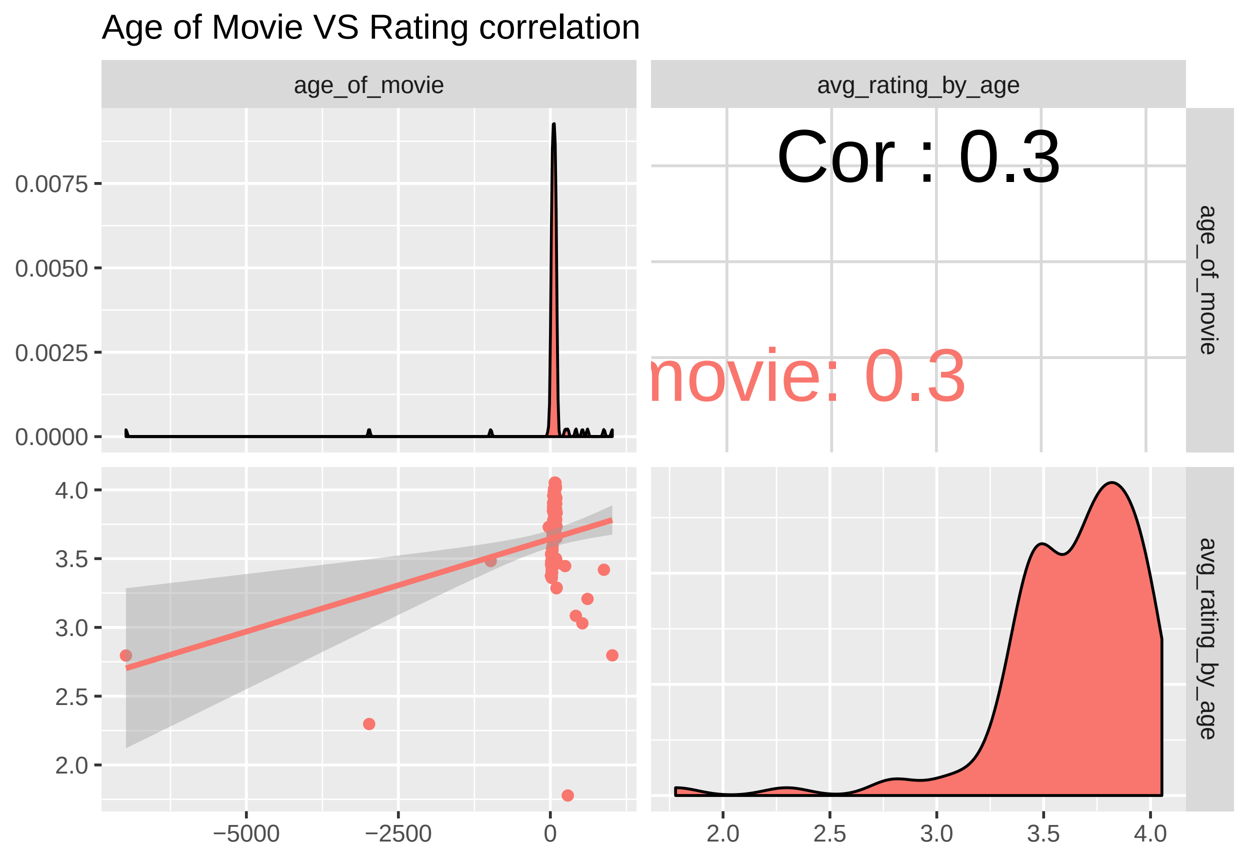
We created a new df with(more dimensions) all the genres extracted and displayed as factors in order to use more features and improve our models if needed.

```
# A tibble: 6 x 30
# Groups:   movieId [6]
  userId movieId rating year_rated title genres release_year
   <int>   <dbl>  <dbl> <chr>      <chr> <chr>         <dbl>
1      1     122      5 1996       Boom~ Comed~         1992
2      1     185      5 1996       Net,~ Actio~         1995
3      1     292      5 1996       Outb~ Actio~         1995
4      1     316      5 1996       Star~ Actio~         1994
5      1     329      5 1996       Star~ Actio~         1994
6      1     355      5 1996       Flin~ Child~         1994
# ... with 23 more variables: number_movies_byUser <int>,
#   number_users_byMovie <int>, Comedy <dbl>, Romance <dbl>, Action <dbl>,
#   Crime <dbl>, Thriller <dbl>, Drama <dbl>, `Sci-Fi` <dbl>,
#   Adventure <dbl>, Children <dbl>, Fantasy <dbl>, War <dbl>,
#   Animation <dbl>, Musical <dbl>, Western <dbl>, Mystery <dbl>,
#   `Film-Noir` <dbl>, Horror <dbl>, Documentary <dbl>, IMAX <dbl>, `(no
#   genres listed)` <dbl>, age_of_movie <dbl>
```

We noticed a column with no genres that contains only one movie in edx matrix, we will delete that column (outlier) also we will delete the columns (genres) with low sum of ratings. The reasons for this is because the train set is already big 9000047 rows so we dont want to have many dimensions during the models building. The second reason is to prevent overfidding

In our data sets the variation of the age of the movies starts from 11 years (that means 2008- this is the last year we have movies until 104 then are the oldest movies we have).
We will check if there is a correlation between age of movie and ratings. First we will create a new object,in order to observe easier, and also plot faster and with less memory!

We will examine if there is a correlation between age of movie and rating. We can observe on the graph the negative skewness

