HarvardX PH125.9x - Data Science: Capstone

Machine Learning - Case study: Movielens Data Set

Paolo Sebastianelli

2021-07-27

Contents

Preface

This report is part of the requirements for completing the final course in HarvardX's multi-part **Data Science Professional Certificate** series.

Introduction

Members of the GroupLens research group, at the University of Minnesota, maintain the Movielens research site that can be found at this link.

Using "collaborative filtering" technology, personalized predictions are generated for the user that wants find new movies to enjoy.

The Movielens 10M dataset (Harper and Konstan 2015) is provided publicly at this link https://grouplens.org/datasets/movielens/10m/ and http://files.grouplens.org/datasets/movielens/ml-10m.zip is used to develop this Capstone project.

This is what is written in the official web page about the data set:

> *This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens. Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided.".*

Specifically, the **goal** of this project is to create a Movie Recommendation Machine Learning Algorithm for predicting movie ratings a particular user will give a specific item.

To reach the objective of building an appropriate ML model, the data have been investigated numerically and visually and the results of the gained insights are discussed in a dedicated section.

Data Analysis

This is the code provided by the instructor Rafael Irizarry to generate the data sets:

```
### Generating the Data Set

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

##### LOADING THE LIBRARIES ###############

library(tidyverse)
library(caret)
library(data.table)
library(Metrics)
library(dplyr)

dl <- tempfile()

### downloading the zip file ml-10m.zip from files.grouplens.org
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::",
                             "\t",
                             readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl,
                                          "ml-10M100K/movies.dat")), "\\::", 3)

colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:

##mutating movies columns

movies <- as.data.frame(movies) %>%
          mutate(movieId = as.numeric(movieId),
```

```
                  title  = as.character(title),
                  genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

set.seed(1) # if using R 3.5 or earlier, use `set.seed(1)`

test_index <- createDataPartition(y = movielens$rating,
                                  times = 1,
                                  p = 0.1,
                                  list = FALSE)

edx <- movielens[-test_index,] ### removing data
temp <- movielens[test_index,]

# Make sure userId and movieId in validation
# set are also in edx set using semi_join function

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)  #### edx contains again all the rows

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

By running the provided code, two data sets are available to train, test (*edx* data set) and validate (*validation* data set) the ML model that is meant to be developed.

Edx Data Set

The *edx* data set can be introduce by the use of DataExplorer (Cui 2020) library:

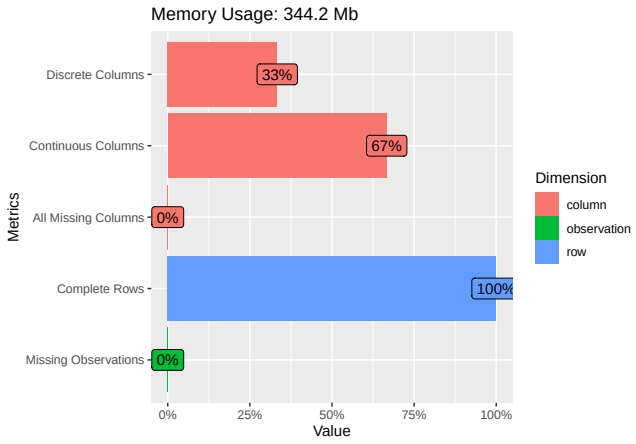

Figure 1: Metrics for edx data set

There are no missing value in the data set, as shown in Fig. 1.

Before to continue the analysis *edx* and *validation* sets have been modified for further investigation, adding

other five columns.

The five columns are: *date*, *date_week*, *time* with format "%H," the year of the rating *rat_year* with format "%Y" and the movie release year *movie_year* also with format "%Y" .

```
library(tidyr)
edx <- mutate(edx, date = as_datetime(timestamp))
edx <- mutate(edx, date_week = round_date(date, unit = "week"))
edx <- mutate(edx, time = format(date, format = "%H"))
edx <- mutate(edx, rat_year = format(date, format = "%Y"))
movie_year <- as.numeric(str_sub(edx$title,-5,-2))
edx <- mutate(edx, movie_year = movie_year)
validation <- mutate(validation, date = as_datetime(timestamp))
validation <- mutate(validation, date_week = round_date(date, unit = "week"))
validation <- mutate(validation, time = format(date, format = "%H"))
validation <- mutate(validation, rat_year = format(date, format = "%Y"))
movie_year_v <- as.numeric(str_sub(validation$title,-5,-2))
validation <- mutate(validation, movie_year = movie_year_v)
```

The result can be seen as a *tibble*:

