MovieLens Recommendation System Project

Harvardx's Data Science Professional Certificate

Otman Cherrati

7/16/2021

Contents

1 Intorduction :

This report is part of the final capstone course of HarvardX Data Science Professional certificate taught by the famous professor Rafael Irizzary. The report is on movilens dataset. MovieLens is run by GroupLens Research, a research lab in the Department of Computer Science and Engineering at the University of Minnesota. Movilens recommends movies for its users to watch, based on their film preferences using collaborative filtering of members' movie ratings and movie reviews.

1.1 Object :

Our goal in this analysis is to create a movie recommendation system, using movilens 10M dataset, capable of generate predictions of users rating based on a training dataset. We will start by taking a look on the dataset and then explore the diffrent variables to see if there any trends that can help us make some assumptions. After cleaning the data we will start building and testing our models. The evaluation is based on the RMSE - Root Mean Squared Error that should be at least lower than 0.87750.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$$

Finally we will apply our final best model to the validation data set and discuss the results.

1.2 package used :

tidyverse()
dplyr()
ggplot2()
stringr()
data.table()
caret()
lubridate()
Xgboost()

1.3 SessionInfo() :

The output of sessionInfo() is placed here for reproducibility purposes.

```
## R version 4.0.5 (2021-03-31)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19042)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
```

```
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] xgboost_1.4.1.1   lubridate_1.7.10  data.table_1.14.0 forcats_0.5.1
##  [5] stringr_1.4.0     dplyr_1.0.5       purrr_0.3.4       readr_1.4.0
##  [9] tidyr_1.1.3       tibble_3.1.1      ggplot2_3.3.3     tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.0  xfun_0.22         lattice_0.20-41   haven_2.4.1
##  [5] colorspace_2.0-0  vctrs_0.3.7       generics_0.1.0    htmltools_0.5.1.1
##  [9] yaml_2.2.1        utf8_1.2.1        rlang_0.4.10      pillar_1.6.0
## [13] glue_1.4.2        withr_2.4.2       DBI_1.1.1         dbplyr_2.1.1
## [17] modelr_0.1.8      readxl_1.3.1      lifecycle_1.0.0   munsell_0.5.0
## [21] gtable_0.3.0      cellranger_1.1.0  rvest_1.0.0       evaluate_0.14
## [25] knitr_1.33        fansi_0.4.2       broom_0.7.6       Rcpp_1.0.6
## [29] scales_1.1.1      backports_1.2.1   jsonlite_1.7.2    fs_1.5.0
## [33] hms_1.0.0         digest_0.6.27     stringi_1.5.3     grid_4.0.5
## [37] cli_2.5.0         tools_4.0.5       magrittr_2.0.1    crayon_1.4.1
## [41] pkgconfig_2.0.3   Matrix_1.3-2      ellipsis_0.3.1    xml2_1.3.2
## [45] reprex_2.0.0      assertthat_0.2.1  rmarkdown_2.7     httr_1.4.2
## [49] rstudioapi_0.13   R6_2.5.0          compiler_4.0.5
```

1.4 Load the movilens data set :

To load the data set we run the code provided by Edx.

1.5 Dataset Description

We will use the 10M version of the MovieLens dataset. The 10 Millions dataset has been divided into two dataset: edx for training purpose and validation for the final validation phase. The edx dataset contains approximately 9 Millions of rows with 69878 different users and 10677 movies. we will focus in our exploration and analysis on Edx data set, the validation set has the same features and will only be used in the final step.

2 Data Exploration :

2.1 Initial exploration :

The structure of the data :

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

The six features of the data set are (same as validation) :
movieId: Unique ID for the movie.
userId: Unique ID for the user.
rating: A rating between 0 and 5 for the movie.
timestamp: Date and time the rating was given.
title: Movie title and Year the movie was released. (not unique).
genres: Genres associated with the movie.

Let's take a look on the first six rows of Edx data :

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

The Edx data contains 9000055 rows and 6 variables

x
9000055
6

Look for missing values :
we notice that there is no missing values in the edx dataset.

	x
userId	0
movieId	0
rating	0
timestamp	0
title	0
genres	0

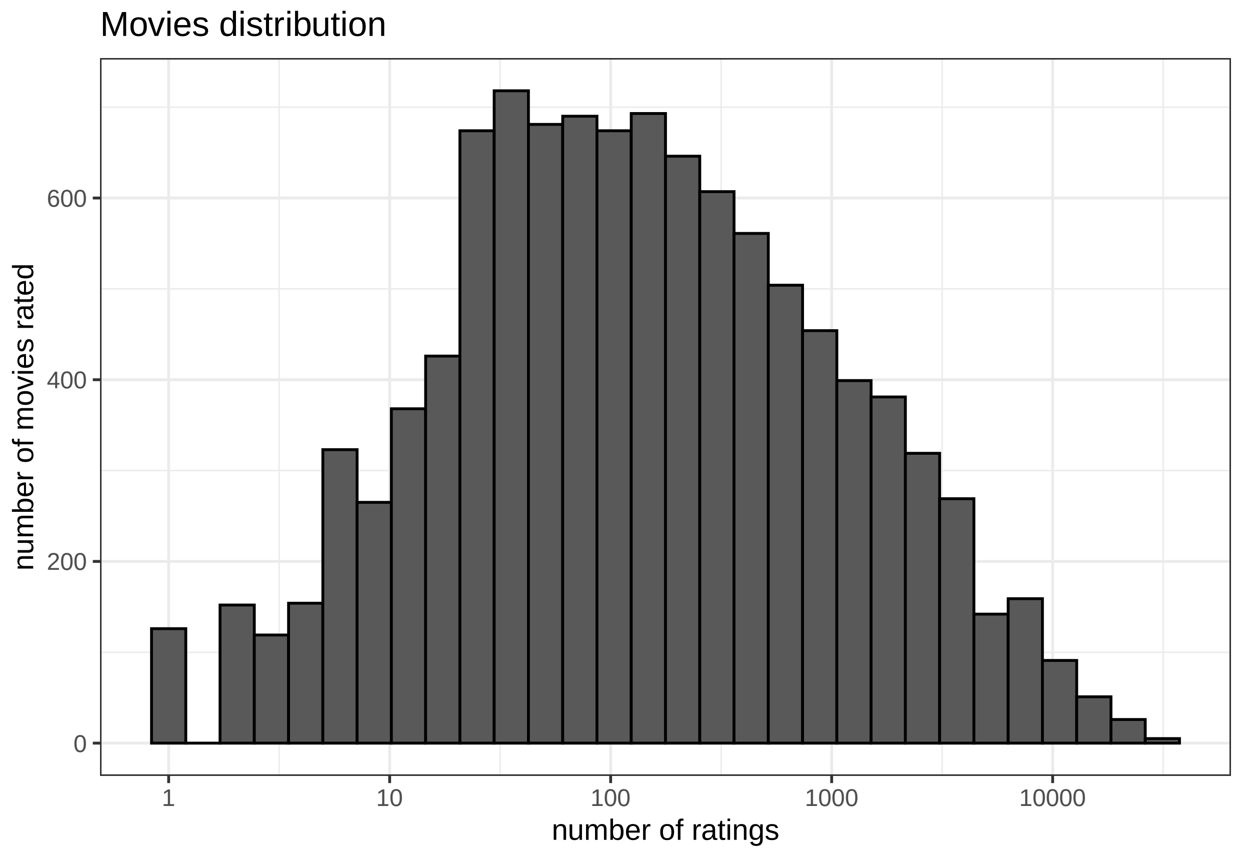
Number of movies and users : There is 69878 user and 10677 movie in the edx dataset.