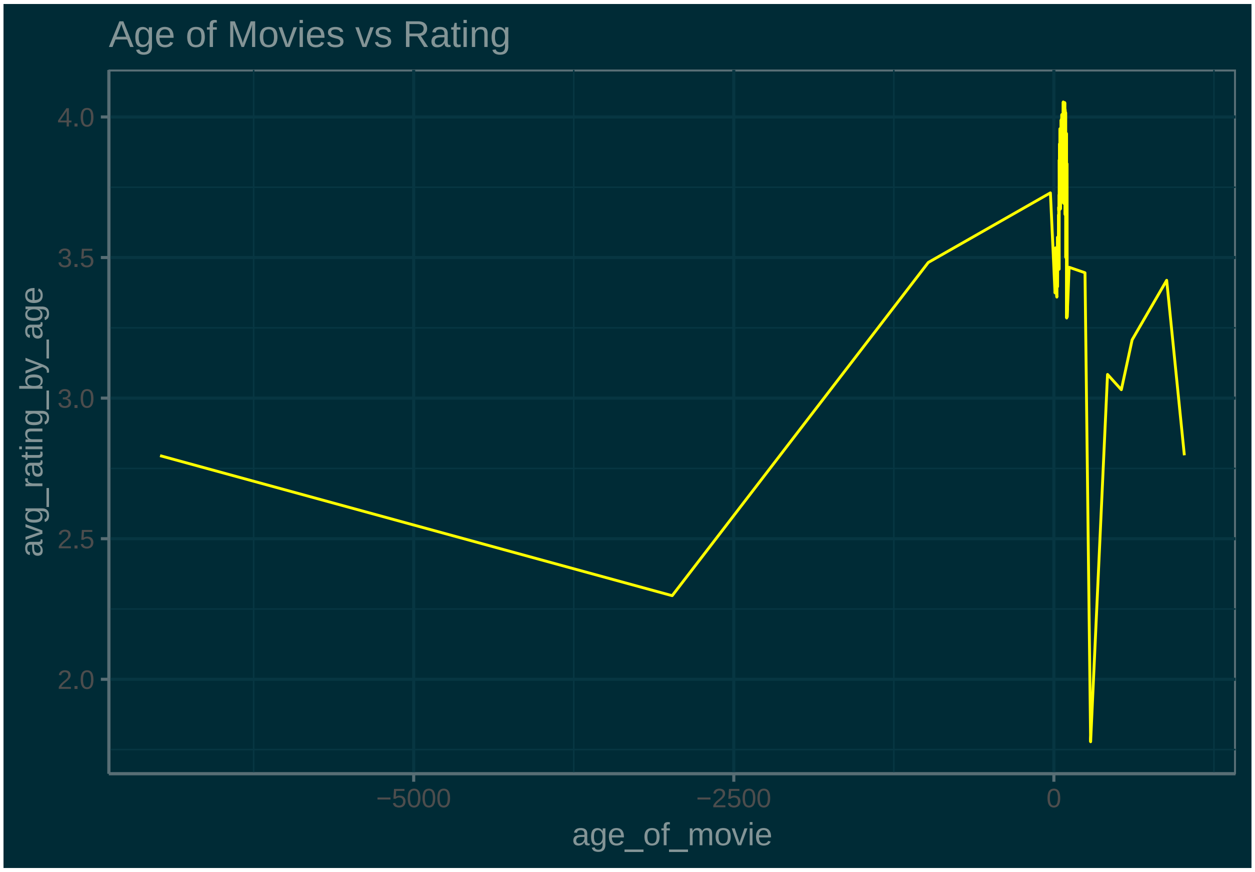


We can clearly notice that there is a positive trend. The oldest the movie the highest the ratings it receives.

This is due to 2 reasons.

-First the oldest the movie the more ratings it has. -Second usually the old movies are consider classics and they are rated better by the audience

It looks that the **age of movie will have low p value in or ML models training.** We create one more plot to demonstrate it



Check if there is a correlation between the year that the film was rated and the rating. The films rated year varies from 1995 until 2009.

```
# A tibble: 15 x 2
   year_rated avg_rating
   <chr>           <dbl>
 1 1995             4
 2 1996             3.55
 3 1997             3.59
 4 1998             3.51
 5 1999             3.62
 6 2000             3.58
 7 2001             3.54
 8 2002             3.47
 9 2003             3.47
10 2004             3.43
11 2005             3.44
```

```
12 2006                    3.47
13 2007                    3.47
14 2008                    3.54
15 2009                    3.46
```

We notice that the oldest the movie was rated (1995) the highest was the rating. Same correlation as the age of the movie

Model Building - Training and Validation

Matrix Factorization with parallel stochastic gradient descent

As we mentioned earliaer. There are 2 types of recommender systems: Content filtering (based on the description of the item - also called meta data or side information)

And collaborative Filtering: Those techniques are calculating the similarity measures of the target ITEMS and finding the minimum (Euclidean distance, or Cosine distance, or other metric, it depends on the algorithm). This is done by filtering the interests of a user, by collecting preferences from many users (collaborating). The underlying assumption is that if a person X has the same opinion as a person Y then the recommendation system should be based on preferences of person Y (similarity).

We will enhance the collaborative filtering with the help of Matrix factorization. MF is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.This family of methods became widely known during the Netflix prize challenge due to its effectiveness as reported by Simon Funk in his 2006 blog post, where he shared his findings with the research community [link](https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)

We will apply Matrix Factorization with parallel stochastic gradient descent. With the help of "recosystem" package it is an R wrapper of the LIBMF library which creates a Recommender System by Using Parallel Matrix Factorization. The main task of recommender system is to predict unknown entries in the rating matrix based on observed values.

More info on the recosystem package and the techniques link Before we proceed with the model building,training and validation we define the RMSE function

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 in order to use recosystem package we create 2 new matrices (our train and our validation set) with the below 3 features
-(movieId,userId,rating)

Recosystem needs to save the files as tables on hard disk, (recosystem package needed).

We reate a model object (a Reference Class object in R) by calling the function Reco()

This step is optional. We call the $tune() method to select the best tuning parameters (along a set of candidate values).

Display of the tuning

```
$min
$min$dim
[1] 30

$min$costp_l1
[1] 0

$min$costp_l2
[1] 0.1

$min$costq_l1
[1] 0

$min$costq_l2
[1] 0.01

$min$lrate
[1] 0.1

$min$loss_fun
[1] 0.7974779


$res
   dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
1   10        0     0.01        0     0.01   0.1 0.8246018
2   20        0     0.01        0     0.01   0.1 0.8083978
3   30        0     0.01        0     0.01   0.1 0.8148374
4   10        0     0.10        0     0.01   0.1 0.8280235
5   20        0     0.10        0     0.01   0.1 0.8040914
6   30        0     0.10        0     0.01   0.1 0.7974779
7   10        0     0.01        0     0.10   0.1 0.8269870
8   20        0     0.01        0     0.10   0.1 0.8023862
9   30        0     0.01        0     0.10   0.1 0.8005307
10  10        0     0.10        0     0.10   0.1 0.8387589
11  20        0     0.10        0     0.10   0.1 0.8295851
12  30        0     0.10        0     0.10   0.1 0.8289237
13  10        0     0.01        0     0.01   0.2 0.8631933
14  20        0     0.01        0     0.01   0.2 0.9152933
15  30        0     0.01        0     0.01   0.2 0.9257084
16  10        0     0.10        0     0.01   0.2 0.8216410
17  20        0     0.10        0     0.01   0.2 0.8000816
18  30        0     0.10        0     0.01   0.2 0.8003977
```

19	10	0	0.01	0	0.10	0.2	0.8156165
20	20	0	0.01	0	0.10	0.2	0.9598589
21	30	0	0.01	0	0.10	0.2	0.8634026
22	10	0	0.10	0	0.10	0.2	0.8398871
23	20	0	0.10	0	0.10	0.2	0.8267008
24	30	0	0.10	0	0.10	0.2	0.8262832

Now we train the model by calling the $train() method. A number of parameters can be set inside the function, coming from the result of the previous step - $tune().

With the $predict() method we will make predictions on validation set and will calculate RMSE:

Mean squared error (abbreviated MSE) and ** root mean square error (RMSE)** refer to the amount by which the values predicted by an estimator differ from the quantities being estimated (typically outside the sample from which the model was estimated).

We calculate the **standard deviation of the residuals (prediction errors) RMSE** . Between the predicted
ratings and the real ratings . If one or more predictors are significant, the second step is to assess how well the model fits the data by inspecting the **Residuals Standard Error (RSE).**