```
## # A tibble: 9,000,061 x 11
##    userId movieId rating timestamp title       genres        date
##     <int>   <dbl>  <dbl>     <int> <chr>       <chr>         <dttm>
##  1      1     122      5 838985046 Boomerang ~ Comedy|Roman~ 1996-08-02 11:24:06
##  2      1     185      5 838983525 Net, The (~ Action|Crime~ 1996-08-02 10:58:45
##  3      1     231      5 838983392 Dumb & Dum~ Comedy        1996-08-02 10:56:32
##  4      1     292      5 838983421 Outbreak (~ Action|Drama~ 1996-08-02 10:57:01
##  5      1     316      5 838983392 Stargate (~ Action|Adven~ 1996-08-02 10:56:32
##  6      1     329      5 838983392 Star Trek:~ Action|Adven~ 1996-08-02 10:56:32
##  7      1     355      5 838984474 Flintstone~ Children|Com~ 1996-08-02 11:14:34
##  8      1     356      5 838983653 Forrest Gu~ Comedy|Drama~ 1996-08-02 11:00:53
##  9      1     362      5 838984885 Jungle Boo~ Adventure|Ch~ 1996-08-02 11:21:25
## 10      1     364      5 838983707 Lion King,~ Adventure|An~ 1996-08-02 11:01:47
## # ... with 9,000,051 more rows, and 4 more variables: date_week <dttm>,
## #   time <chr>, rat_year <chr>, movie_year <dbl>
```

and summarized in this way:

```
##   n_genres n_movies n_users n_ratings n_years n_movie_years
## 1      797    10677   69878   9000061      15            94
```

The data are spread on *fifteen* years of rating. The range of movies release year is of ninety four years.

It contains 10677 movies and 69878 users. The called user-item matrix (or Rating) $\mathbf{R}$ is characterized by

$$10677 \cdot 69878 = 7.46087406 \cdot 10^8$$

entries. Logically, not all the users have provided a rating for all the 10677 movies, ,maybe nobody of them. The R matrix is a sparse matrix. The goal is to predict the rating $y_{u,i}$ that the user u would give to the item (movie) i.

Talking about "Rating"

Talking about *rating*, a bunch of questions can be addressed before to develop a ML algorithm, to build a prediction model more efficient.

- Are some movies more rated than others?
- Are some movies characterized for a higher mean rating value?

- Are some users more active in rating than others?
- Are there users that usually rate movies with a higher or lower rating than others?
- The same kind of questions can be proposed for the genres and time variables?

Trying to address all these questions, a first glance to the *edx* data set is provided.

- What is the rating mean value all over the movies in *edx* data set (hereafter MRedx)?

The mean value is:

```
## [1] 3.512464
```

- What is the distribution of rating values all over movies that are in the *edx* set?

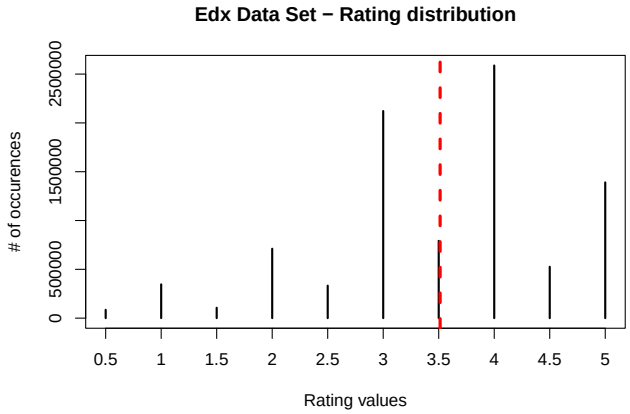

Figure 2: Rating values distribution for the edx data set. The red dashed line represents the mean value

As can be seen in Fig. 2, the major part of ratings are placed between 3 and 4.

Movies

If the data are grouped by title and the *Rating Mean Value* (here after RMV) is calculated per each title, it is possible to observe that some titles are characterized by RMV higher than the mean value MRedx and others lower.

The following tables shows some interesting insights.

The first one shows the movie RMV in a *count* (i.e. number of occurrences) descending order. Pulp Fiction (1994) is the movie with the highest number of ratings.

```
## # A tibble: 10,676 x 3
##    title                                                     count edx_mov_RMV
##    <chr>                                                     <int>       <dbl>
##  1 Pulp Fiction (1994)                                       31336        4.16
##  2 Forrest Gump (1994)                                       31076        4.01
##  3 Silence of the Lambs, The (1991)                          30280        4.21
##  4 Jurassic Park (1993)                                      29291        3.66
##  5 Shawshank Redemption, The (1994)                          27988        4.46
##  6 Braveheart (1995)                                         26258        4.08
##  7 Terminator 2: Judgment Day (1991)                         26115        3.93
##  8 Fugitive, The (1993)                                      26050        4.01
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (197~ 25809        4.22
## 10 Batman (1989)                                             24343        3.39
## # ... with 10,666 more rows
```

The second one shows the data in a movie RMV descending order.

Blue Light, The (Das Blaue Licht) (1932) is one of the movies with the best RMV.

