PH125.9x Capstone Project 1: MovieLens

Chris-FR

29 janvier 2019

Introduction

For this first project of the EDX PH125.9x capstone course, we will be creating a movie recommendation system using the MovieLens dataset. We will use the **10M records** version of this dataset. As a large number of students, one of the issue of this project was the performance of my laptop (2003, 4Gb RAM) to process the `edx` dataset.
In this report, we will:
- do a quick data exploratory analysis,
- then build 4 different models based on the `edx` dataset using Azure ML and the Penalized Least Squares regularization method
- and to finish we will compute the RMSEs of a `validation` dataset.

Data exploration and visualization

Loading data

To load the data, we used the code provided by the course page. We modified it to use the **fread** method of the **data.table** package.

The `edx` dataset has **9000055** rows and **6** columns.
The `validation` dataset has **999999** rows and **6** columns.

Movie genres dataset

We create a `movieGenres` matrix dataset containing the movieId as rows and genres as columns. We delete the "timestamp" and "title" columns from the `edx` and `validation` datasets.

```
# create the movieGenres dataset, containing the movieId as rows and the genres as columns
movieGenres <- edx %>% select(movieId, genres) %>%
    unique() %>%
    separate_rows(genres, sep = "\\|") %>%
    mutate(val=1) %>%
    spread(genres, val, fill=0)
# drop the "timestamp", "title" columns from edx and validation
edx$timestamp <- NULL
edx$title <- NULL
# we also drop the "genres" column from validation
validation$timestamp <- NULL
validation$title <- NULL
validation$genres <- NULL

head(movieGenres)
```

```
##   movieId (no genres listed) Action Adventure Animation Children Comedy
## 1       1                  0      0         1         1        1      1
## 2       2                  0      0         1         0        1      0
## 3       3                  0      0         0         0        0      1
## 4       4                  0      0         0         0        0      1
## 5       5                  0      0         0         0        0      1
## 6       6                  0      1         0         0        0      0
```

```
##   Crime Documentary Drama Fantasy Film-Noir Horror IMAX Musical Mystery
## 1     0           0     0       1         0      0    0       0       0
## 2     0           0     0       1         0      0    0       0       0
## 3     0           0     0       0         0      0    0       0       0
## 4     0           0     1       0         0      0    0       0       0
## 5     0           0     0       0         0      0    0       0       0
## 6     1           0     0       0         0      0    0       0       0
##   Romance Sci-Fi Thriller War Western
## 1       0      0        0   0       0
## 2       0      0        0   0       0
## 3       1      0        0   0       0
## 4       1      0        0   0       0
## 5       0      0        0   0       0
## 6       0      0        1   0       0
```

To store the different RMSE results, we define a `rmse_results` tibble.

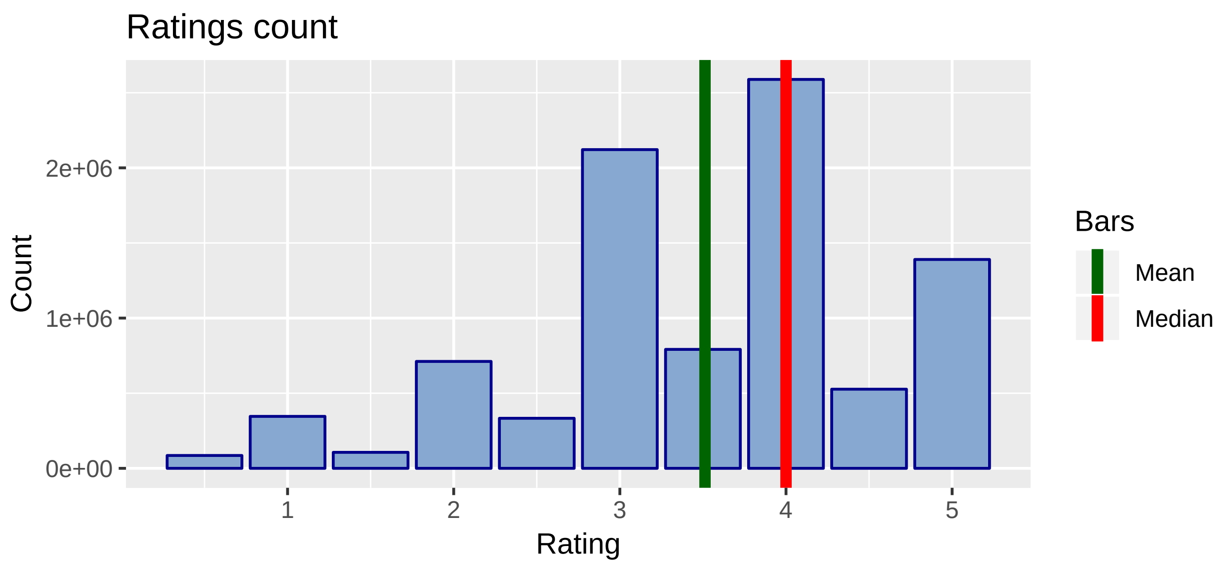
```
# creating the RMSE result table
rmse_results <- data_frame(method=character(0),
                                    RMSE = numeric(0))
```

Data visualization

This section displays some of the graphs used to analyse the data.
We use the following consolidated data:

```
stats_users <- edx %>%
    group_by(userId) %>%
    summarize(countRating=n(), meanRating=mean(rating), medianRating=median(rating))

stats_movies <- edx %>%
    group_by(movieId) %>%
    summarize(countRating=n(), meanRating=mean(rating), medianRating=median(rating))

stats_rating <- edx %>%
    group_by(rating) %>%
    summarize(countRating=n())

stats_genres <- edx %>%
    separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarize(countRating=n(), meanRating=mean(rating),
              medianRating=median(rating), countMovies=n_distinct(movieId))
```