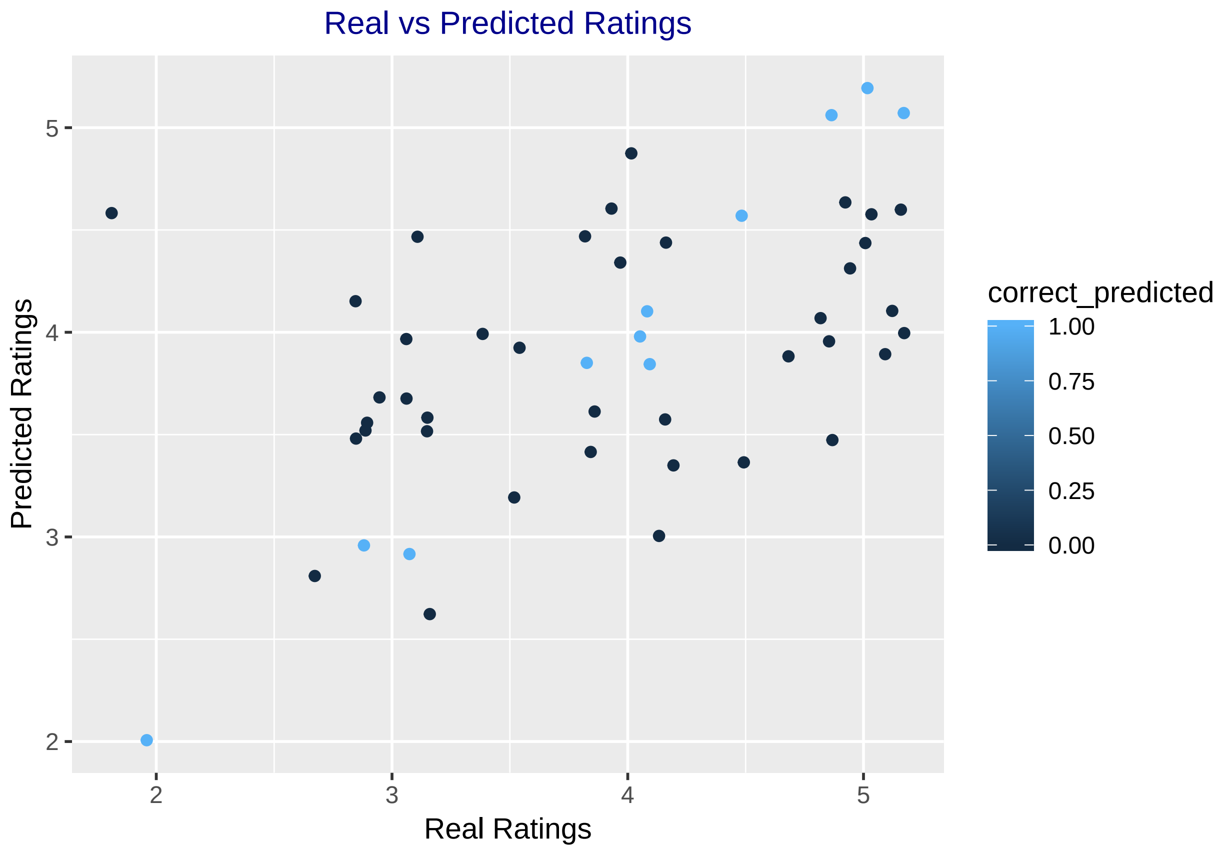
Root mean squared error of the Matrix Factorization model

```
[1] 0.7829978
```

We see that the RMSE is extremely low. And possibly until know the Matrix factorization with SGD is the best approach to create a recommender system. I would like to thank Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang for creating the LIMBF library but also Yixuan Qiu that created the R wrapper. link

We will compare the first 50 predictions of the MF model with the real ratings. First we round the predictions for visualization convenience

Plot - with the 50 first predicted ratings of the MF model.The light blue are the correct predictions



real_ratings	**predicted_ratings**	correct_predicted
5.0	**4.0**	0
5.0	**5.0**	1
5.0	**5.0**	1
3.0	**3.5**	0
2.0	**4.5**	0
3.0	**2.5**	0
3.5	**4.0**	0
4.5	**4.5**	1
5.0	**4.5**	0
3.0	**3.5**	0
3.0	**3.5**	0
3.0	**3.5**	0
3.0	**3.5**	0
3.0	**4.0**	0
3.0	**3.5**	0
3.0	**4.5**	0
3.0	**4.0**	0
3.0	**3.0**	1
4.0	**4.0**	1
5.0	**4.0**	0
3.0	**3.5**	0
4.0	**5.0**	0
4.0	**4.5**	0
4.0	**4.5**	0
5.0	**5.0**	1
4.0	**3.5**	0
2.0	**2.0**	1
5.0	**4.5**	0
5.0	**4.5**	0
5.0	**4.5**	0
4.0	**4.0**	1
3.0	**3.0**	1
4.0	**4.5**	0
4.0	**3.5**	0
5.0	**4.5**	0
5.0	**4.0**	0
5.0	**4.0**	0
4.0	**3.0**	0
4.0	**4.0**	1
4.0	**4.5**	0
4.0	**4.0**	1
5.0	**4.0**	0
3.5	**3.0**	0
5.0	**3.5**	0
4.0	**3.5**	0
4.5	**4.0**	0
2.5	**3.0**	0
4.5	**3.5**	0

Root Mean Squared error of the Matrix factorization with parallel stochastic gradient descent

Algorithm	**RMSE**
Matrix factorization with SGD	**0.7829978**

Note: RMSE OF ALL MODELS

H2o Machine learning training, build and validating part

H2o open source machine learning and artificial intelligence platform

Create factors because in h2o the dependent variables need to be factors

```
require(h2o)

h2o.init(ignore_config=TRUE,
         nthreads=-1,
         max_mem_size = "20G")
```

```
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         3 seconds 324 milliseconds
    H2O cluster timezone:       Europe/Berlin
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.22.1.1
    H2O cluster version age:    4 months and 5 days !!!
    H2O cluster name:           H2O_started_from_R_npapaco_jms911
    H2O cluster total nodes:    1
    H2O cluster total memory:   10.64 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  8
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         Algos, AutoML, Core V3, Core V4
    R Version:                  R version 3.5.3 (2019-03-11)
```

After the start of the cluster optionally we can access it from browser to http://localhost:54321 I recommend to try it. Play also with pojo saving of models

We start with h2o automl - H2O's AutoML can be used for automating the machine learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit. Stacked Ensembles - one based on all previously trained models, another one on the best model of each family - will be automatically trained on collections of individual models to produce highly predictive ensemble models which, in most cases, will be the top performing models in the AutoML Leaderboard.

link

H2O AutoML

H2O's AutoML can be used for **automating the machine learning workflow** , which includes **automatic training and tuning** of many models within a user-specified time-limit. Stacked Ensembles - one based on all previously trained models, another one on the best model of each family - will be automatically trained on collections of individual models to produce highly predictive ensemble models which, in most cases, will be the **top performing models in the AutoML Leaderboard.**

link

```
h2oamlmodel2 <- h2o.automl ( x = c("movieId","n.users_bymovie","age_of_movie","Drama","C
                             y = "rating" ,
                             training_frame = train3,
                             leaderboard_frame = test_validation,
                             project_name = "movielens",
                             max_models = 20,
                             max_runtime_secs = 3600,
                             nfolds = 3,
                             keep_cross_validation_predictions= TRUE,
                             seed = 1)
```

The model with the least RMSE in the leaderboard

```
Model Details:
==============

H2ORegressionModel: drf
Model ID:  DRF_1_AutoML_20190424_155200
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              50                       50              185395         1
  max_depth mean_depth min_leaves max_leaves mean_leaves
1        20   14.40000          2        680   160.96000


H2ORegressionMetrics: drf
** Reported on training data. **
** Metrics reported on Out-Of-Bag training samples **

MSE:  1.01896
RMSE:  1.009436
MAE:  0.8091325
RMSLE:  0.2679033
Mean Residual Deviance :  1.01896



H2ORegressionMetrics: drf
** Reported on cross-validation data. **
** 3-fold cross-validation on training data (Metrics computed for combined holdout predi

MSE:  1.017104
RMSE:  1.008516
MAE:  0.8084183
RMSLE:  0.2677314
Mean Residual Deviance :  1.017104


Cross-Validation Metrics Summary:
                              mean           sd cv_1_valid cv_2_valid
mae                      0.8084183   2.020126E-5  0.8083981 0.80839807
mean_residual_deviance   1.0171036 1.16297306E-4   1.016921  1.0173197
mse                      1.0171036 1.16297306E-4   1.016921  1.0173197
r2                     0.095347136   8.510263E-5 0.09520545  0.0953363
residual_deviance        1.0171036 1.16297306E-4   1.016921  1.0173197
rmse                     1.0085155  5.7656973E-5   1.008425  1.0086226
```

```
rmsle                     0.2677314   3.339011E-5 0.26766708  0.2677791
                        cv_3_valid
mae                      0.8084587
mean_residual_deviance  1.0170699
mse                      1.0170699
r2                      0.09549966
residual_deviance        1.0170699
rmse                     1.0084988
rmsle                   0.26774806
```

we see the model that had the lowest RMSE on the leaderboard .The (leader) of all models tested was the below.