```
## # A tibble: 10,676 x 3
##    title                                                  count edx_mov_RMV
##    <chr>                                                  <int>       <dbl>
##  1 Blue Light, The (Das Blaue Licht) (1932)                   1        5
##  2 Constantine's Sword (2007)                                 1        5
##  3 Fighting Elegy (Kenka erejii) (1966)                       1        5
##  4 Hellhounds on My Trail (1999)                              1        5
##  5 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                        2        5
##  6 Shadows of Forgotten Ancestors (1964)                      1        5
##  7 Sun Alley (Sonnenallee) (1999)                             1        5
##  8 Human Condition II, The (Ningen no joken II) (1959)        3        4.83
##  9 Human Condition III, The (Ningen no joken III) (1961)      4        4.75
## 10 Who's Singin' Over There? (a.k.a. Who Sings Over There) (K~     4        4.75
## # ... with 10,666 more rows
```

An important insight that can be noticed is that Blue Light , The (Das Blaue Licht) (1932), for example, received a five stars score from *only one* user. It means that there are movies with high score rating (or with low score rating) that have been rated very few times. This aspect modify the calculation of least squares and has to be taken into account when the ML algorithm is trained.

The pictorial representation of this aspects can be found in Fig.3 and Fig. 4.

Fig.3 highlights the wide difference in number of occurrences for different movies (left part).

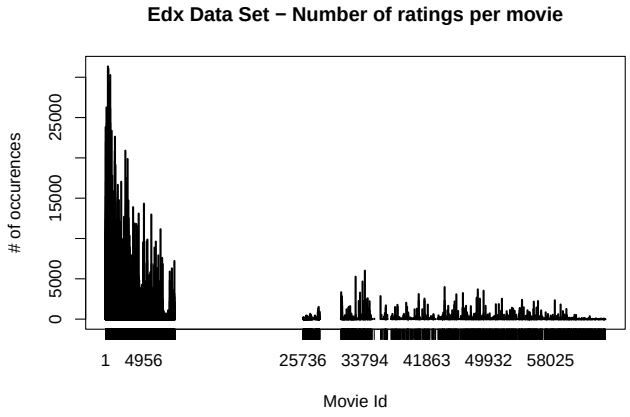

Figure 3: MovieId vs number of occurences.

Fig.4 shows an "average zone," between the red and blue line, in which the major part of the ratings fall.

If the situation is depicted in a 3D plot, it is clear as the most liked movies have been usually rated more times than the others (left up corner in Fig. 5).

Users

A similar approach can be implemented when the *rating* is analyzed according the user typologies.

There are users that seem to know only the 5 star rating

```
## # A tibble: 69,878 x 3
##    userId count edx_user_RMV
##     <int> <int>        <dbl>
##  1      1    21            5
##  2   1686    21            5
```

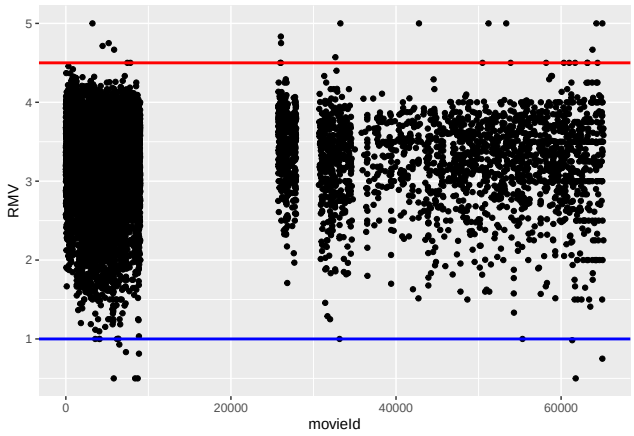

Figure 4: MovieId vs RMV. The blue and red line define a zone where usually the user rating falls. As can been noticed there are few movies outside the borders

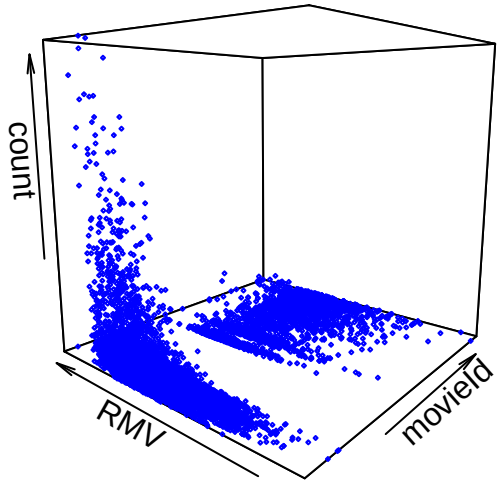

Figure 5: 3D scatter plot. The axis are RMV, movieId and count

```
##  3   7984    18           5
##  4  11884    19           5
##  5  13027    31           5
##  6  13513    19           5
##  7  13524    19           5
##  8  15575    28           5
##  9  18965    50           5
## 10  22045    22           5
## # ... with 69,868 more rows
```

and others than only distribute 0.5 points of ranking

```
## # A tibble: 69,878 x 3
##    userId count edx_user_RMV
##     <int> <int>        <dbl>
##  1  13496    19          0.5
##  2  48146    27          0.5
##  3  49862    18          0.5
##  4  62815    19          0.5
```

```
##  5  63381    20        0.525
##  6   6322    19        0.816
##  7   8920    19        0.974
##  8   3457    17        1
##  9  24176   138        1
## 10  24490    19        1
## # ... with 69,868 more rows
```

At the same time, there are users that are able to rate more than six thousand movies, and other that have rated just some titles.