Ratings distribution

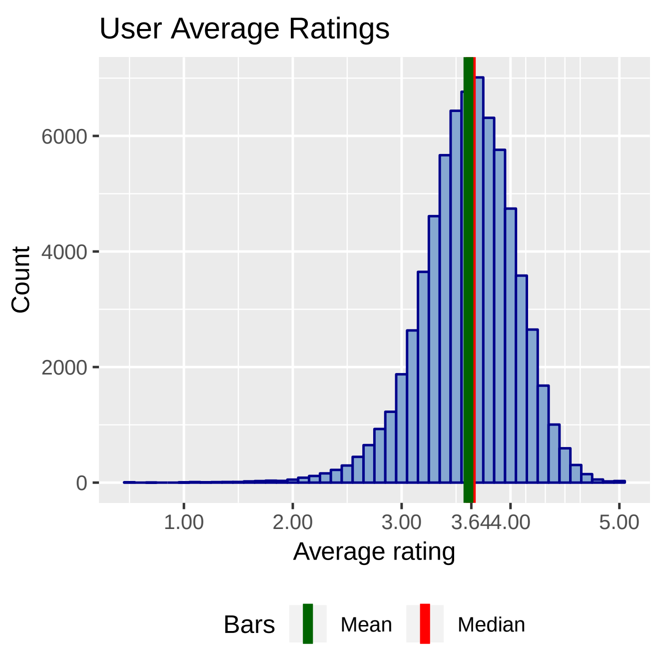


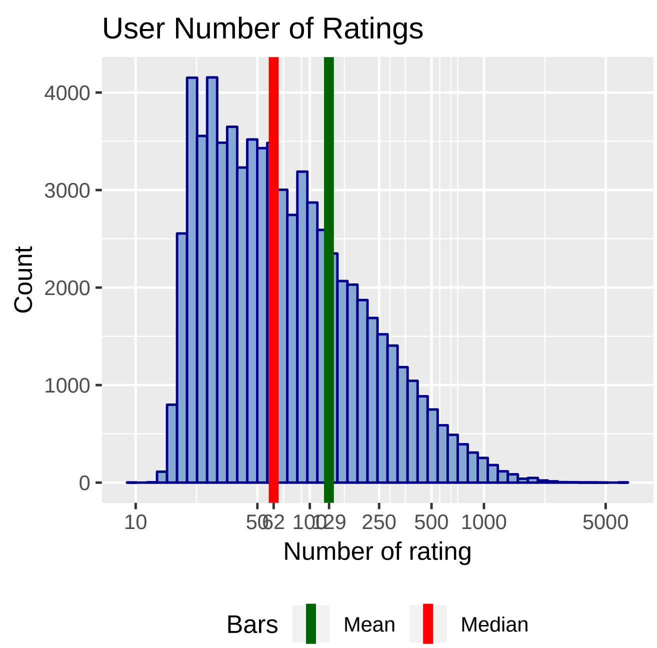
There are **9000055** ratings.

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

79.5% of these ratings are **whole** numbers. There is an unbalance between whole and half rating.

Users





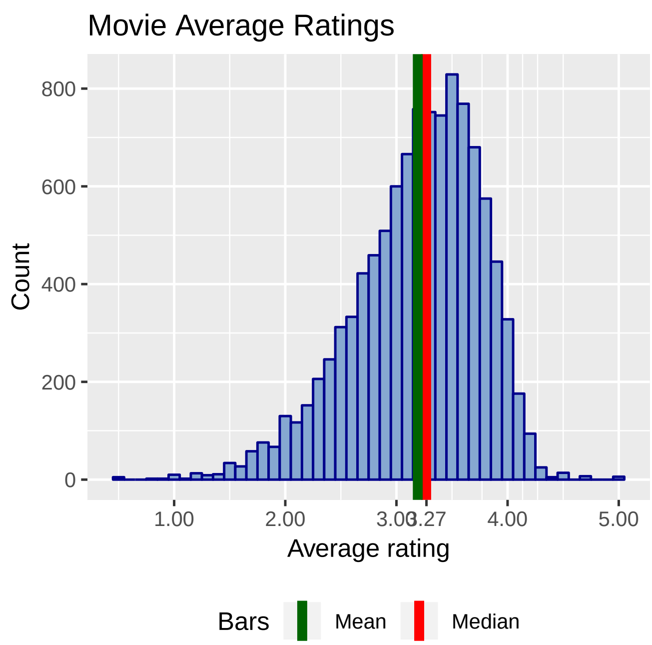
There are **69878** users.

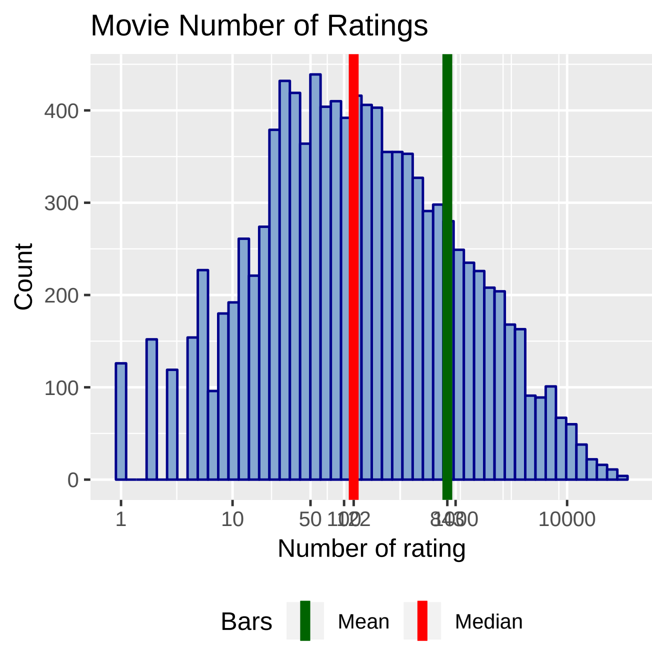
Number of ratings by users:

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

The mean number of rating by user is : **128.8**.
25% of the user have less than **32** ratings.
There are some very high value : **610** users rated more than 1000 movies, with a max number of rating of **6616**.