-Model ID: DRF_1_AutoML_20190424_155200.

-Algorithm: Algorithm: Distributed Random Forest

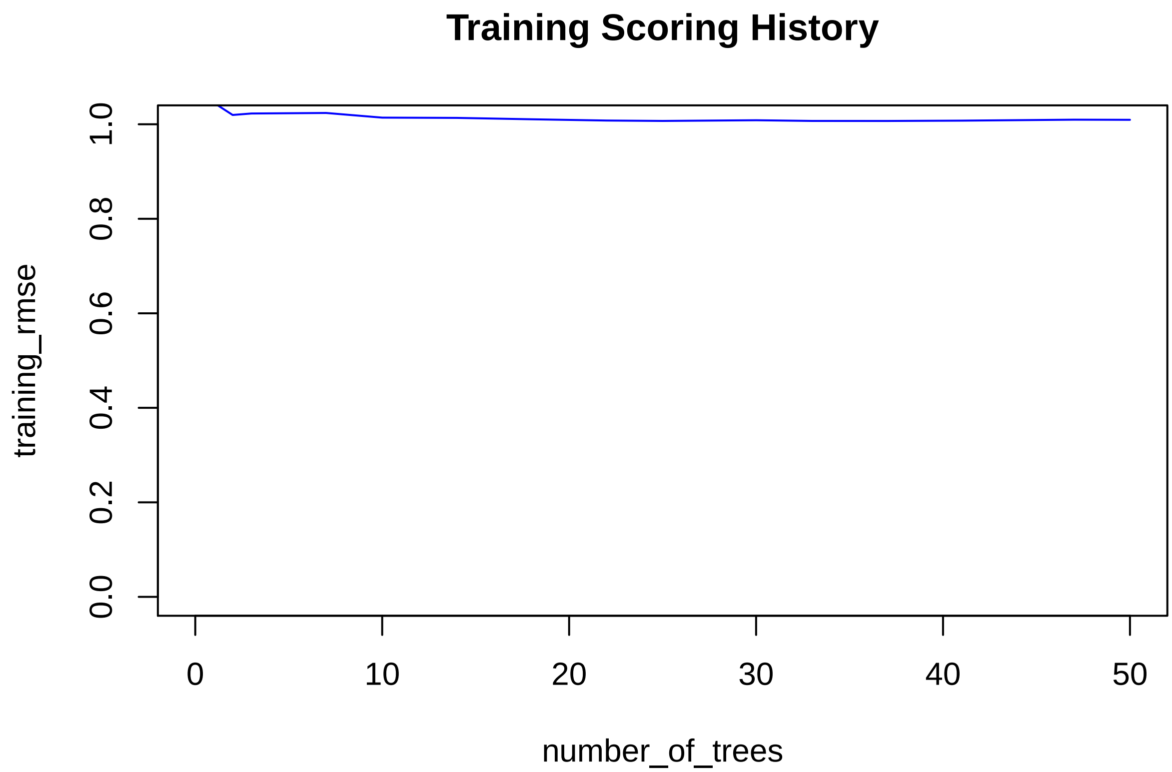
Display of the 6 best models from the leaderboard

```
                                   model_id mean_residual_deviance
1               DRF_1_AutoML_20190424_155200               1.077967
2               XRT_1_AutoML_20190424_155200               1.088622
3 GLM_grid_1_AutoML_20190424_155200_model_1               1.099932
4               GBM_4_AutoML_20190424_155200               1.112490
5               GBM_2_AutoML_20190424_155200               1.116864
6 GLM_grid_1_AutoML_20190424_142708_model_1               1.117170
      rmse      mse       mae     rmsle
1 1.038252 1.077967 0.8410296 0.2702502
2 1.043370 1.088622 0.8475965 0.2717722
3 1.048777 1.099932 0.8462009 0.2689410
4 1.054746 1.112490 0.8592683 0.2673023
5 1.056818 1.116864 0.8620348 0.2674363
6 1.056963 1.117170 0.8571898 0.2725451

[15 rows x 6 columns]
```

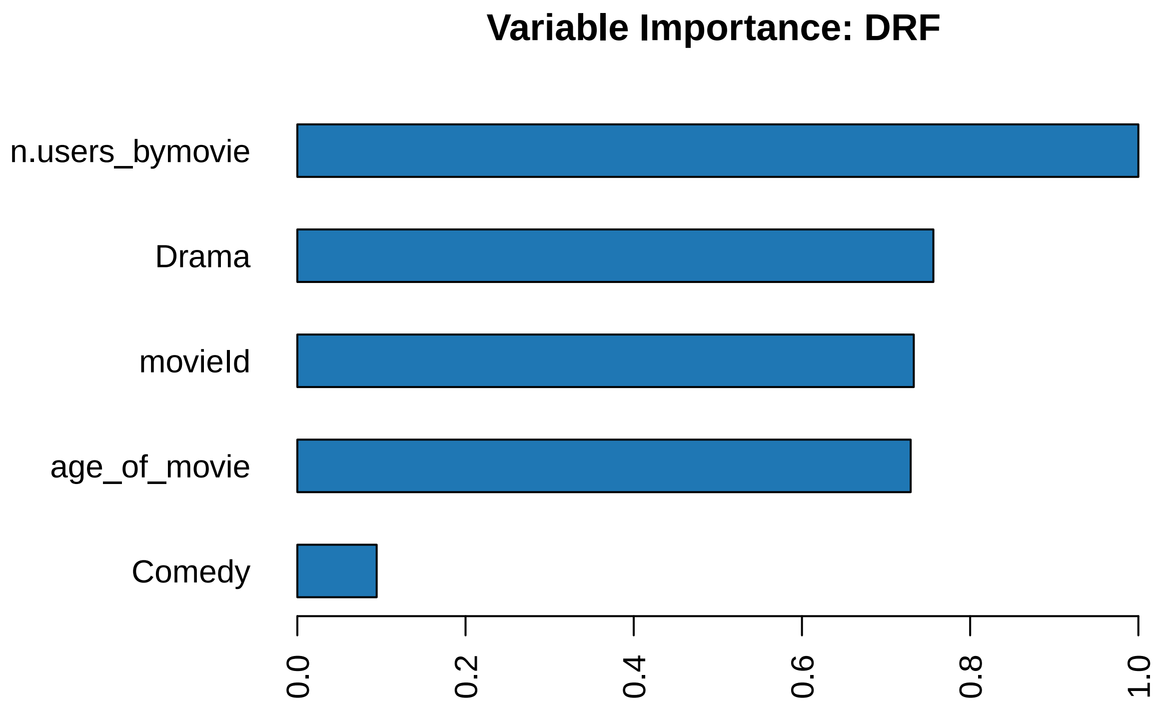
OUTPUT - VARIABLE IMPORTANCES
-variable relative_importance scaled_importance percentage
-n.users_bymovie 7164049.0 1.0 0.3019
-Drama 5418105.0 0.7563 0.2283
-movieId 5250999.0 0.7330 0.2212
-age_of_movie 5224642.5000 0.7293 0.2201
-Comedy 675853.8125 0.0943 0.0285

```
[1] "Print of the scoring history"

Scoring History:
             timestamp          duration number_of_trees training_rmse
1  2019-04-24 15:56:44  4 min 43.962 sec               0            NA
2  2019-04-24 15:56:45  4 min 44.868 sec               1       1.04531
3  2019-04-24 15:56:46  4 min 46.496 sec               2       1.01972
4  2019-04-24 15:56:48  4 min 47.738 sec               3       1.02279
5  2019-04-24 15:56:52  4 min 52.322 sec               7       1.02390
6  2019-04-24 15:56:57  4 min 56.892 sec              10       1.01407
7  2019-04-24 15:57:01  5 min  1.097 sec              14       1.01350
8  2019-04-24 15:57:06  5 min  5.648 sec              18       1.01059
9  2019-04-24 15:57:11  5 min 10.711 sec              22       1.00790
10 2019-04-24 15:57:15  5 min 15.087 sec              25       1.00708
11 2019-04-24 15:57:19  5 min 19.196 sec              30       1.00855
12 2019-04-24 15:57:24  5 min 23.735 sec              33       1.00705
13 2019-04-24 15:57:28  5 min 27.811 sec              37       1.00703
```

```
14 2019-04-24 15:57:32  5 min 32.253 sec              41        1.00771
15 2019-04-24 15:57:37  5 min 37.472 sec              47        1.00971
16 2019-04-24 15:57:42  5 min 41.837 sec              50        1.00944
   training_mae training_deviance
1            NA                NA
2       0.84292           1.09268
3       0.81434           1.03983
4       0.81802           1.04610
5       0.82144           1.04838
6       0.81201           1.02834
7       0.81177           1.02718
8       0.80910           1.02129
9       0.80635           1.01587
10      0.80546           1.01422
11      0.80753           1.01717
12      0.80580           1.01414
13      0.80591           1.01412
14      0.80671           1.01548
15      0.80936           1.01952
16      0.80913           1.01896

[1] "Plot of the training history, metric (rmse)"
```




```
[1] "Plot of the variables Importance (by order)"
```