```
## # A tibble: 69,878 x 3
##    userId count edx_user_RMV
##     <int> <int>        <dbl>
##  1  59269  6637         3.26
##  2  67385  6376         3.20
##  3  14463  4637         2.41
##  4  68259  4056         3.56
##  5  27468  4018         3.83
##  6  19635  3740         3.50
##  7   3817  3736         3.11
##  8  63134  3390         3.27
##  9  58357  3318         3.00
## 10  27584  3139         3.00
## # ... with 69,868 more rows
```

The next plots are depicted to show these aspects.

In Fig. 6 it can appreciated rating activity per user.

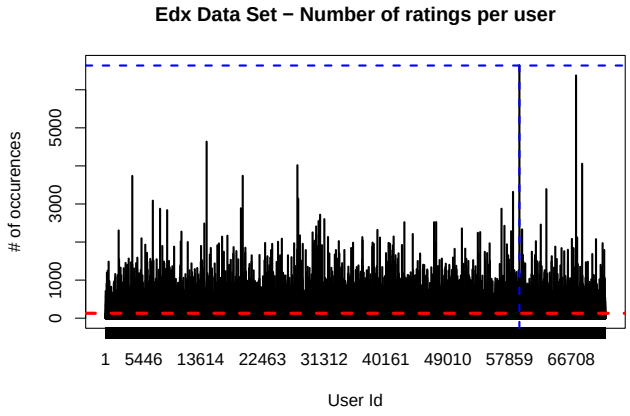

Figure 6: UserId vs number of occurences.

The most active user (blue dashed line in Fig. 6) is:

```
## [1] 59269
```

The number of rating for the user 59269 is:

```
## [1] 6637
```

Fig.7 composes the info about the user effect showing the RMV per user and the associated *count* value (the height of the cylinders).

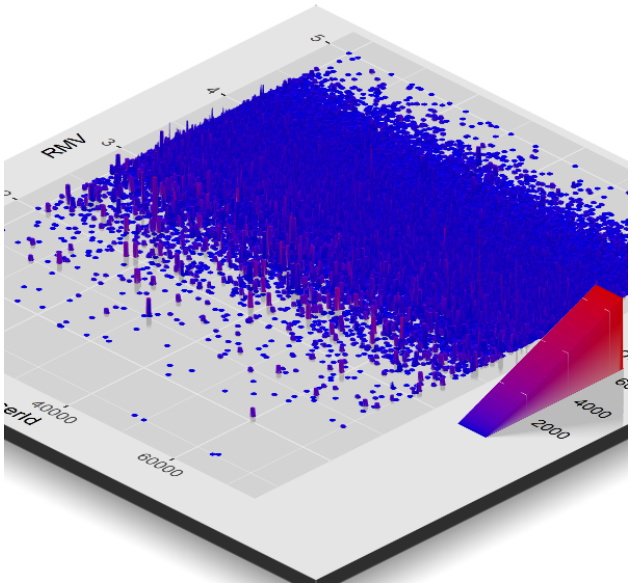

Figure 7: UserId vs RMV

Genres

The column *genres* in *edx* data set is characterized by a movie multi-genres definition. The different genres that appear for each movie have not divided for developing this report.

The plots in Fig. 8 show that effectively there are genres more rated than others.

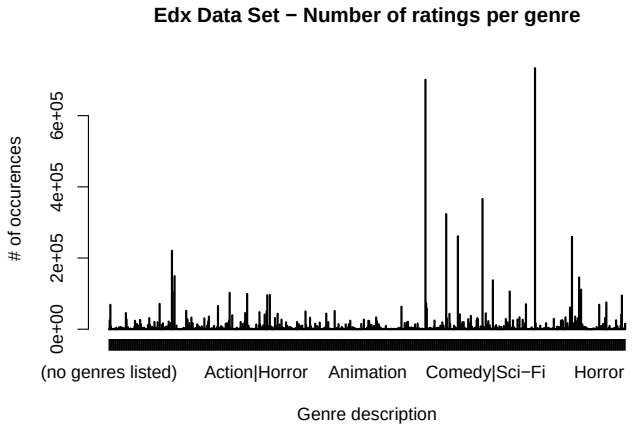

Figure 8: Not all the genres are shown in the x-axis

Some of the genres with the best (or worst) score ratings (e.g *Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi* or *Documentary|Horror*) have been rated by very few users.

The following tables display some of the values are needed to document this aspect.

Arranging the RMV in ascending order it is possible to observe as *Action|Animation|Comedy|Horror* , for example, has been rated only two times.

```
## # A tibble: 797 x 3
##    genres                                  count   RMV
```

```
##    <chr>                                  <int> <dbl>
##  1 Documentary|Horror                       655  1.47
##  2 Action|Animation|Comedy|Horror             2  1.5
##  3 Comedy|Film-Noir|Thriller                 23  1.59
##  4 Action|Horror|Mystery|Thriller           325  1.59
##  5 Action|Drama|Horror|Sci-Fi                 5  1.6
##  6 Adventure|Drama|Horror|Sci-Fi|Thriller   220  1.73
##  7 Action|Children|Comedy                   523  1.88
##  8 Action|Adventure|Drama|Fantasy|Sci-Fi     61  1.89
##  9 Children|Fantasy|Sci-Fi                   57  1.89
## 10 Action|Horror|Mystery|Sci-Fi              23  1.91
## # ... with 787 more rows
```

At the same time in the genre category *Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi* only three ratings can be found.