Movies



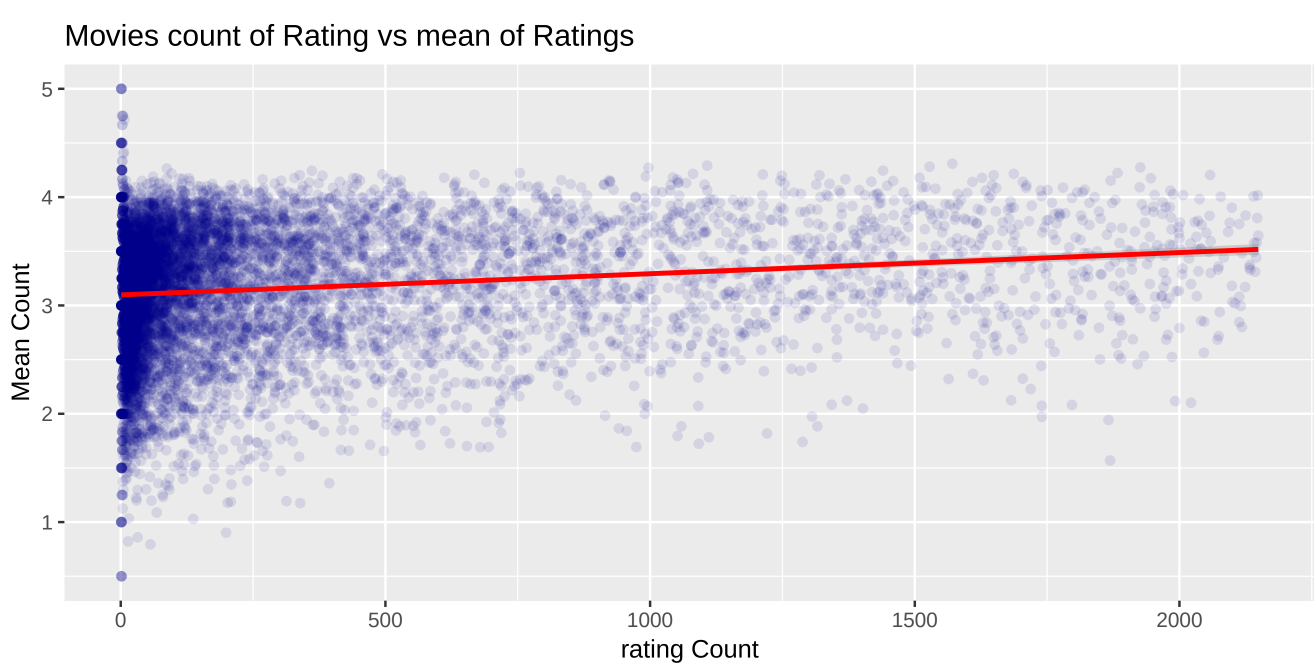


There are **10677** movies

Number of ratings by movies:

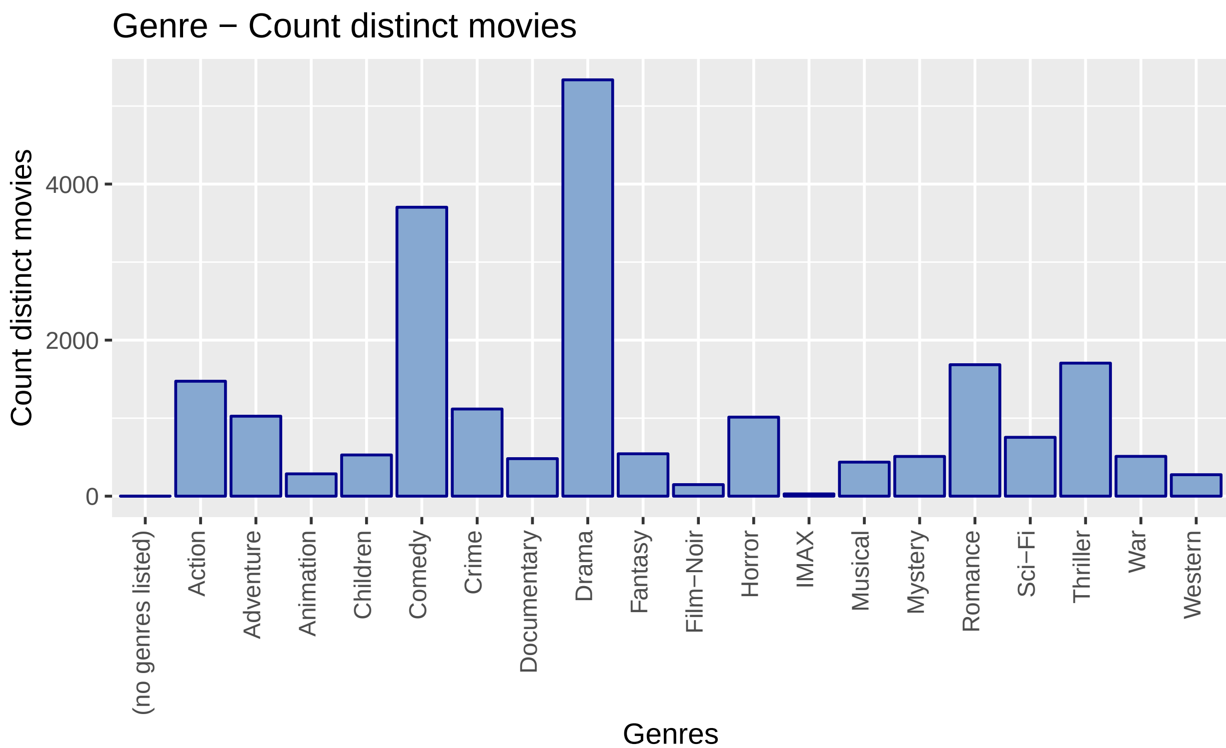
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    30.0   122.0   842.9   565.0 31362.0
```

The mean number of rating by movie is : **842.9**.
25% of the movies have less than **30** ratings. 25% of the movies have more than **565** ratings.
There are some very high value : **143** movies have more than 10000 movies, with a max number of rating of **31362**.

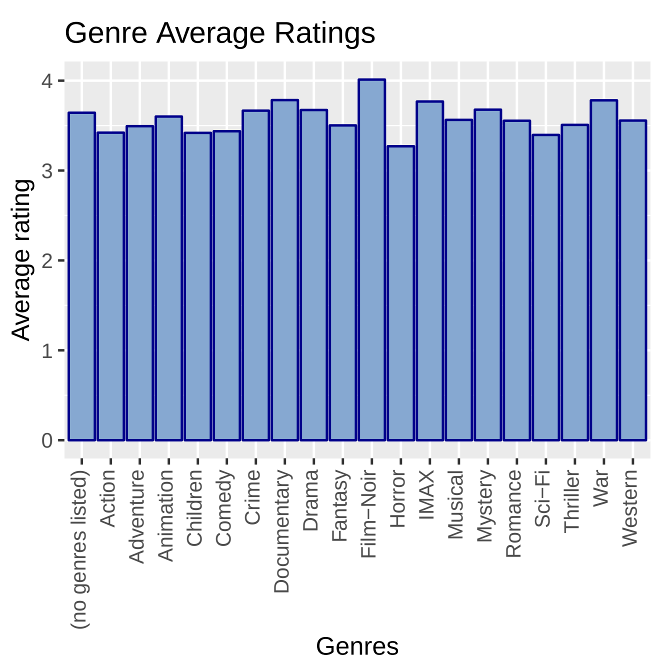


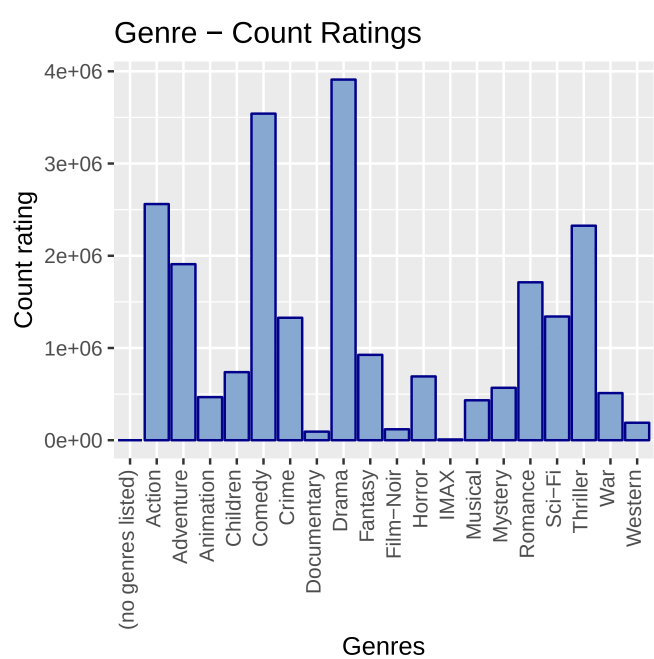
Movies with a high number of ratings seem to have higher means and less variabilities.

Genres



'Drama' and 'Comedy' are the most represented genres.





There is a genre effect.

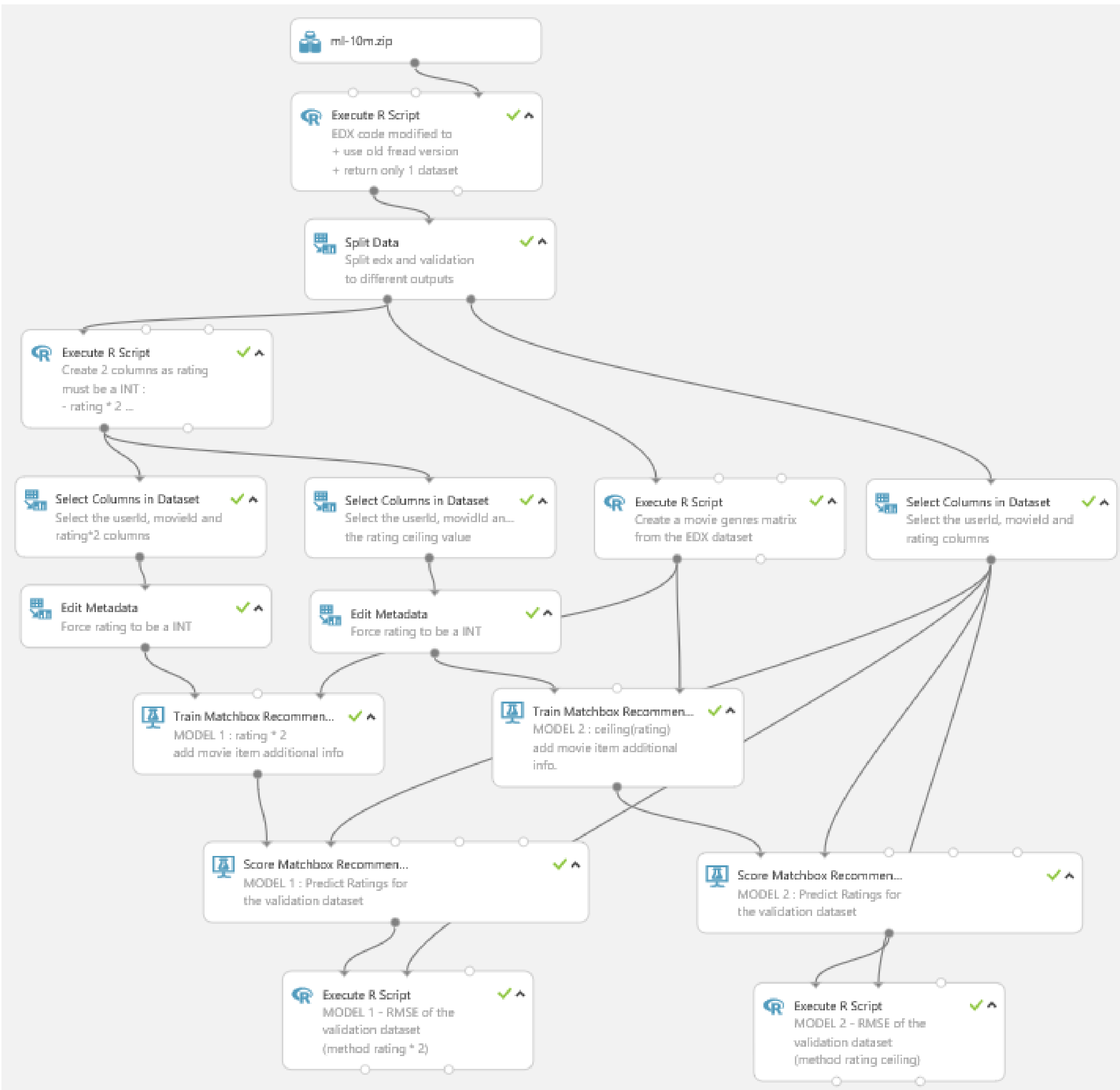
Data analysis

Even if it was out of this course scope, I found intereting to test Azure ML studio (with R integration).

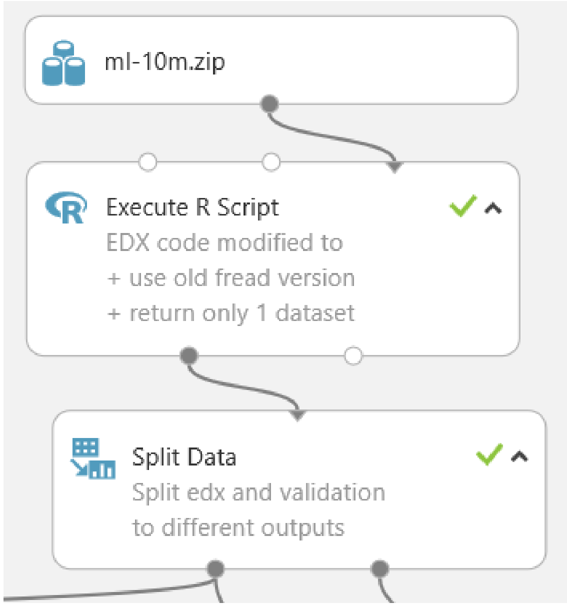
Azure ML - Matchbox recommender

Microsoft provides a free trial access to test the Azure ML studio (100 modules by experiment, 10 Gb storage, 1 hour per experiment, single node execution).
Azure ML has a recommender module : the Matchbox Recommender.