users	movies
69878	10677

2.2 Variables Exploration

2.2.1 Movies :

There are 10677 different movies in the edx set, some of them are rated more than others



First six most rated movies

```
## `summarise()` has grouped output by 'movieId'. You can override using the `.groups` argument.
```

movieId	title	n
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

We notice that 90% of movies are rated at least 10 times:

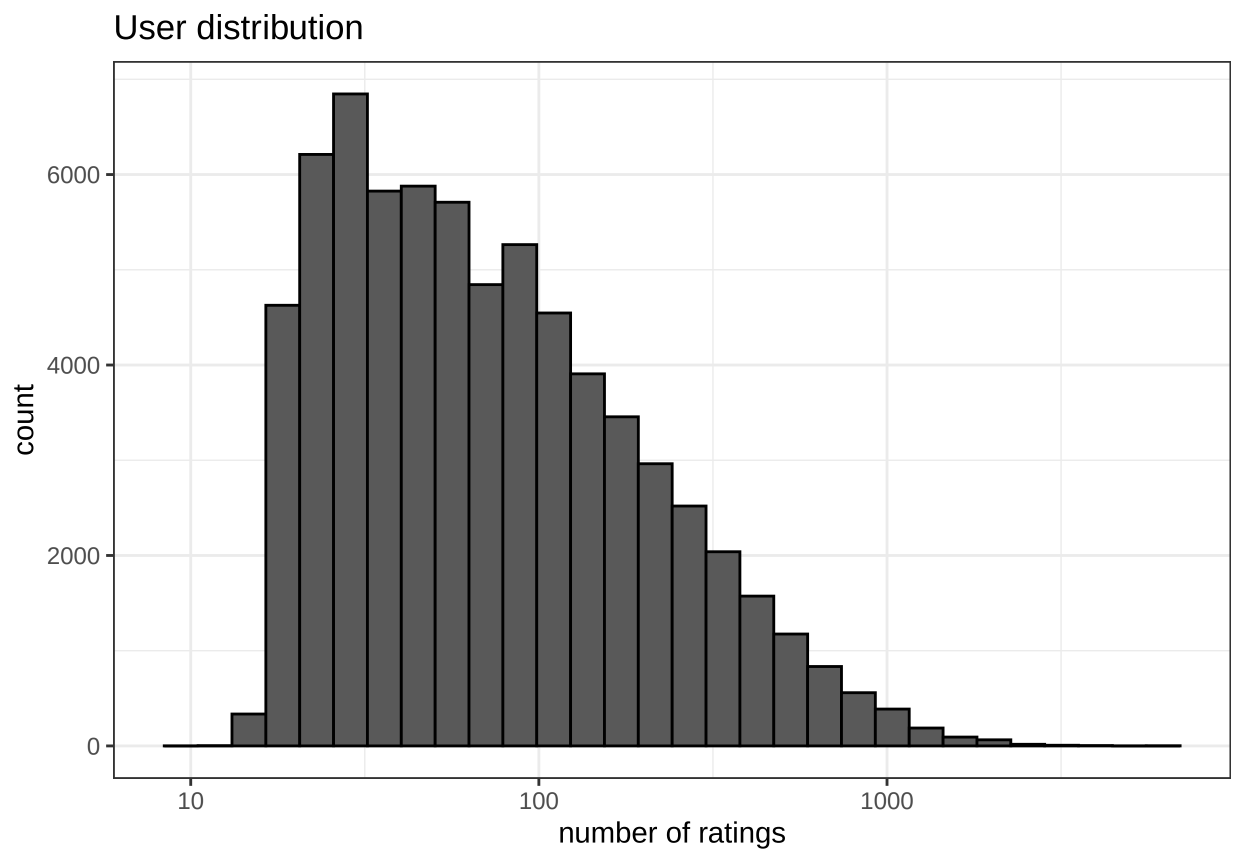
movies which are rated at least 10 times
0.9012831

Summary of rating per movies:
The average number of rating per movie is 842.9.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    30.0   122.0   842.9   565.0 31362.0
```

2.2.2 users

There are 69878 different users in the edx set :



We notice that the user distribution is right skewed. Not every user is equally active. Some users have rated very few movie and others have rated a lot of movies.

Six users that rated the most :

userId	n
59269	6616
67385	6360
14463	4648
68259	4036
27468	4023
19635	3771

Summary rating per users: The average number rating per user is : 129 ratings

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    32.0    62.0   128.8   141.0  6616.0
```