Root mean squared error of h2oamlmodel2

```
[1] 1.038252
```

RMSE of MF algorithm

Algorithm	RMSE
Matrix factorization with SGD	0.7829978
H2o Auto ML model2	1.0382518

H2o Generalized Linear Models (GLM)

Because on autoML we cant tune many hyperparameters, we will also try on other models with various hyper tunings.Then **we will stack those different models stacked ensemble**.

Ensemble machine learning methods use**multiple learning algorithms to obtain better predictive** performance than could be obtained from any of the constituent learning algorithms We start with:

-Generalized Linear Models (GLM) estimate regression models for outcomes following exponential distributions. In addition to the Gaussian (i.e. normal) distribution, these include

-Poisson

-binomial and

-gamma distributions link

```r
h2o_glm <- h2o.glm( x = c("movieId","userId","n.users_bymovie","Drama","age_of_movie","C
                    y = "rating" ,
                    training_frame = train3 ,
                    validation_frame = test3,
                    alpha = 0.5,   #0=ridge , 1 = lasso, so we leave it in the middle t
                    lambda = seq(0, 10, 0.25),
                    nlambdas = 30,
                    seed = 1,
                    keep_cross_validation_predictions = TRUE,
                    nfolds = 3)
```

output - Standardized Coefficient Magnitudes (standardized coefficient magnitudes)
-movieId
-userId
-age_of_movie

```
[1] "summary(h2o_glm)"

Model Details:
==============

H2ORegressionModel: glm
Model Key:  GLM_model_R_1555942382794_38
GLM Model: summary
    family     link                             regularization
1 gaussian identity Elastic Net (alpha = 0.5, lambda = 0.0 )
  number_of_predictors_total number_of_active_predictors
1                      80648                       80578
  number_of_iterations   training_frame
1                   50 RTMP_sid_9155_88

H2ORegressionMetrics: glm
** Reported on training data. **

MSE:  0.8456564
RMSE:  0.9195958
MAE:  0.7172375
RMSLE:  0.2437822
Mean Residual Deviance :  0.8456564
R^2 :  0.2479454
Null Deviance :7084475
Null D.o.F. :6300326
```

```
Residual Deviance :5327912
Residual D.o.F. :6219748
AIC :16984513


H2ORegressionMetrics: glm
** Reported on validation data. **

MSE:  0.8500083
RMSE:  0.9219589
MAE:  0.7187125
RMSLE:  0.2443887
Mean Residual Deviance :  0.8500083
R^2 :  0.243719
Null Deviance :3034302
Null D.o.F. :2699720
Residual Deviance :2294785
Residual D.o.F. :2619142
AIC :7383907


H2ORegressionMetrics: glm
** Reported on cross-validation data. **
** 3-fold cross-validation on training data (Metrics computed for combined holdout predi

MSE:  0.8432479
RMSE:  0.9182853
MAE:  0.7153296
RMSLE:  0.2434671
Mean Residual Deviance :  0.8432479
R^2 :  0.2500874
Null Deviance :7084481
Null D.o.F. :6300326
Residual Deviance :5312737
Residual D.o.F. :6219832
AIC :16966375


Cross-Validation Metrics Summary:
                             mean           sd cv_1_valid cv_2_valid
mae                     0.7153301 8.4168854E-4   0.713684  0.7164582
mean_residual_deviance 0.84324986 0.0015927338  0.8402663  0.8437752
mse                    0.84324986 0.0015927338  0.8402663  0.8437752
null_deviance           2361493.8    1591.0111  2358487.2  2363899.5
r2                      0.2500874 5.5939064E-4 0.25109872 0.24999613
```

```
residual_deviance       1770912.4    2327.2556  1766270.8  1772933.8
rmse                   0.91828555   8.67428E-4 0.91666037 0.91857237
rmsle                  0.24346721 2.9301652E-4 0.24303728 0.24333727
                       cv_3_valid
mae                     0.7158482
mean_residual_deviance 0.84570813
mse                    0.84570813
null_deviance           2362094.2
r2                      0.2491674
residual_deviance       1773532.8
rmse                    0.9196239
rmsle                  0.24402706

Scoring History:
            timestamp   duration iterations negative_log_likelihood
1 2019-04-24 21:36:36  0.000 sec          0           7084475.04044
2 2019-04-24 21:36:36  0.203 sec          1           6791137.50291
3 2019-04-24 21:36:36  0.438 sec          2           6672861.94944
4 2019-04-24 21:36:36  0.653 sec          3           6649442.26266
5 2019-04-24 21:36:37  0.853 sec          4           6569032.76605
  objective
1   1.12446
2   1.07790
3   1.05913
4   1.05541
5   1.04265

---
             timestamp   duration iterations negative_log_likelihood
46 2019-04-24 21:36:58 21.867 sec         45           5394055.33858
47 2019-04-24 21:36:58 22.074 sec         46           5378878.06715
48 2019-04-24 21:36:58 22.274 sec         47           5368760.04431
49 2019-04-24 21:36:58 22.496 sec         48           5351761.62870
50 2019-04-24 21:36:59 23.315 sec         49           5347649.79277
51 2019-04-24 21:36:59 23.522 sec         50           5327911.81262
   objective
46   0.85615
47   0.85375
48   0.85214
49   0.84944
50   0.84879
51   0.84566

Variable Importances: (Extract with `h2o.varimp`)
=================================================
```

```
Standardized Coefficient Magnitudes: standardized coefficient magnitudes
         names coefficients sign
1  movieId.193     1.771044  NEG
2 movieId.3593     1.740096  NEG
3 userId.30272     1.728364  NEG
4  movieId.720     1.684223  POS
5 movieId.2422     1.603026  NEG

---
             names coefficients sign
80644 movieId.7537     0.000000  POS
80645 movieId.7568     0.000000  POS
80646 movieId.7750     0.000000  POS
80647 movieId.8038     0.000000  POS
80648 movieId.8148     0.000000  POS
80649                        NA   NA

[1] "Root mean squared error of h2o_glm"

[1] 1.01631
```