Here is the schema of the full experiment I developed:



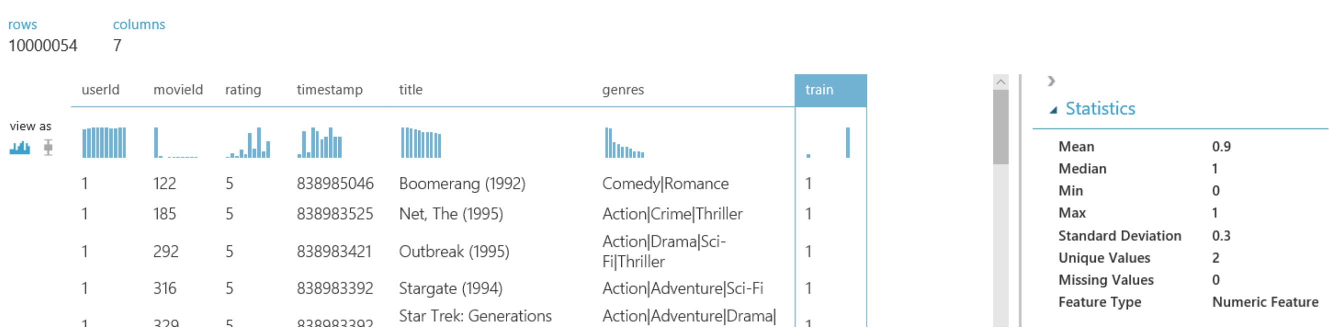
Loading the data



To load the data, an R script container containing the EDX code has been added.
3 main modifications have been done :
- the original ZIP file is not uploaded using HTTP. It was manually loaded as a dataset file. We used this dataset as the 3rd R script input. This input provides the unzipped file content in a SRC/ virtual directory.
- to speed up the `ratings` file loading, we used the **data.table fread** fonction. An issue was that in the available version of this package, the `text` parameter is unvailable and we have to separate the `readLines` fonction from the `fread` fonction.
- the R script module has only **one output**: to overcome this issue, I added a flag (train) and row binded the 2 datasets (edx and validation). We then used a **Split module** to split the `edx` and `validation` datasets using the **train** flag as split condition.

At the end of the R script, validation and edx datasets are concatenated, with an extra train column:

```
...
# use the old fread fonction : separate gsub and use paste.
ratings <- gsub("::", "\t", readLines("src/ml-10M100K/ratings.dat"))
ratings <- 'data.table'::fread(paste(ratings, collapse="\n"), col.names = c("userId", "movieId", "rating",
...
# add the train column and rbind the 2 datasets
edx$train<-1
validation$train<-0
data.set = rbind(edx, validation);
# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("data.set");
```



rows	columns
10000054	7

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

Statistics

Mean	0.9
Median	1
Min	0
Max	1
Standard Deviation	0.3
Unique Values	2
Missing Values	0
Feature Type	Numeric Feature

The left output of the **Split data** module is the `edx` dataset, the right output is the `validation` dataset.

Adding an item feature dataset

The 2nd and 3rd entries of the Matchbox recommender are a user features and item features datasets.
For the item features dataset, we create a `genre` matrix dataset from the `edx` main dataset.

```
# Map 1-based optional input ports to variables
edx <- maml.mapInputPort(1) # class: data.frame
library(dplyr)
library(tidyr)
# create the genres matrix
movieGenres <- edx %>% select(movieId, genres) %>%
    unique() %>%
    tidyr::separate_rows(genres, sep = "\\|") %>%
    mutate(val=1) %>%
    spread(genres, val, fill=0)
# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("movieGenres");
```

EDX : MovieLens ratings › Execute R Script › Result Dataset

rows	columns
10677	21

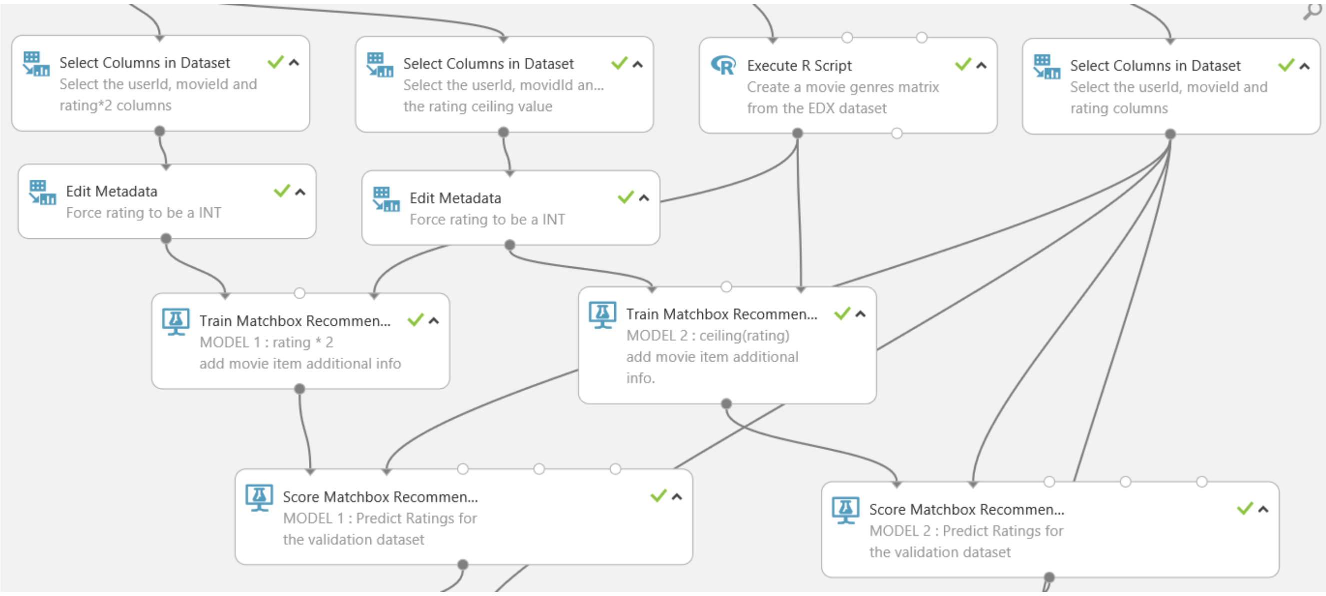
movieId	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir
1	0	0	1	1	1	1	0	0	0	1	0
2	0	0	1	0	1	0	0	0	0	1	0
3	0	0	0	0	0	1	0	0	0	0	0

Train the Matchbox recommender

The 'Train Matchbox Recommender' module uses a dataset of user / item / rating entries. It returns a trained Matchbox recommender. We can then use the trained model in the Score Matchbox Recommender module to generate recommendations, find related users, find related items, compute expected ratings .

The rating of the user / item / rating dataset must be an `integer`.
I decided to test 2 cases (model 1 and 2):
- the ratings are multiplied by 2 (ratings from 1 to 10) - we keep the unbalance caused by the half notation numbers;
- the ratings are rounded using the R `ceiling` function.

```
# Map 1-based optional input ports to variables
dataset1 <- maml.mapInputPort(1) # class: data.frame
# convert rating
dataset1$ratingbytwo <- dataset1$rating * 2
dataset1$ratingceiling <- ceiling(dataset1$rating)
# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("dataset1");
```



The train module parameters were set to default.

what was not done :
In this version of the experiment, the full edx dataset was used to train the models. I used split validation on another experiment but did not on this one.
I also did not try to tune the recommender parameters (the parameters are the default ones).

Regularized Movie + User Effect Model

The 3rd solution tested in this report is the `Regularized Movie + User Effect Model`.

Finding lambdas with 5-folds CV

We will compute 2 different lambdas as the penalties may not be the same between the movie and user effects :
- li : for the movie effect
- lu : for the user effect

To pick the best values of these 2 lambdas, we will do a **5-folds cross validation**. We will split `edx` is 5 folds, then take 4 folds as train set and 1 fold as test set and process every combination. We will then take the values having the better RMSE result average accross the 5 folds.

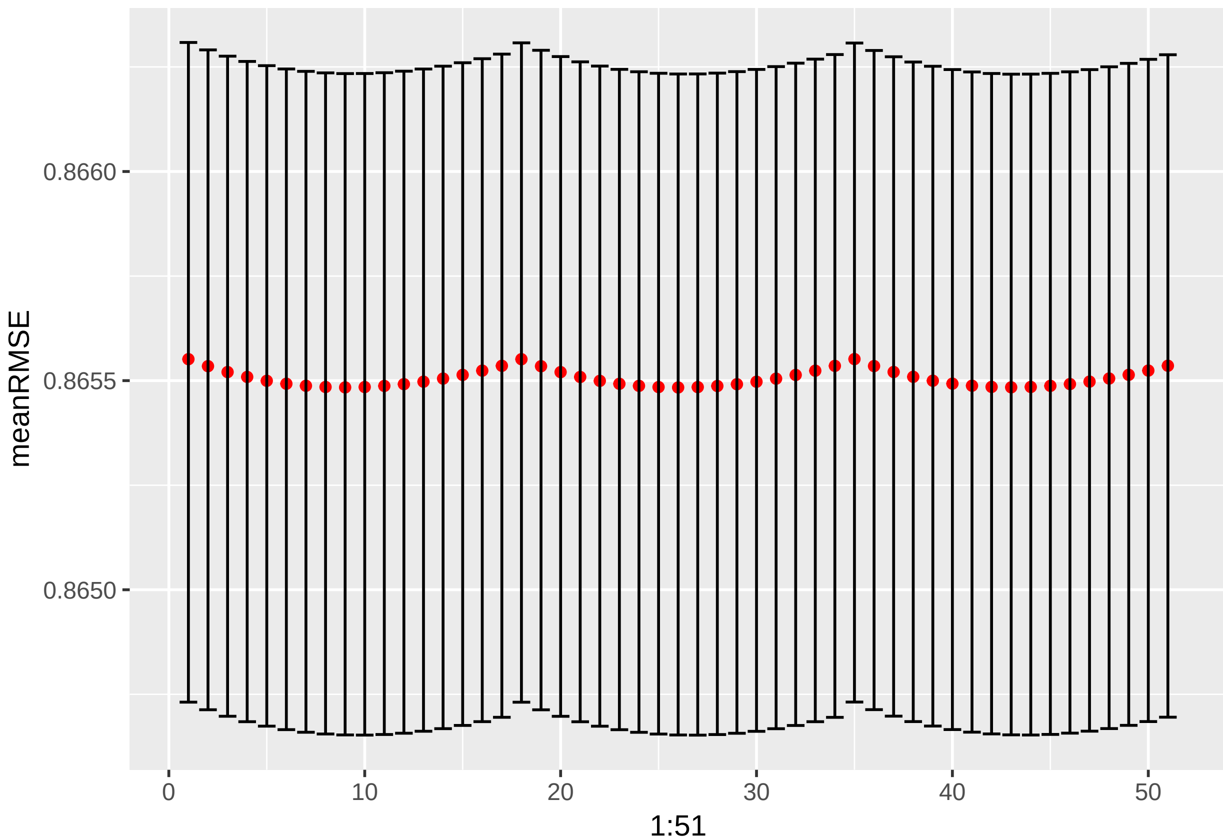
To avoid the cases where the test set contains a movie or a user not present in the train set, we will use the same code than the first edx / validation creation (using a temp dataset first).