95 % of the users rated more than 20 movies

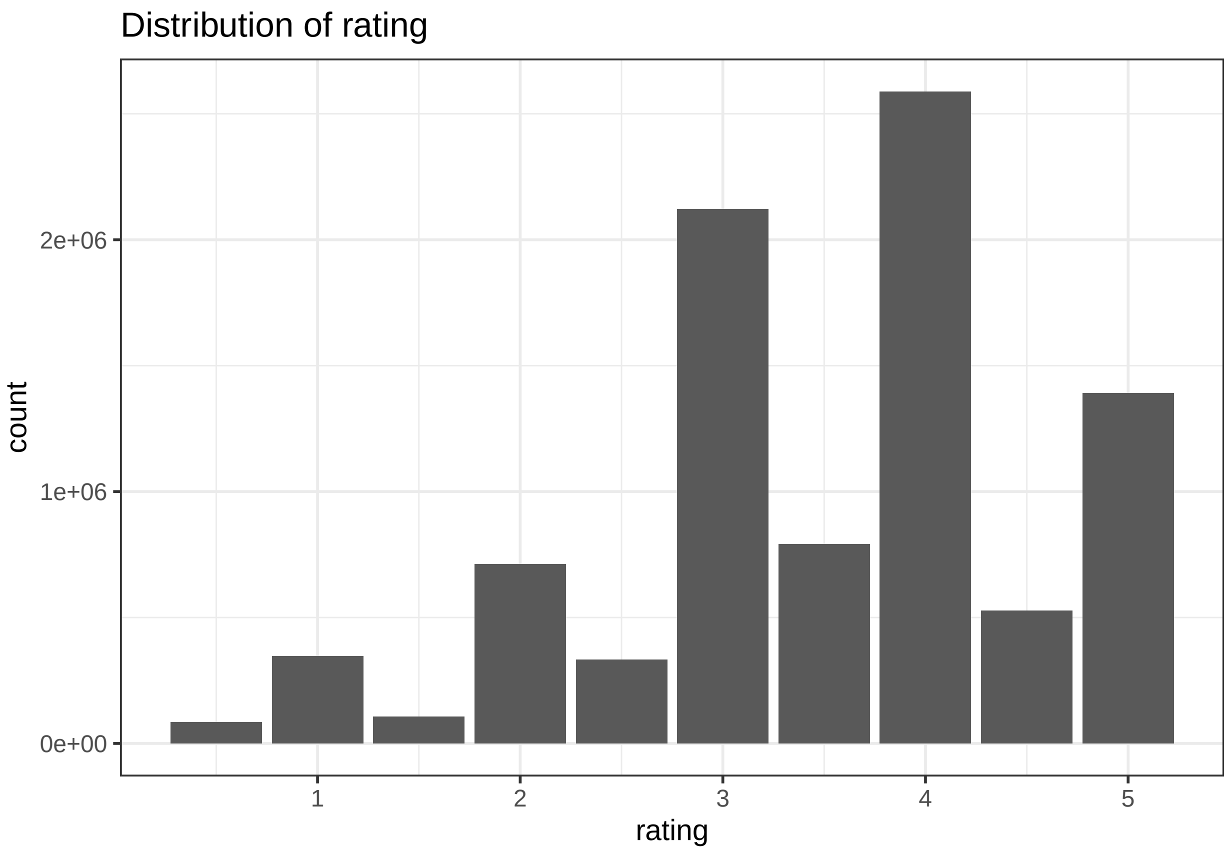
users who rated more than 20 movies
0.950342

2.2.3 Rating

The number of ratings in the dataset is : 9000055
The rating range is between 0.5 and 5 : 10 possible choices.

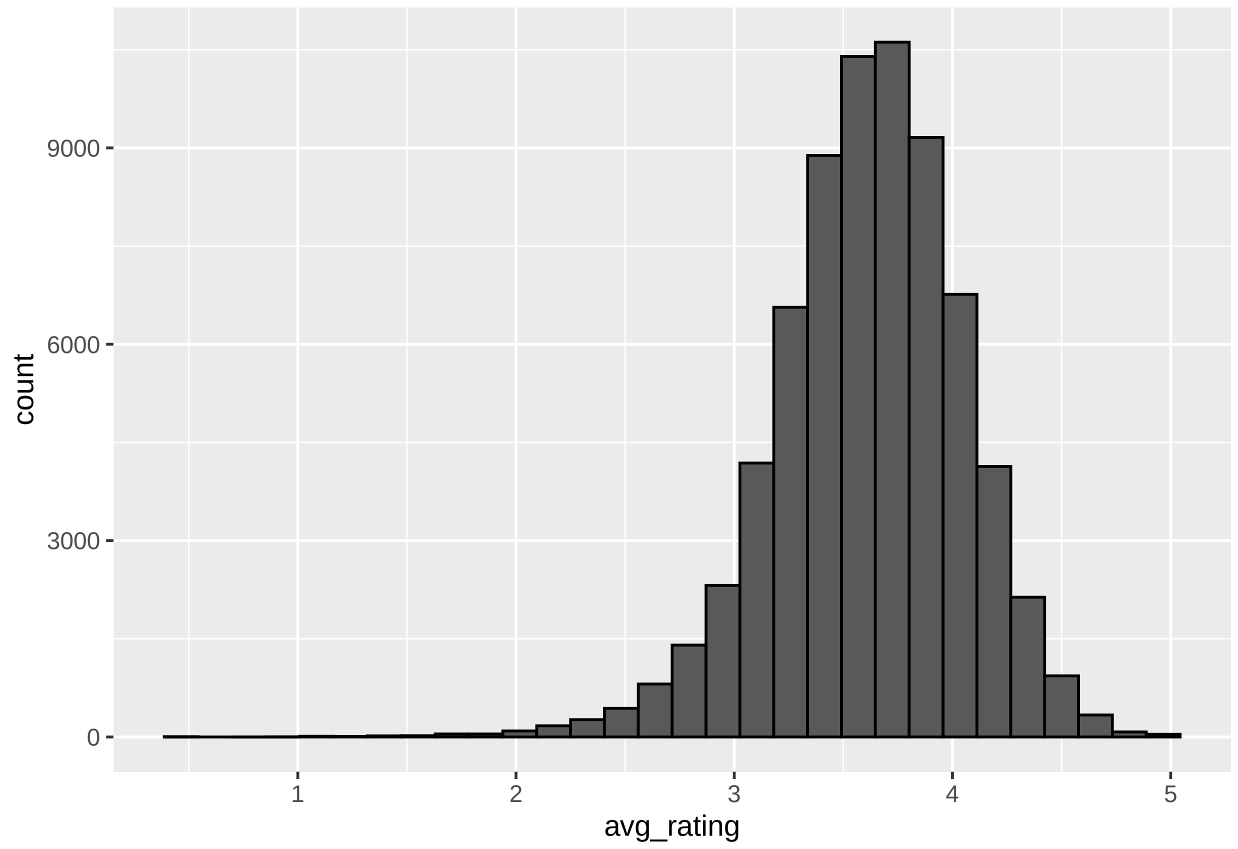
Total number for each rating:

rating	n
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



We se that most rating get rounded value and that users have a general tendency to rate movies between 3 and 4.

Average rating per user :



The distribution is almost symmetric.