RMSE results of all the models until now

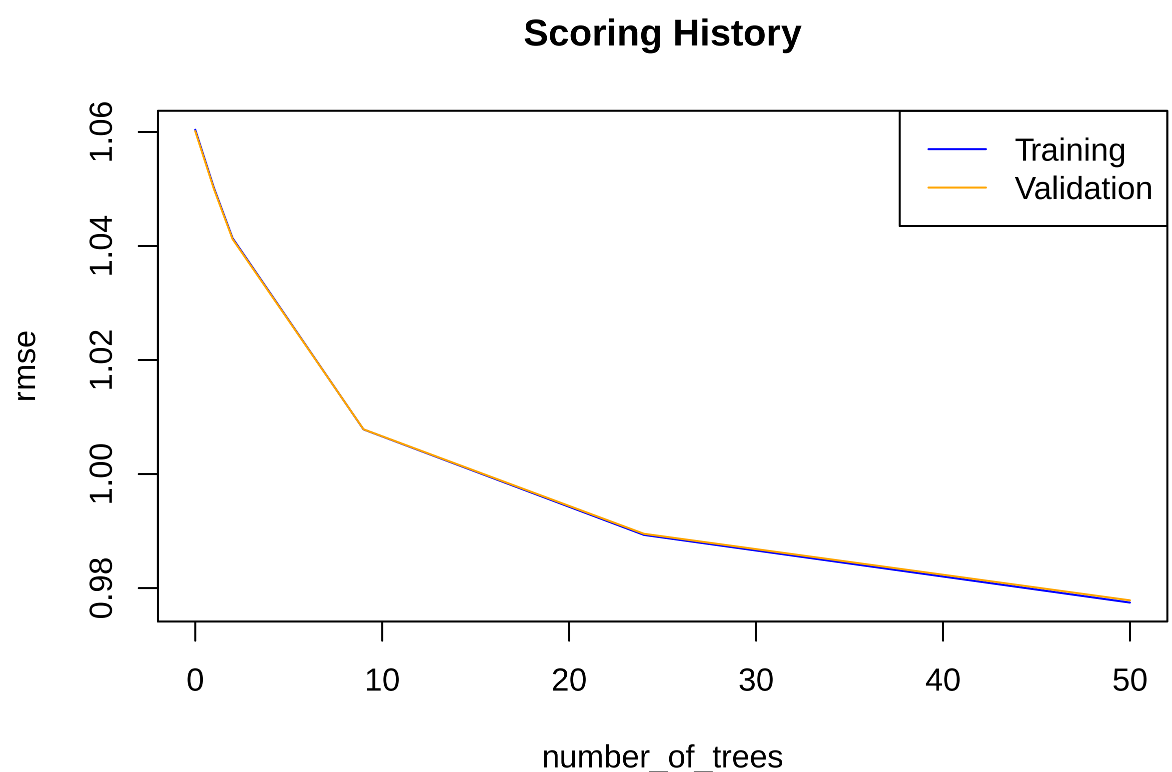
Algorithm	RMSE
Matrix factorization with SGD	0.7829978
H2o Auto ML model	1.0382518
H2o GLM model	1.0163097

H2o - Gradient Boosting Machine model

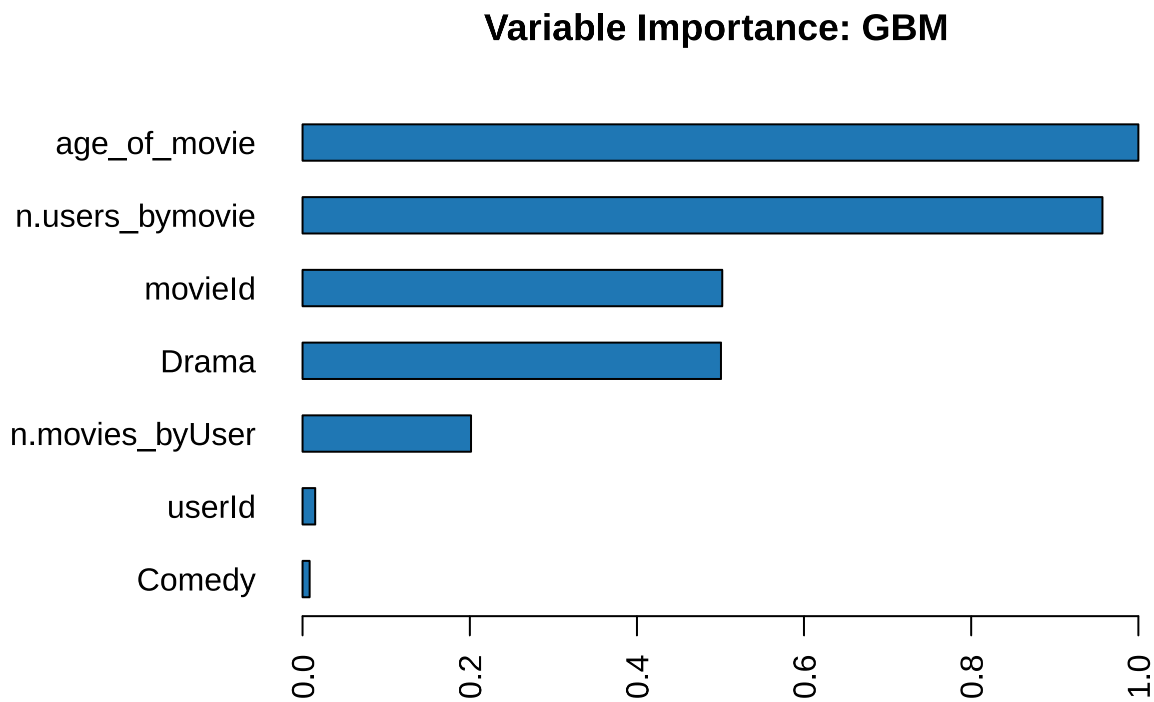
Gradient Boosting Machine (for Regression and Classification) is a forward learning ensemble method. The guiding heuristic is that good predictive results can be obtained through increasingly refined approximations. H2O's GBM sequentially builds regression trees on all the features of the dataset in a fully distributed way - each tree is built in parallel. link

variable relative_importance scaled_importance percentage
-age_of_movie 1759746.6250 1.0 0.3140
-n.users_bymovie 1684104.0 0.9570 0.3005
-movieId 883807.2500 0.5022 0.1577
-Drama 881101.2500 0.5007 0.1572
-n.movies_byUser 354473.7813 0.2014 0.0632
-userId 26781.1113

```
[1] "Plot scoring history"
```




```
[1] "Plot the variables importance"
```




```
[1] "Print the scoring history"

Scoring History:
            timestamp          duration number_of_trees training_rmse
1 2019-04-24 21:30:32  2 min 32.011 sec               0       1.06041
2 2019-04-24 21:30:33  2 min 33.057 sec               1       1.05020
3 2019-04-24 21:30:35  2 min 35.116 sec               2       1.04137
4 2019-04-24 21:30:40  2 min 39.933 sec               9       1.00783
5 2019-04-24 21:30:50  2 min 50.090 sec              24       0.98933
6 2019-04-24 21:31:08  3 min  7.809 sec              50       0.97746
  training_mae training_deviance validation_rmse validation_mae
1      0.85561           1.12446         1.06016        0.85539
2      0.84784           1.10293         1.05000        0.84765
3      0.84121           1.08444         1.04120        0.84105
4      0.80780           1.01572         1.00785        0.80779
5      0.78449           0.97877         0.98954        0.78462
6      0.77237           0.95543         0.97786        0.77263
  validation_deviance
1             1.12393
2             1.10249
3             1.08410
```

```
4              1.01577
5              0.97918
6              0.95621

[1] "Root mean squared error of h2o_gbm_model"

[1] 1.035326
```