```
## # A tibble: 797 x 3
##    genres                                  count   RMV
##    <chr>                                   <int> <dbl>
##  1 Animation|IMAX|Sci-Fi                       7  4.71
##  2 Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi  3  4.33
##  3 Drama|Film-Noir|Romance                  2985  4.31
##  4 Action|Crime|Drama|IMAX                  2333  4.29
##  5 Animation|Children|Comedy|Crime          7183  4.28
##  6 Film-Noir|Mystery                        5993  4.24
##  7 Film-Noir|Romance|Thriller               2420  4.23
##  8 Crime|Film-Noir|Mystery                  3989  4.23
##  9 Crime|Film-Noir|Thriller                 4818  4.20
## 10 Crime|Mystery|Thriller                  26774  4.20
## # ... with 787 more rows
```

In order to stay in the average rating range, one could take into account, for example, the movies with more than one hundred thousand ratings.

Plotting the RMV of this subgroup it is possible to visualize how some genres are on average better rated than others.

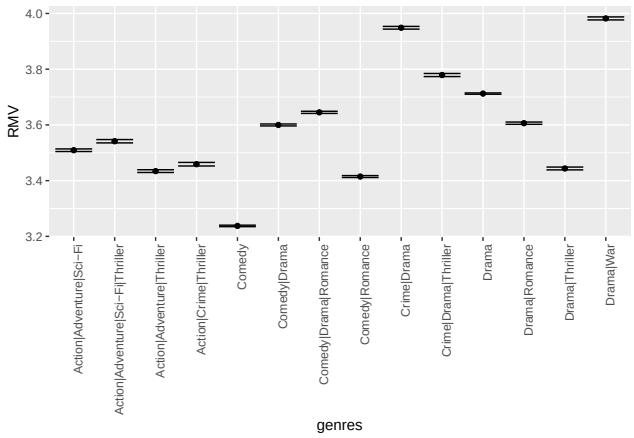

Figure 9: Genres and RMV

In the subgroup showed in Fig.9 *Comedy* has the lowest RMV and *Drama.War* the highest. Fig. 10 summarizes the relation RMV vs Genres filling the value with the number of occurences (height of the cilynders).

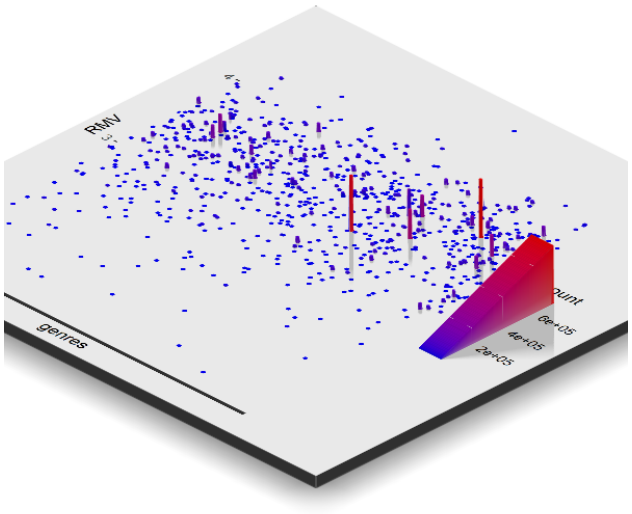

Figure 10: Genres vs RMV

Time

In order to address the question about the effect that can have the variable time on the rating, the relation between year of movie rating, year movie release and ratings are studied here.

Fig. 11 shows how the year of movie rating has some little effect on the rating scores. The time period is divided in weeks. After a descending phase, around the year 2005 an slight incremental tendency of RMV can be noticed.

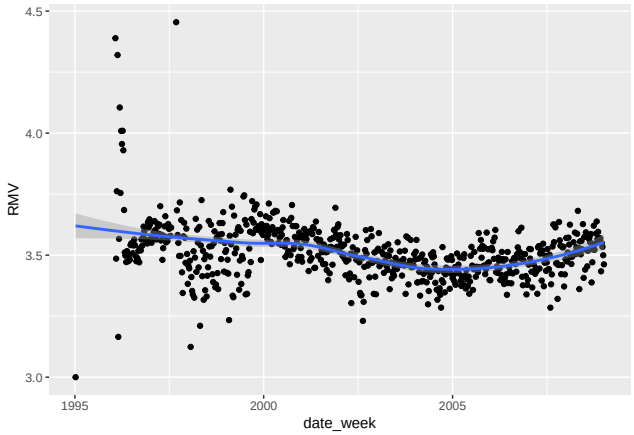

Figure 11: Year of movie rating (units: week) vs RMV

Fig.12 shows clearly that the RMV varies widely according to the movie release year, from 1930 to 2010. A long descending phase starts from the year 1950. It could be interesting to have the data after the year 2010 to understand if this tendency is continuing or not.

Only the second kind of time effect have been analysed in the next sections.

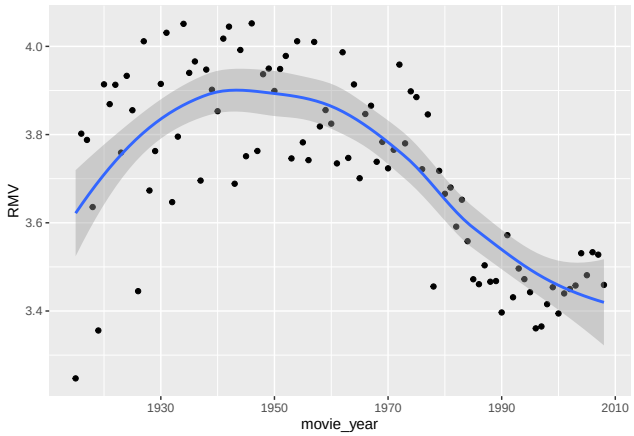

Figure 12: Movie release year vs RMV

Modeling approaches

In order to find the best ML model to implement, two approaches have been chosen.

The first one is an expansion of the method proposed during the final ML course in HarvardX's multi-part **Data Science Professional Certificate** series, which will be named *DSexp*.

The second approach is based on Matrix Factorization method (hereafter MF).

As shown in the next code chunk, the provided *edx* data set has been partitioned in two others sub data sets (90:10) in order to train and test the models *before* to use them on the *validation* data set.