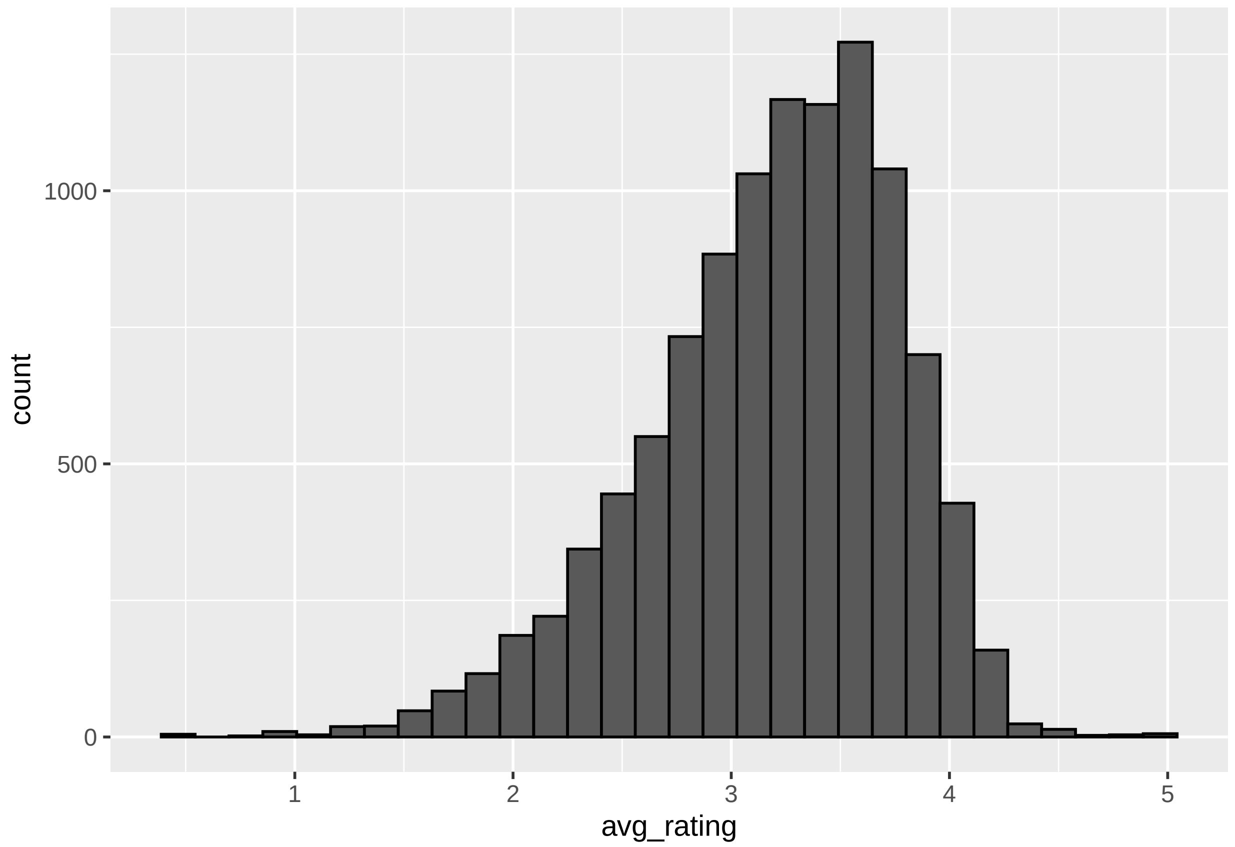
Summary of average rating per user :

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.357   3.635   3.614   3.903   5.000
```

93% of users have an average rating of at least 3

```
## # A tibble: 1 x 1
##   `mean(avg_rating >= 3)`
##                     <dbl>
## 1                   0.928
```

Average rating per movie:



The distribution is left skewed.

Summary of average rating per movie

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   2.844   3.268   3.192   3.609   5.000
```

92% of movies have an average rating between 2 and 4 :

```
## # A tibble: 1 x 1
##   `mean(avg_rating >= 3)`
##                     <dbl>
## 1                   0.680
```

2.2.4 Date :

We will have to do some data processing to explore the date of release and date of rating.

2.2.4.1 Date of rating :

We convert timestamp to year of rating :

That's how our data looks like now :

userId	movieId	rating	title	genres	rating_year
1	122	5	Boomerang (1992)	Comedy|Romance	1996
1	185	5	Net, The (1995)	Action|Crime|Thriller	1996
1	292	5	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1996

userId	movieId	rating	title	genres	rating_year
1	316	5	Stargate (1994)	Action|Adventure|Sci-Fi	1996
1	329	5	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1996
1	355	5	Flintstones, The (1994)	Children|Comedy|Fantasy	1996

Summary of year of rating :

The rating start in 1995, ends in 2009 and last more than 14 years.

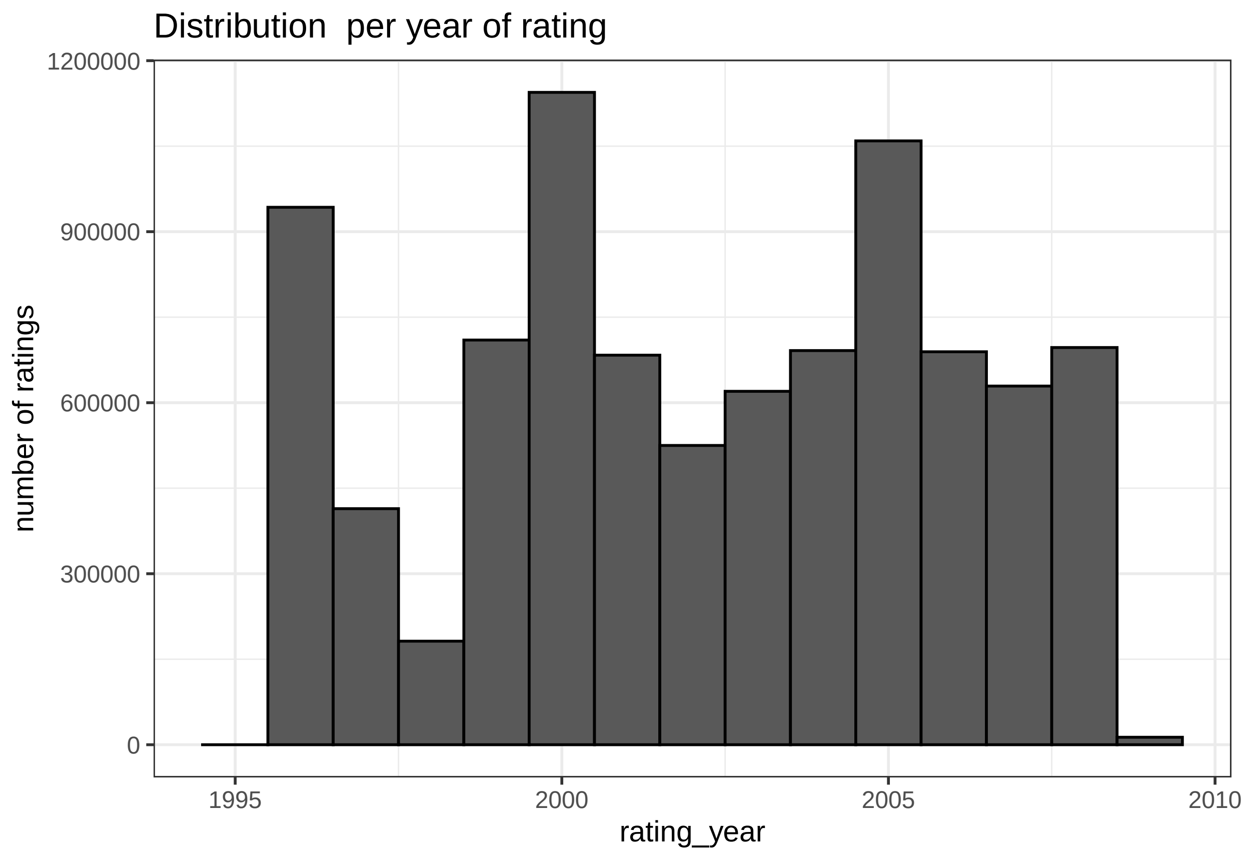
```
##   Start  End Period
## 1  1995 2009     14
```

|| || || ||

The ten years with most number of ratings:

rating_year	n
1995	2
1996	942772
1997	414101
1998	181634
1999	709893
2000	1144349
2001	683355
2002	524959
2003	619938
2004	691429