RMSE results of all the models until now

Algorithm	RMSE
Matrix factorization with SGD	0.7829978
H2o Auto ML model	1.0382518
H2o GLM model	1.0163097
H2o GBM model	1.0353257

H2o Distributed Random Forest

Distributed Random Forest (DRF) is a powerful classification and regression tool. When given a set of data, DRF generates a forest of classification (or regression) trees, rather than a single classification (or regression) tree. Each of these trees is a weak learner built on a subset of rows and columns

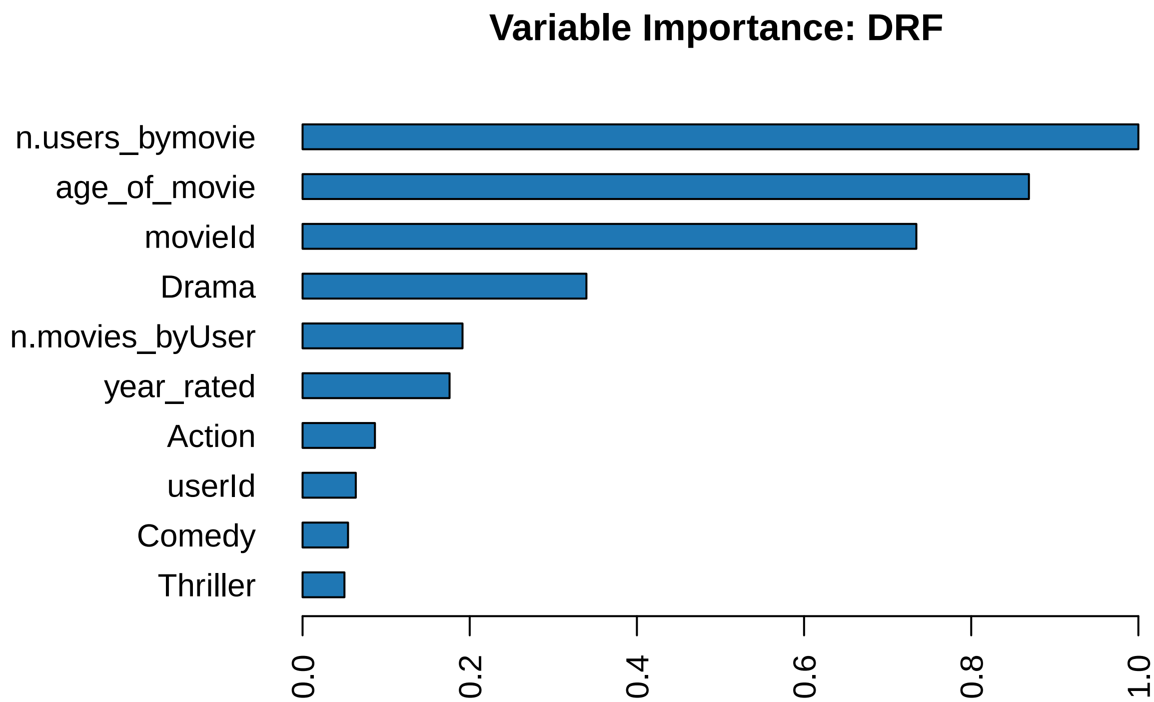
```r
h2orf1 <- h2o.randomForest( x = c("movieId","userId","n.movies_byUser","n.users_bymovie'
                                  "Comedy","age_of_movie","year_rated") ,
                            y = "rating" ,          # Dependent var
                            training_frame = train3,        ## the H2O frame for trainin
                            validation_frame = test3, # the testing frame NOT the real
                            model_id = "h2orf1",    ## name the model ID so you can load
                            ntrees = 50,                  ## numb of  maximum trees to u
                            max_depth = 30,
                            keep_cross_validation_predictions= TRUE, # i recommend to us
                            min_rows = 100, # min rows during training
                            #  You can add the early stopping criteria decide when to s
                            score_each_iteration = T,      ## Predict against training a
                            nfolds = 3,
                            fold_assignment = "AUTO",
                            seed = 1)
```

variable relative_importance scaled_importance percentage
-n.users_bymovie 26496842.0 1.0 0.2790
-age_of_movie 22493868.0 0.8489 0.2369
-movieId 21997428.0 0.8302 0.2317
-Drama 9633707.0 0.3636 0.1015

```
[1] "Summary of h2orf1"

Model Details:
==============

H2ORegressionModel: drf
Model Key:  h2orf1
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              50                       50            15818918        20
  max_depth mean_depth min_leaves max_leaves mean_leaves
1        20   20.00000      17600      22151 19815.38000

H2ORegressionMetrics: drf
```

```
** Reported on training data. **
** Metrics reported on Out-Of-Bag training samples **

MSE:  0.8816749
RMSE:  0.9389755
MAE:  0.7369724
RMSLE:  0.2496137
Mean Residual Deviance :  0.8816749



H2ORegressionMetrics: drf
** Reported on cross-validation data. **
** 3-fold cross-validation on training data (Metrics computed for combined holdout predi

MSE:  0.8846702
RMSE:  0.9405691
MAE:  0.7384566
RMSLE:  0.2501497
Mean Residual Deviance :  0.8846702


Cross-Validation Metrics Summary:
                             mean           sd cv_1_valid cv_2_valid
mae                     0.7384566 2.1804204E-5 0.73846406  0.7384157
mean_residual_deviance 0.88467014 1.0132179E-4  0.8845546  0.8848721
mse                    0.88467014 1.0132179E-4  0.8845546  0.8848721
r2                     0.21324943 4.1463666E-4  0.2139826 0.21254726
residual_deviance      0.88467014 1.0132179E-4  0.8845546  0.8848721
rmse                   0.94056904 5.3860465E-5 0.94050765  0.9406764
rmsle                   0.2501497  3.067497E-5  0.2501173   0.250211
                       cv_3_valid
mae                     0.7384901
mean_residual_deviance  0.8845838
mse                     0.8845838
r2                     0.21321847
residual_deviance       0.8845838
rmse                   0.94052315
rmsle                  0.25012076

Scoring History:
            timestamp          duration number_of_trees training_rmse
1 2019-04-22 20:28:56 25 min 48.392 sec               0            NA
2 2019-04-22 20:29:00 25 min 52.347 sec               1       0.95437
3 2019-04-22 20:29:04 25 min 56.322 sec               2       0.95141
```

```
4 2019-04-22 20:29:08 26 min  0.384 sec               3       0.94985
5 2019-04-22 20:29:12 26 min  4.251 sec               4       0.94858
  training_mae training_deviance
1           NA                NA
2      0.74891           0.91081
3      0.74656           0.90518
4      0.74548           0.90221
5      0.74441           0.89981

---
             timestamp          duration number_of_trees training_rmse
46 2019-04-22 20:31:55 28 min 46.636 sec              45       0.93905
47 2019-04-22 20:31:59 28 min 50.840 sec              46       0.93903
48 2019-04-22 20:32:03 28 min 54.761 sec              47       0.93900
49 2019-04-22 20:32:07 28 min 58.739 sec              48       0.93900
50 2019-04-22 20:32:11 29 min  2.568 sec              49       0.93898
51 2019-04-22 20:32:14 29 min  6.439 sec              50       0.93898
   training_mae training_deviance
46      0.73704           0.88181
47      0.73702           0.88179
48      0.73699           0.88172
49      0.73699           0.88171
50      0.73697           0.88169
51      0.73697           0.88167

Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:
          variable relative_importance scaled_importance percentage
1  n.users_bymovie     14200654.000000          1.000000   0.280495
2     age_of_movie     12341077.000000          0.869050   0.243764
3          movieId     10428516.000000          0.734369   0.205987
4            Drama      4822726.500000          0.339613   0.095260
5  n.movies_byUser      2717909.250000          0.191393   0.053685
6       year_rated      2497641.000000          0.175882   0.049334
7           Action      1229735.500000          0.086597   0.024290
8           userId       904534.812500          0.063697   0.017867
9           Comedy       772681.062500          0.054412   0.015262
10        Thriller       711706.687500          0.050118   0.014058

[1] "Plot variables importance"


[1] "The variable importance"
```




```
[1] "The scoring history of h2orf1"

Scoring History:
            timestamp          duration number_of_trees training_rmse
1 2019-04-22 20:28:56 25 min 48.392 sec               0            NA
2 2019-04-22 20:29:00 25 min 52.347 sec               1       0.95437
3 2019-04-22 20:29:04 25 min 56.322 sec               2       0.95141
4 2019-04-22 20:29:08 26 min  0.384 sec               3       0.94985
5 2019-04-22 20:29:12 26 min  4.251 sec               4       0.94858
  training_mae training_deviance
1           NA                NA
2      0.74891           0.91081
3      0.74656           0.90518
4      0.74548           0.90221
5      0.74441           0.89981

---
             timestamp          duration number_of_trees training_rmse
46 2019-04-22 20:31:55 28 min 46.636 sec              45       0.93905
47 2019-04-22 20:31:59 28 min 50.840 sec              46       0.93903
48 2019-04-22 20:32:03 28 min 54.761 sec              47       0.93900
```

```
49 2019-04-22 20:32:07 28 min 58.739 sec                48        0.93900
50 2019-04-22 20:32:11 29 min  2.568 sec                49        0.93898
51 2019-04-22 20:32:14 29 min  6.439 sec                50        0.93898
   training_mae training_deviance
46      0.73704           0.88181
47      0.73702           0.88179
48      0.73699           0.88172
49      0.73699           0.88171
50      0.73697           0.88169
51      0.73697           0.88167
```

```
[1] "Root mean squared error of h2orf1_model"
```

```
[1] 1.029505
```