```
options(digits = 10)

### SPLITTING THE EDX DATA SET INTO TRAIN AND TEST SETS

set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)

train_edx_set <- edx[-test_index,]
test_edx_set <- edx[test_index,]

test_edx_set <- test_edx_set %>%
  semi_join(train_edx_set, by = "movieId") %>%
  semi_join(train_edx_set, by = "userId")

### Does these data sets contain NA values? Answer: NO
sum(is.na(train_edx_set))  ## 0
sum(is.na(test_edx_set))   ## 0

### Does these data sets contain repeated rows?
### No, they are clean

sum(duplicated(train_edx_set))  ### 0
sum(duplicated(test_edx_set))   ### 0
```

Loss function (RMSE)

To compare the various models with each other a Loss Function based on the Root Mean Square Error (RMSE) is adopted:

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

Here the $y_{u,i}$ is the rating for movie i by user u and the prediction is denoted with $\hat{y}_{u,i}$.

DSexp Model + Regularization

Given a distribution of observed *ratings* $Y_{u,i}$, How can we model them?

At the very beginning it could be chosen only one common rating score value for all movies, as if the users variability and the other aspects could be negligible.

What is this value? As highlighted during the ML course by the instructor we need to start modeling our reasoning.

One way to model it is the following:

$$Y_{u,i} = \mu + \varepsilon_{u,i}$$

where μ is the "common value" we mentioned before and the $\varepsilon_{i,u}$ are the errors that explicate the deviation from the reality (i.e. the distance between $Y_{u,i}$ and μ).

From statistics deductions we know that the best estimate of μ that minimizes the RMSE is its least squares estimate, that is the average of overall rating.

Throughout the data analysis a lot of insights have been gained and the role of movie and genre type, users attitudes and year movie release has been highlighted. For the sake of the reader and to make the report smoother only the code for the final model *DSexp* and two tables summarizing the step-wise results (i.e. RMSEs) will be shown.

The rest of the code can be found HERE

All the aforementioned effects can be summarized in this model:

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_t + \varepsilon_{u,i}$$

Where the b_x are the named "bias" or deviations from the average and the subscripts i, u, g, t represent the movie, user, genre and time effects respectively.

Since it has been observed during the Data Analysis process that there are a few number of aspects that can determine large estimates of b_x (e.g. a movie with 5 stars rated only one time) a *Regularization* has been applied. It means that a penalty term has been introduce in the least square error equation to minimize it and control the variability of effect sizes.

For example, regularization for estimating movie and user effects can be done minimizing this equation:

$$\sum_{u,i}\left(y_{u,i} - \mu - b_i - b_u\right)^2 + \lambda\left(\sum_i b_i^2 + \sum_u b_u^2\right)$$

where λ is the tuning parameter or penalty term. The larger is λ, the more is the shrinking.

Matrix Factorization (MF)

Since an example of collaborative filtering wanted to be considered for the developing of a recommender system a Matrix Factorization (Aggarwal 2016) approach has been applied. Principally, it has been implemented in order to understand its performance.

As said in previous paragraphs, the Rating matrix R for the *edx* data set is characterized by 10677(titles)*69878(users) = 7.46087406e+8 entries. In the "movie space" defined by the different titles each users has his/her own preferences according the movie's features (genres,actors etc.) that define a specific user pattern. What is useful here is to find the users that show similar patterns or groups of movies having similar rating patterns. To solve this the R (M x N dimensioned) matrix can be *factorized* in two smaller rectangular matrix U (M x s dimensioned) and I (s x N dimensioned), the *factors*, that contain information about the aforementioned "similarities." The letter s is what is called *number of latent factors* (or features), that for the movies, for example, could be genre, actor etc.

Mathematically speaking we can write this as:

$$\mathbf{R} = \mathbf{U} \cdot \mathbf{I}$$

Where R is characterized by the ratings $y_{u,i}$:

$$\mathbf{R} = \begin{pmatrix} y_{1,1} & \cdots & y_{1,10677} \\ y_{2,1} & \cdots & y_{2,10677} \\ & \vdots & \\ y_{68978,1} & \cdots & y_{68978,10677} \end{pmatrix}$$

the matrix

$$\mathbf{U}$$

is

$$\mathbf{U} = \begin{pmatrix} u_{1,1} & \cdots & u_{1,s} \\ u_{2,1} & \cdots & u_{2,s} \\ & \vdots & \\ u_{68978,1} & \cdots & u_{68978,s} \end{pmatrix}$$

and

$$\mathbf{I}$$

is:

$$\mathbf{I} = \begin{pmatrix} i_{1,1} & \cdots & i_{1,10677} \\ i_{2,1} & \cdots & i_{2,10677} \\ & \vdots & \\ i_{s,1} & \cdots & u_{s,10677} \end{pmatrix}$$

A deep description of Matrix Factorization approach for building a ML model is beyond the scope of this report.

To implement a MF the *recosystem* package has been used (Qiu et al. 2021).

Strictly speaking here will be used a Matrix Factorization method with Stochastic Gradient Descent (SGD) to optimize the learned parameters during the tuning phase.

Results

DSexp model + Regularization

RMSE variation has been calculated coding progressively the various effects: movie, user, genre and time.

(See the code in this LINK if you want see how it was done)

Firstly, the deviations b_x have been calculated training the model on the *"train_edx_set"* . With the obtained values of b_x a prediction of the ratings has been performed on the *"test_edx_set"* and the RMSE computed.

The following code for the final model is illustrative and encloses all the coding steps:

```
############################################################################
######################### AVOIDING OVERTRAINING ############################
############################################################################

#### In order to avoid over-training and performing full cross-validation
#### during the regularization phase, the train_edx_set has been split
#### again in other two sub-sets.

set.seed(2, sample.kind="Rounding")

test_index <- createDataPartition( y = train_edx_set$rating,
                                   times = 1,
                                   p = 0.1,   #### 10% for the test data set
                                   list = FALSE )

train_edx_set_for_tuning <- train_edx_set[-test_index,]
temp <- train_edx_set[test_index,]

test_edx_set_for_tuning <- temp %>%
  semi_join(train_edx_set_for_tuning, by = "movieId") %>%
  semi_join(train_edx_set_for_tuning, by = "userId")


# Add rows removed from validation set back into train_edx_set_for_tuning, set
removed <- anti_join(temp, test_edx_set_for_tuning)
train_edx_set_for_tuning <- rbind(train_edx_set_for_tuning, removed)

#####################################################################
##### REGULARIZED MOVIE + User + TIME + Genre Effects ###########
#####################################################################

#### TUNING

lambdas <- seq(0, 10, 0.1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_edx_set$rating)
  b_i <- train_edx_set_for_tuning %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_edx_set_for_tuning %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
```

```
  b_t <- train_edx_set_for_tuning %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(movie_year)%>%
    summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+l))
  b_g <- train_edx_set_for_tuning %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_t, by="movie_year") %>%
    group_by(genres)%>%
    summarize(b_g = sum(rating - b_i - b_u - b_t - mu)/(n()+l))
  predicted_ratings_movieusertimegenre_reg <-
    test_edx_set_for_tuning %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "movie_year") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings_movieusertimegenre_reg, test_edx_set_for_tuning$rating))
})

#### VISUALIZING THE TUNING PROCESS
qplot(lambdas, rmses)
lambda <- lambdas[which.min(rmses)]
lambda  #### best lambda

#### USING THE BEST LAMBDA FOR TRAINING AND TESTING

b_i <- train_edx_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u <- train_edx_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
b_t <- train_edx_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(movie_year)%>%
  summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+lambda))
b_g <- train_edx_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_t, by="movie_year") %>%
  group_by(genres)%>%
  summarize(b_g = sum(rating - b_i - b_u - b_t - mu)/(n()+lambda))

predicted_ratings_movieusertimegenre_reg <-
  test_edx_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_t, by = "movie_year") %>%
```

```
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
  pull(pred)

#### CALCULATING THE RMSE
RMSE_i_u_t_g <- RMSE(predicted_ratings_movieusertimegenre_reg, test_edx_set$rating)
RMSE_i_u_t_g  ### 0.8636102103
```

To validate the selected model, the *validation* data set has been used and the RMSE is shown at the end of the code chunk:

```
#####################################################################
#################          VALIDATION          ####################
################# FINAL DSexp MODELS (M+U+T+G) ####################
#####################################################################

predicted_ratings_movieusertimegenre_reg_validation <-
  validation %>%  #### <<<<<
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_t, by = "movie_year") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
  pull(pred)

RMSE_validation <- RMSE(predicted_ratings_movieusertimegenre_reg_validation, validation$rating)
RMSE_validation  #### 0.8646581736
```

Matrix Factorization

Some simple steps are required to implement a MF by *recosystem* package (Qiu et al. 2021).

- Select from edx and validation sets the columns movieId, userId, rating.
- Write two tables in two .txt files as *recosystem* package needs.
- Create a r = Reco() object.
- Tuning the parameters.
- Training
- Predicting

```
#############################################################
############### MATRIX FACTORIZATION ########################
#############################################################

### Loading library recosystem
library(recosystem)

#### Instructions from the package website have been followed:
#### https://github.com/yixuan/recosystem

############## WORKING ON VALIDATION DATA SET ###################
#####################################################################

##########DATA FORMAT################################################

## From: https://github.com/yixuan/recosystem
```

```
## 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: user_index, item_index, rating

### The same train data set used for the final model will be used:

edxf <- train_edx_set %>% select(userId, movieId, rating)

## TESTING DATA FILE is similar to training data,
## but since the ratings in testing data are usually UNKWON,
## the rating entry in testing data file can be omitted,
## or can be replaced by any placeholder such as 0 or ?.