Plot the distribution of number of rating per year :



2.2.4.2 Year of release:

We'll extract the year of release from the title and take a look on the data :

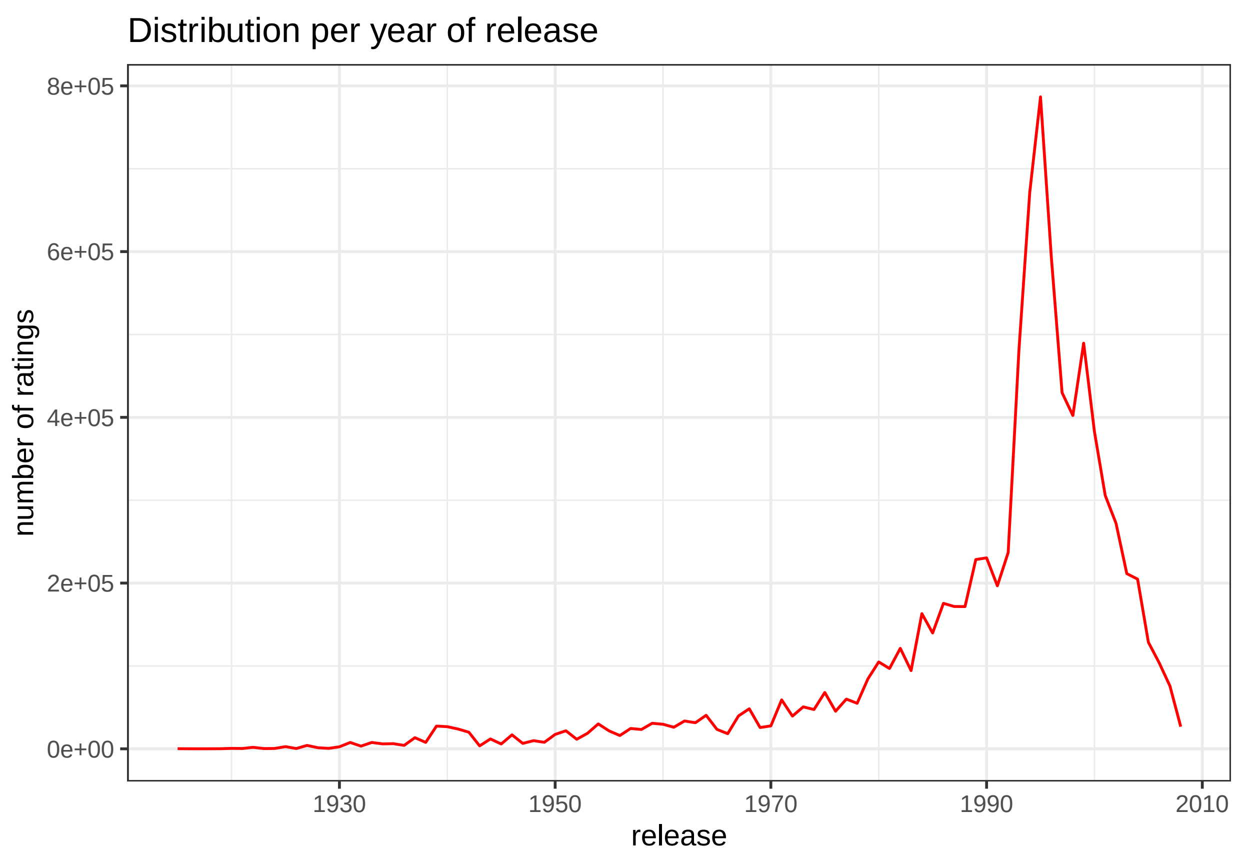
userId	movieId	rating	title	release	genres	rating_year
1	122	5	Boomerang	1992	Comedy|Romance	1996
1	185	5	Net, The	1995	Action|Crime|Thriller	1996
1	292	5	Outbreak	1995	Action|Drama|Sci-Fi|Thriller	1996
1	316	5	Stargate	1994	Action|Adventure|Sci-Fi	1996
1	329	5	Star Trek: Generations	1994	Action|Adventure|Drama|Sci-Fi	1996
1	355	5	Flintstones, The	1994	Children|Comedy|Fantasy	1996

Summary year of relaese:

The first movie was released in 1915 and the last one 2008.

```
##   Start  End period
## 1  1915 2008     93
```

Distribution of ratings per year of release



The 10 years of release with the most rated movies:

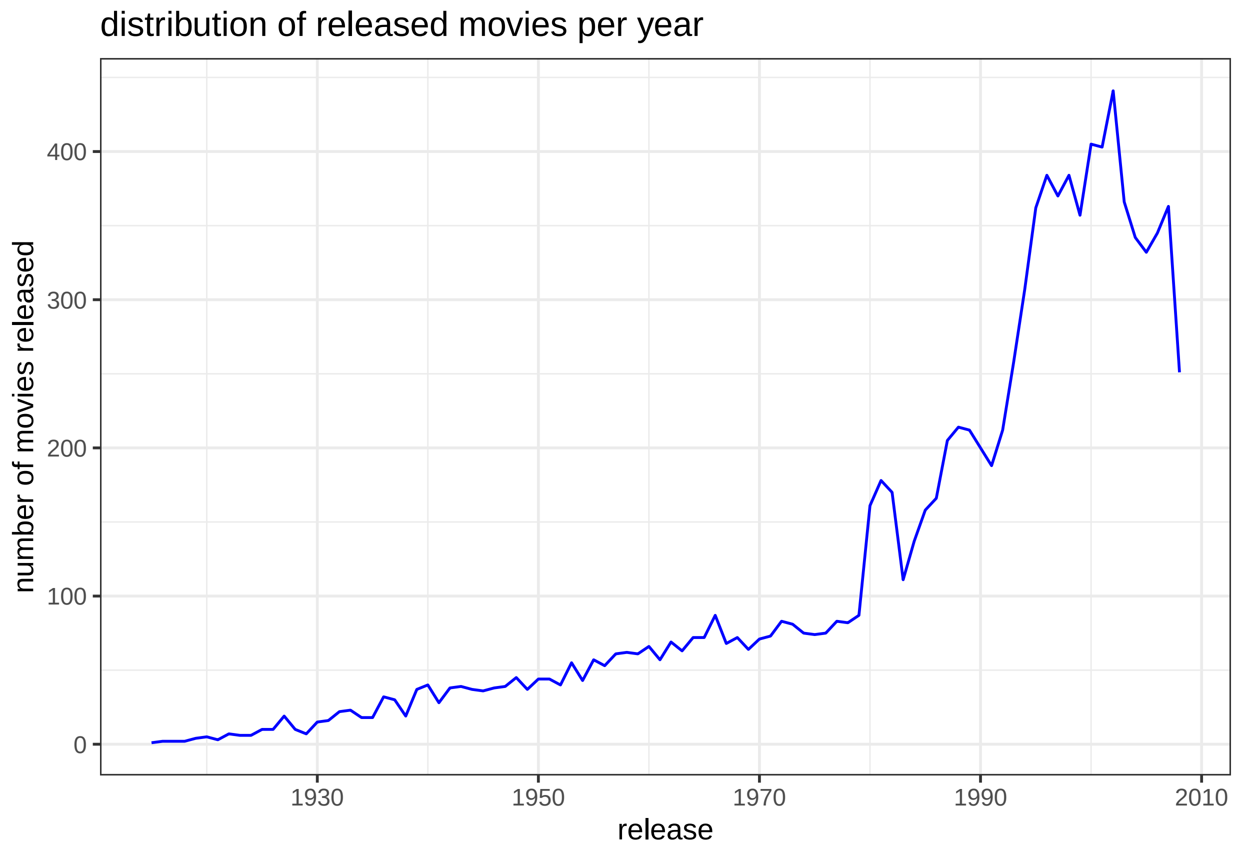
release	n
1995	786762
1994	671376
1996	593518
1999	489537
1993	481184
1997	429751
1998	402187
2000	382763
2001	305705
2002	272180