RMSE results of all the models until now

Algorithm	RMSE
Matrix factorization with SGD	0.7829978
H2o Auto ML model	1.0382518
H2o GLM model	1.0163097
H2o GBM model	1.0353257
H2o RF model	1.0295050

H2o Stacked Ensembles

Ensemble machine learning methods use **multiple learning algorithms** to obtain better predictive performance than could be obtained from any of the **constituent learning algorithms** . H2O's Stacked Ensemble method is **supervised ensemble machine learning algorithm** that finds the optimal combination of a collection of prediction algorithms using a process called stacking. This method currently supports regression and binary classification. Stacking, also called Super Learning or Stacked Regression, is a class of algorithms that involves training a **second-level "metalearner"** to find the optimal combination of the base learners. Unlike bagging and boosting, the goal in stacking is to ensemble **strong, diverse sets of learners together.**

link

We will stack the previous 3 models:

-Generalized Linear Models
-Gradient Boosting Machine and the
-Distributed Random Forest (h2o_glm, h2o_gbm_model, h2orf_model)

```
h2oensemble2 <- h2o.stackedEnsemble(x = c("movieId","userId","n.movies_byUser","n.users_
                                         "Comedy","age_of_movie") ,
                                    y = "rating" ,
                                    training_frame = train3,
                                    validation_frame = test3,
                                    model_id = "h2oensemble2",
                                    seed = 1,
                                    base_models = list(h2o_glm, h2o_gbm_model,h2orf1))
```

-Algorithm: Stacked Ensemble
-Model ID: h2oensemble2

```
[1] "Root mean squared error of model h2oensemble2 (ensemble stack : GBM,GLM,DRF)"
```

```
[1] 1.003683
```

RMSE results of all the models until now

methods	rmse
Matrix factorization with SGD	0.7829978
H2o Auto ML model	1.0382518
H2o GLM model	1.0163097
H2o GBM model	1.0353257
H2o RF model	1.0295050
H2o Ensemble model	1.0036833

OUTPUT - VARIABLE IMPORTANCES
variable relative_importance scaled_importance percentage
-n.users_bymovie 7164049.0 1.0 0.3019
-Drama 5418105.0 0.7563 0.2283
-movieId 5250999.0 0.7330 0.2212
-age_of_movie 5224642.5000 0.7293 0.2201
-Comedy 675853.8125 0.0943 0.0285

Conclusions

The lowest RMSE was achieved only with **2 Features user ID and Movie ID** in Matrix factorization with SGD (RMSE 0.78).

The second lowest was the h2o ensemble model (RMSE 1.003) which stacked the below models

-(GLM,GBM,DRF)

With more hyper parameters tuning even lower RMSE can be achieved. But not so low like with MF models.

I wouldn't recommend the auto ML model seems you can't tune the hyper parameters of the models, that results not into the lowest RMSE or higher accuracy on classification models.

I also trained the same models with **scaled values but the RMSE was in all models higher**. It seems that the most important features where:

-Number of users rated the movie
-age of movie = more ratings and higher mean rating
-the movie id and
-Drama = (genre with the most ratings)

The other features didn't had low p value and didn't improve the model efficiency but the overfitted it.

In H2o Auto ML model (RMSE 1.03) the most important features where the below
OUTPUT - VARIABLE IMPORTANCES
-variable relative_importance scaled_importance percentage
-n.users_bymovie 7164049.0 1.0 0.3019
-Drama 5418105.0 0.7563 0.2283
-movieId 5250999.0 0.7330 0.2212

The GLM Model (RMSE 1.01) has similar evaluation process with Matrix factorization with SGD. Thats why put more weight on the below features
output - Standardized Coefficient Magnitudes (standardized coefficient magnitudes)

-movieId
-userId
-age_of_movie
-But the RMSE 1.01 was not so low like MF model.

The Gradient Boosting Machine Model (RMSE 1.03) put more weight on the below features
-variable relative_importance scaled_importance percentage
-age_of_movie 1759746.6250 1.0 0.3140
-n.users_bymovie 1684104.0 0.9570 0.3005
-movieId 883807.2500 0.5022 0.1577

The distributed Random Forest Model (RMSE 1.02) put more weight on the below features

-variable relative_importance scaled_importance percentage
-n.users_bymovie 26496842.0 1.0 0.2790
-age_of_movie 22493868.0 0.8489 0.2369
-movieId 21997428.0 0.8302 0.2317
-Drama 9633707.0 0.3636 0.1015

The **h2o ensemble model (RMSE 1.003)** stacked the below 3 models:

-GLM
-GBM
-DRF

And had significant the lowest RMSE of the H2o Models.
With more hyper parameters tuning even lower RMSE can be achieved. But not so low like with MF models.

Recommendations

With web scraping we could add more dimensions into our datasets such us:

-budget of movie

-critics rating and

-duration of movie and compare the RMSE's

I wouldn't recommend the auto ML model seems you can't tune the hyper parameters of the models and that results not into the optimal model.

Thank you for reading my analysis.

KR

Niko