### Selecting only userId and movie Id from validation data set
### To compare the performance with the final model previously selected
validf <- validation %>%  select(userId, movieId)

### Ratings values from validation
ratings_val <- validation %>%  select(userId,moviedId,rating)


## Note: From version 0.4 recosystem supports two special
## types of matrix factorization: the binary matrix factorization (BMF),
## and the one-class matrix factorization (OCMF). BMF requires ratings to take
## value from {-1, 1}, and OCMF requires all the ratings to be positive.

## The rating provided here are all positive.

edxf   <- as.matrix(edxf)
validf <- as.matrix(validf)
write.table(edxf, file = "train.txt", sep = " ", row.names = FALSE,
            col.names = FALSE)
write.table(validf, file = "validationset.txt", sep = " ",
            row.names = FALSE, col.names = FALSE)

set.seed(1)

# From the recosystem manual:
# "The following function data_file() is used to specify the source of data in
# the recommender system. It is intended to provide the input argument of
# functions such as $tune(), $train(), and $predict()".

training_dataset <- data_file("train.txt")
validation_dataset <- data_file("validationset.txt")

## From recosystem manual: "Reco() returns an object of class "RecoSys"
## equipped with methods $train(), $tune(), $output() and $predict(),
## which describe the typical process of building and tuning model,
## exporting factorization matrices, and predicting results."
r=Reco()

## From recosystem manual: This method is a member function of class "RecoSys"
## that uses cross validation to tune the model parameters
```

```
opts = r$tune(training_dataset,
              opts = list(dim = c(10, 20, 30), ### dim, number of latent factors
              lrate = c(0.1,0.2), ##the learning rate, which can be thought of as
                                  ## the step size in gradient descent.
              costp_l1 = 0, ## L1 regularization cost for user factors
              costq_l1 = 0, ## L1 regularization cost for item factors
              nthread = 1,  ## number of threads for parallel computing
              niter = 10))  ## number of iterations

### L2 regularization cost left as the default 0.1

stored_prediction = tempfile()

### From the manual:  Function train() trains a recommender model. It will read
### from a training data source and create a model file at the specified location.
### The model file contains necessary information for prediction.

r$train(training_dataset,
        opts = c(opts$min,
                 nthread = 1,
                 niter = 20))

### From the manual: "Predicts unknown entries in the rating matrix.
### Prior to calling this method, model needs to be trained using member
### function $train()

r$predict(validation_dataset, out_file(stored_prediction))
print(r)
### Storing the values from validation set
validation_ratings <- read.table("validationset.txt", header = FALSE, sep = " ")$V3

### Scanning the temporary file containing the predictions
predicted_ratings <- scan(stored_prediction)
predicted_ratings

### Calculating the RMSE using the ratings from validation set
rmse_of_model_mf <- RMSE(validation$rating, predicted_ratings)
rmse_of_model_mf
```

The following tables summarize the RMSEs obtained on the test data set and finally on the validation data set:

```
## # A tibble: 7 x 2
##   method                                    RMSE
##   <chr>                                    <dbl>
## 1 Algorithm - Average                    1.060054
## 2 Algorithm - Movie effect               0.94296150
## 3 Regularized: Movie                     0.94293918
## 4 Movie + User effects                   0.86468429
## 5 Regularized: Movie + User              0.86413618
## 6 Regularized: Movie + User + Time       0.86383848
## 7 Regularized: Movie + User + Time + Genre 0.86361021
```

```
## # A tibble: 2 x 2
```

```
##   method                                          RMSE
##   <chr>                                          <dbl>
## 1 "Regularized Movie + User + Time + Genre" 0.86465817
## 2 "Matrix Factorization\""                  0.78611815
```

Conclusions

Developing a recommender system is not a simple task. Different techniques are nowadays available. In this report only two are presented: DSexp and MF.

If the approach described in the HarvardX PH125.8x Data Science: Machine Learning is adopted (i.e. Regularizing only Movie and User effects) the RMSE obtainable on a test data set is 0.8641361793.

This model has been expanded adding the genres and time effect. It was enough to reduce the RMSE at 0.8636102103.

Model based on Matrix Factorization showed the best performance. In fact, the associated RMSE is 0.7861181454.

It is my intention to use in the future other techniques such as PCA, SVD or Clustering, to explore the same data set and try to building a new ML model with other algorithms.

REFERENCES

Aggarwal, C. C. 2016. *Recommender Systems: The Textbook*. Springer International Publishing. https://books.google.com.au/books?id=GKjWCwAAQBAJ.

Cui, Boxuan. 2020. *DataExplorer: Automate Data Exploration and Treatment*. https://CRAN.R-project.org/package=DataExplorer.

Harper, F Maxwell, and Joseph A Konstan. 2015. "The Movielens Datasets: History and Context." *Acm Transactions on Interactive Intelligent Systems (Tiis)* 5 (4): 1–19.

Qiu, Yixuan, Chih-Jen Lin, Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and other contributors. See file AUTHORS for details. 2021. *Recosystem: Recommender System Using Matrix Factorization*. https://CRAN.R-project.org/package=recosystem.