Number of released movies per year

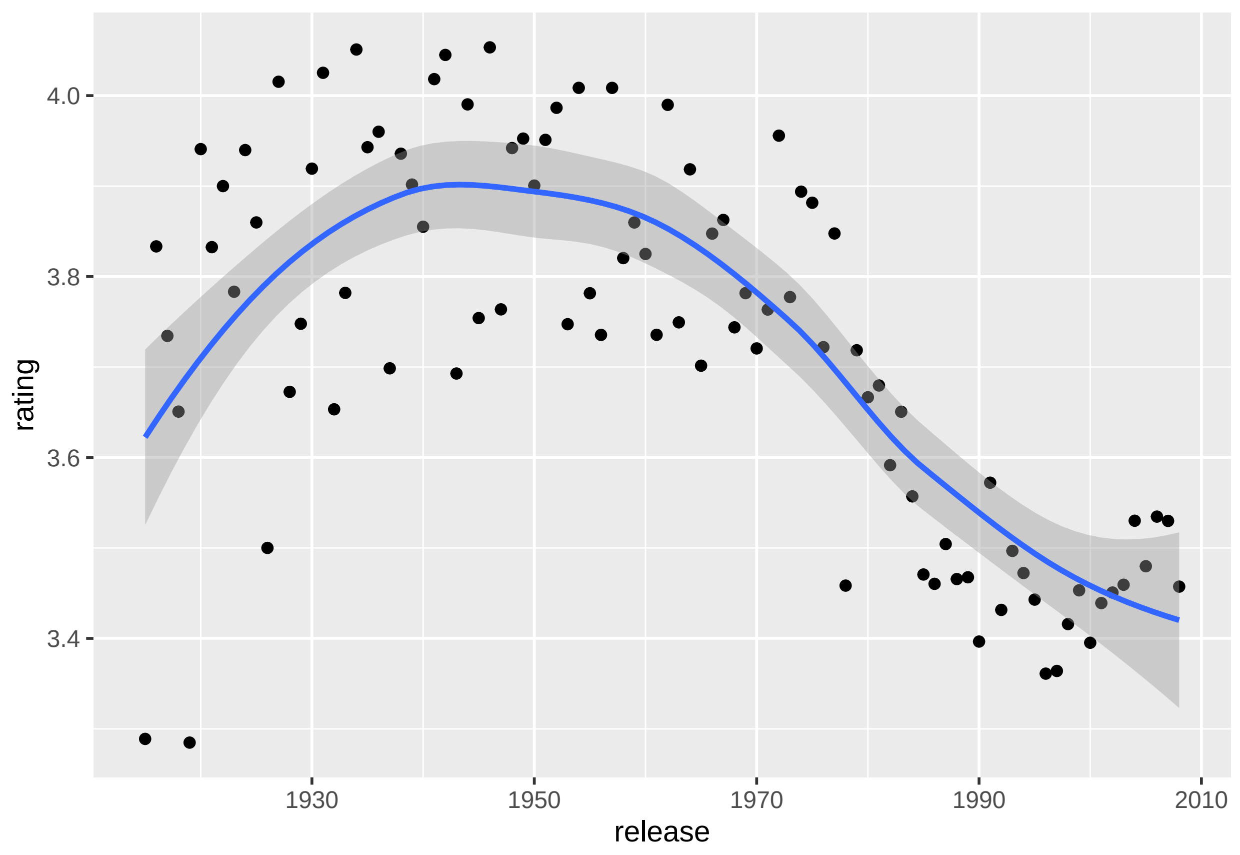
The 10 years with most released movies :

release	n
2002	441
2000	405
2001	403
1996	384
1998	384

release	n
1997	370
2003	366
2007	363
1995	362
1999	357



rating vs year of release



The general trend shows modern users relatively rate movies lower.

2.2.5 Genre

Number of rating per genre (genre in attached format) :

genres	n
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Drama|Romance	259355

Number of genre combinations :

x
797

We will split the genres so as to have one genre in each row.

This is how our data looks now:

userId	movieId	rating	release	genres	rating_year
1	122	5	1992	Comedy	1996
1	122	5	1992	Romance	1996
1	185	5	1995	Action	1996
1	185	5	1995	Crime	1996
1	185	5	1995	Thriller	1996
1	292	5	1995	Action	1996