```
# create a dataframe to store ecach result
df.result <- data.frame( "k" = integer(0),
                         "li" = numeric(0),
                         "lu" = numeric(0),
                         "RMSE" = numeric(0))


# we are doing k-folds cross validation
k<-5

# create the k folds with the CARET package
set.seed(1)
folds <- createFolds(edx$rating, k=k, returnTrain = FALSE)

# for the report compute only 3 to 7
lambdasI <- seq(3, 7, 0.25)
```

```r
lambdasU <- seq(3, 7, 0.25)

# for each of the 10 folds:
for (i in 1:k){

    # print(paste("Starting fold:" , i))

    # separate the train / test set using the fold #i
    train_set <- edx[-folds[[i]],]
    temp <- edx[folds[[i]],]

    # Make sure userId and movieId in the test set are also in train set
    test_set <- temp %>%
        semi_join(train_set, by = "movieId") %>%
        semi_join(train_set, by = "userId")

    # Add rows removed from test_set set back into train_set set
    removed <- anti_join(temp, test_set)
    train_set <- rbind(train_set, removed)
    rm(removed, temp)


    print(paste("Computing rmse for fold:" , i, dim(train_set)[1], dim(test_set)[1]))

    mu <- mean(train_set$rating)

    b_i <- train_set %>%
        group_by(movieId) %>%
        summarize(sum_b_i = sum(rating - mu), n=n())

    for(li in lambdasI){

        b_i$b_i <- b_i$sum_b_i/(b_i$n+li)

        b_u <- train_set %>%
            left_join(b_i, by="movieId") %>%
            group_by(userId) %>%
            summarize(sum_b_u = sum(rating - b_i - mu), n=n())

        for(lu in lambdasU){

            b_u$b_u <- b_u$sum_b_u / (b_u$n+lu)

            predicted_ratings <-
                test_set %>%
                left_join(b_i, by = "movieId") %>%
                left_join(b_u, by = "userId") %>%
                mutate(pred = mu + b_i + b_u) %>%
                .$pred

            myrmse <- RMSE(predicted_ratings, test_set$rating, na.rm = T)

            # print(paste(li, lu, myrmse))
            df.result[nrow(df.result) + 1,] = list(k=i, li=li, lu=lu, RMSE=myrmse)
        }
    }

}
```

```
## [1] "Computing rmse for fold: 1 7200090 1799965"
## [1] "Computing rmse for fold: 2 7200087 1799968"
## [1] "Computing rmse for fold: 3 7200080 1799975"
## [1] "Computing rmse for fold: 4 7200087 1799968"
## [1] "Computing rmse for fold: 5 7200083 1799972"
```

li	lu	minRMSE	meanRMSE	maxRMSE
4.5	5	0.8646528	0.8654837	0.8662331



Mean, Min and Max of 3 li and theirs lu over the 5-folds.

Compute the full model values (mu, b_i, b_u) with the best li and lu lambdas

Computing the model with li = **4.5** and lu = **5**.

```
mu <- mean(edx$rating)

b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+li))

b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lu))
```

Regularized Movie + User Effect + Genre effect Model

The last solution tested in this report is the `Regularized Movie + User Effect + Genre Effect Model`.

As the computation of the optimal lambdas is very slow on my laptop (+5 hours), we will use the previously calculated lambdas (`lu` and `li`) and just add a genre effect to the result (i.e. as if we start from the residuals of the last models).

The genre effect is given by the following formula:
$Y_{u,i} = \mu + b_i + b_u + \sum k = 1^K x_{u,i} \beta_k + \varepsilon_{u,i}$, with $x^k_{u,i} = 1$ if $g_{u,i}$ is genre k.

To simplify the model, we will group the movies using a **genre clustering**. A movie is now part on an unique cluster group (and not to multiple genres). The formula becomes:
$Y_{u,i} = \mu + b_i + b_u + b_{u,g} + \varepsilon_{u,i}$

For the penalty, we will use the same than the user one.

Genres cluster