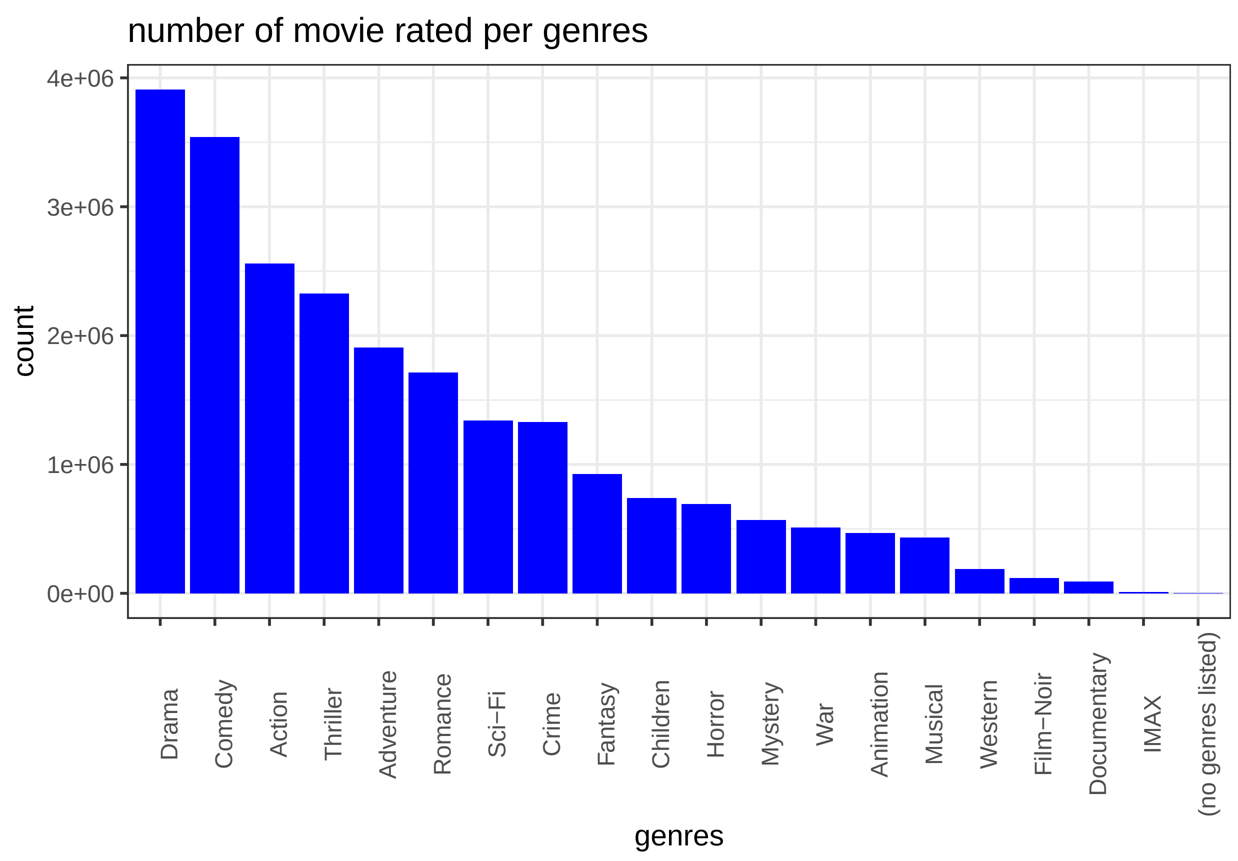
Number of genres in our data :

```
## [1] "20"
```

Number of rating per each genre :

genres	n
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

Distribution of rating per genres :


number of movie rated per genres
count
4e+06
3e+06
2e+06
1e+06
0e+00
Drama
Comedy
Action
Thriller
Adventure
Romance
Sci−Fi
Crime
Fantasy
Children
Horror
Mystery
War
Animation
Musical
Western
Film−Noir
Documentary
IMAX
(no genres listed)
genres

3 Data preprocessing :

3.1 Data cleaning :

We have already cleaned our data, so now we will only convert genres to integer to make our code run fast and because most models works well with numeric class, this will not affect our results.

3.2 Create train and test set :

The train set will be 80% of the edx data while the test set will be 20%.

train set head :

userId	movieId	rating	release	genres	rating_year
1	122	5	1992	6	1996
1	122	5	1992	16	1996
1	185	5	1995	18	1996
1	292	5	1995	2	1996
1	292	5	1995	9	1996
1	292	5	1995	17	1996

test set head :

userId	movieId	rating	release	genres	rating_year
1	185	5	1995	2	1996
1	185	5	1995	7	1996
1	316	5	1994	3	1996
1	329	5	1994	17	1996
1	364	5	1994	5	1996
1	377	5	1994	18	1996

4 Models building :

Machine learning algorithms in recommender systems typically fit into two categories: content-based systems and collaborative filtering systems. Modern recommender systems combine both approaches.

A) Content-based methods are based on the similarity of movie attributes. Using this type of recommender system, if a user watches one movie, similar movies are recommended.

B) With collaborative filtering, the system is based on past interactions between users and movies. With this in mind, the input for a collaborative filtering system is made up of past data of user interactions with the movies they watch.

In our method we will try seven models starting with a naive approach, a content based one and then using some collaborative filtering with the others. We will get their RMSE on the test set, the one with the least RMSE in the test set will then be applied on the validation set to get the final RMSE result.

4.1 Naive model : just the mean

In this simple model we will make our predictions, by simply using the mean of all ratings as the estimate for every unknown rating.

```
## [1] "The mean is"      "3.52697162479366"
```

The formula used for this model is :

$$Y_{u,i} = \hat{\mu} + \epsilon_{u,i}$$

with û the mean rating and upsilon a random error.

method	rmse
just the avg	1.052

The rmse is 1.05 which is very big and that means that in average we are missing the true prediction by a whole star rating.

4.2 Movie effct model :

Some movies are rated more often than others. So in this model we will take into account the variability from movie to movie. A bias term b_i will be calculated for each movie to determine how much better or worse a given film is from the overall average.

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

method	rmse
just the avg	1.052
movie_eff	0.941

The RMSE on the test set is 0.94 , it's better but still not good enough.

4.3 Movie plus User effect model :

Some users are positive and some have negative reviews because of their own personal liking/disliking regardless of movie. Users may have a tendency to rate movies higher or lower than the overall mean. So let's add this into the model. First we'll calculate the bias for each user:

Then we'll combine the user bias with the movie bias to get the prediction.

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

method	rmse
just the avg	1.0520
movie_eff	0.9410
movie_user_eff	0.8575

The RMSE is now 0.857 on the test set .

4.4 Movie, User and genre effect model :

The rating of a movie depends also on its genre, some genre have their avrerage rating a little better than others, we will add the genre bias to our model and see how this can improve our precision.

$$Y_{u,i} = \hat{\mu} + b_i + b_u + b_g + \epsilon_{u,i}$$

method	rmse
just the avg	1.0520
movie_eff	0.9410
movie_user_eff	0.8575
movie_user_g_eff	0.8574

4.5 Movie, User, genre, and release year effect model :

The popularity of the movie genre depends strongly on the contemporary issues. So we should also add the date bias b_d to our analysis.

$$Y_{u,i} = \hat{\mu} + b_i + b_u + b_g + b_d + \epsilon_{u,i}$$

method	rmse
just the avg	1.0520
movie_eff	0.9410
movie_user_eff	0.8575
movie_user_g_eff	0.8574
movie_user_gre_rel_eff	0.8571

4.6 Movie, User, genre, release and rating year effect model :

The users mindset also has evolved over time. This can also effect the average rating of movies over the years. So we will add a rating date bias b_r to our model.

$$Y_{u,i} = \hat{\mu} + b_i + b_u + b_g + b_d + b_r + \epsilon_{u,i}$$

method	rmse
just the avg	1.0520

method	rmse
movie_eff	0.9410
movie_user_eff	0.8575
movie_user_g_eff	0.8574
movie_user_gre_rel_eff	0.8571
movie_user_year_eff	0.8570

4.7 Xgboost model : Extreme Gradient Boosting

XGBoost stands for "Extreme Gradient Boosting,it is an extension to gradient boosted decision trees (GBM) and specially designed to improve speed and performance.

XGBoost, Like other gradient boosting algorithms, operates on decision trees, models that construct a graph that examines the input under various "if" statements. Whether the "if" condition is satisfied influences the next "if" condition and eventual prediction. XGBoost progressively adds more and more "if" conditions to the decision tree to build a stronger model.

$$y_i = \sum_{k=1}^{K} f_k(x_i), f_k £F$$

where K is the number of trees, and f is a function in the functional space F, and F is the set of all possible Classification and regression trees.

XGBoost also works in a similar manner as Gradient Boosting model but introduces regularisation terms to counter overfitting..

4.7.1 Data processing :

From the last model we conclude that there is an impact of the rating average per movie, per user and per genre on the true rating, we we will add those to our data to let our model take advantage of these features.

4.7.1.1 add averages to train set

userId	movieId	rating	release	genres	rating_year	mov_avg	user_avg	genre_avg
1	122	5	1992	6	1996	2.853	5	3.437
1	122	5	1992	16	1996	2.853	5	3.553
1	185	5	1995	18	1996	3.127	5	3.508
1	292	5	1995	2	1996	3.417	5	3.422
1	292	5	1995	9	1996	3.417	5	3.674
1	292	5	1995	17	1996	3.417	5	3.397

4.7.1.2 add averages to test set :

userId	movieId	rating	release	genres	rating_year	mov_avg	user_avg	genre_avg
1	185	5	1995	2	1996	3.137	5	3.421
1	185	5	1995	7	1996	3.137	5	3.670
1	316	5	1994	3	1996	3.335	5	3.493
1	329	5	1994	17	1996	3.337	5	3.393
1	364	5	1994	5	1996	3.750	5	3.421
1	377	5	1994	18	1996	3.529	5	3.506

4.7.1.3 Define predictor and response variables in the datasets : We convert our train and test set predictors to an object data.matrix and we define the response variablse using this piece of code :

> train_x = data.matrix(train[, -3])
> train_y = train[,3]
> test_x = data.matrix(test[, -3])
> test_y = test[, 3]

4.7.2 define final training and testing sets :

xgb_train = xgb.DMatrix(data = train_x, label = train_y) xgb_test = xgb.DMatrix(data = test_x, label = test_y)

4.7.3 Define watchlist :

watchlist = list(train=xgb_train, test=xgb_test)

4.7.4 Fit XGBoost model and display training and testing data at each round to chose the best parameters :

```
## [1]  train-rmse:2.326025 test-rmse:2.326017
## [2]  train-rmse:1.741133 test-rmse:1.740154
## [3]  train-rmse:1.365716 test-rmse:1.363926
## [4]  train-rmse:1.137158 test-rmse:1.134715
## [5]  train-rmse:1.006137 test-rmse:1.003436
## [6]  train-rmse:0.935125 test-rmse:0.932227
## [7]  train-rmse:0.897967 test-rmse:0.895104
## [8]  train-rmse:0.878937 test-rmse:0.876211
## [9]  train-rmse:0.869264 test-rmse:0.866669
## [10] train-rmse:0.864265 test-rmse:0.861857
## [11] train-rmse:0.861663 test-rmse:0.859381
## [12] train-rmse:0.860235 test-rmse:0.858115
## [13] train-rmse:0.859326 test-rmse:0.857379
## [14] train-rmse:0.858688 test-rmse:0.856999
## [15] train-rmse:0.858179 test-rmse:0.856683
## [16] train-rmse:0.857665 test-rmse:0.856367
## [17] train-rmse:0.857318 test-rmse:0.856162
## [18] train-rmse:0.856882 test-rmse:0.856007
## [19] train-rmse:0.856623 test-rmse:0.855915
## [20] train-rmse:0.856471 test-rmse:0.855860
```

4.7.5 Define final model :

```
## [1]  train-rmse:2.326025
## [2]  train-rmse:1.741133
## [3]  train-rmse:1.365716
## [4]  train-rmse:1.137158
## [5]  train-rmse:1.006137
## [6]  train-rmse:0.935125
## [7]  train-rmse:0.897967
## [8]  train-rmse:0.878937
## [9]  train-rmse:0.869264
## [10] train-rmse:0.864265
## [11] train-rmse:0.861663
## [12] train-rmse:0.860235
## [13] train-rmse:0.859326
## [14] train-rmse:0.858688
## [15] train-rmse:0.858179
```

4.7.6 Get predictions on test set :

```
## [1] "The rmse for Xgboost is" "0.856782750024054"
```

method	rmse
just the avg	1.0520
movie_eff	0.9410
movie_user_eff	0.8575
movie_user_g_eff	0.8574
movie_user_gre_rel_eff	0.8571
movie_user_year_eff	0.8570
xgboost	0.8568

5 Final results :

apply the winning model for on validation set

Based on the results we have got in the previous section now we know that the Xgboost model did the best, in this final section we will apply it on the validation data set and get the final RMSE. But before this we will have to do some data preprocessing on the the dataset.
## Validation loading and cleaning :

There is no missing values in the validation data set.

```
##    userId   movieId    rating timestamp     title    genres 
##         0         0         0         0         0         0
```

We will extract the rating year, year of release, the genres and convert genres to integer in the validation data set. Then we will add averages column of rating per movie, user and genre.

5.1 Validation set preparation :

In the next step we define the matrix of predictors and our response variable.

5.2 Runn final model on validation set :

We run the final Xgboost model on the validation matrix.

```
## [1] "The MRSE of Xgboost applied on validation set is"
## [2] "0.833739583764478"
```

The overall objective to training a machine learning algorithm of course resides in being able to generate predictions. On the validation dataset, we managed to reach an RMSE of 0.8337.

6 Conclusion :

The project goal was to build a model that predict movie ratings. Our approach undertaken here relied on an initial exploratory analysis with data visualizations, then we start building models based on the assumptions made in the last part, the models take into account all the features. Finnally, we chose the model that performs the least RMSE and applied it on the validation set which produced an RMSE of 0.83 that is better than out target of 0.86.

While these results are promising, future work could improve both predictive performance and speed of the algorithm in the following ways :
- Use matrix factorization.
- Use models that consume less time and memory.
- Use the sampling techniques as it permit to work on large data set and guarantee generalizability.