```
colSums(movieGenres[2:21])
```

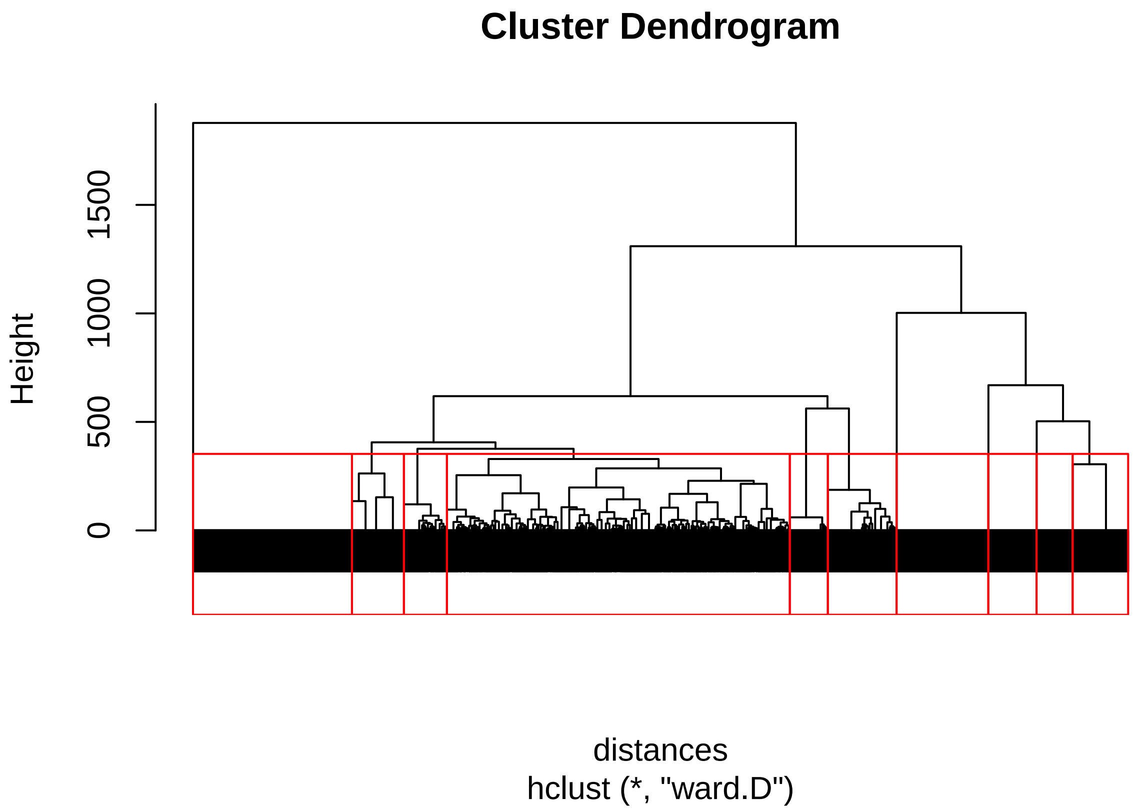
```
## (no genres listed)             Action          Adventure
##                  1               1473               1025
##          Animation           Children             Comedy
##                286                528               3703
##              Crime        Documentary              Drama
##               1117                481               5336
##            Fantasy          Film-Noir             Horror
##                543                148               1013
##               IMAX            Musical            Mystery
##                 29                436                509
##            Romance             Sci-Fi           Thriller
##               1685                754               1705
##                War            Western
##                510                275
```

As there is only one record of the "(no genres listed)", we remove this value.
We then compute the euclidean distance matrix and build out a Hierarchical Cluster.

```
movieGenres$`(no genres listed)` <- NULL

# compute the distance
# no need to normalize the values are they have the same scale
distances <- dist(movieGenres[2:20], method="euclidean")
# Hierarchical Cluster
clusterMovies <- hclust(distances, method="ward.D")
```

We split in 10 groups.

```
plot(clusterMovies, labels=FALSE)
clusterGroups <- cutree(clusterMovies, k = 10)
rect.hclust(clusterMovies, k=10, border="red")
```



Here is a view of the mean of each genres in each cluster:

```r
view_cluster_means <- sapply(1:10, function(x){round(colMeans(movieGenres[clusterGroups==x,2:20]),2)})

view_cluster_means #%>% knitr::kable()
```

```
##             [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## Action      0.32  0.0    0 0.09    0 0.00    0 0.00 0.28     0
## Adventure   0.24  0.0    0 0.02    0 0.00    0 0.01 0.16     0
## Animation   0.07  0.0    0 0.00    0 0.00    0 0.00 0.00     0
## Children    0.13  0.0    0 0.00    0 0.00    0 0.00 0.00     0
## Comedy      0.33  1.0    1 0.10    0 0.00    0 0.06 0.11     1
## Crime       0.21  0.0    0 0.04    0 0.46    0 0.01 0.01     0
## Documentary 0.01  0.0    0 0.00    0 0.00    0 1.00 0.03     0
## Drama       0.35  0.4    0 0.11    1 0.79    1 0.05 0.76     1
## Fantasy     0.14  0.0    0 0.00    0 0.00    0 0.00 0.01     0
## Film-Noir   0.04  0.0    0 0.00    0 0.00    0 0.00 0.00     0
## Horror      0.06  0.0    0 1.00    0 0.00    0 0.02 0.01     0
## IMAX        0.00  0.0    0 0.00    0 0.00    0 0.04 0.00     0
## Musical     0.11  0.0    0 0.00    0 0.00    0 0.00 0.01     0
## Mystery     0.11  0.0    0 0.10    0 0.00    0 0.00 0.01     0
## Romance     0.15  1.0    0 0.00    0 0.00    1 0.01 0.12     0
## Sci-Fi      0.16  0.0    0 0.17    0 0.00    0 0.00 0.02     0
## Thriller    0.24  0.0    0 0.36    0 0.74    0 0.00 0.07     0
## War         0.01  0.0    0 0.00    0 0.00    0 0.01 0.95     0
## Western     0.07  0.0    0 0.00    0 0.00    0 0.00 0.00     0
```

We save the pair : movieId / cluster.
Than we add the `gcluster` (genre cluster) to the `edx` and `validation` dataset. (this step will save us a JOIN each time).

```r
movieGenres$gcluster <- clusterGroups
movieGenres <- movieGenres %>% select(movieId, gcluster)
# clean up
rm(distances, clusterMovies, view_cluster_means, clusterGroups)
# add the cluster group to EDX and validation dataset
edx <- edx %>%
    inner_join(movieGenres, by = "movieId")
edx$genres <- NULL
head(edx)
```

```
##   userId movieId rating gcluster
## 1      1     122      5        2
## 2      1     185      5        1
## 3      1     292      5        1
## 4      1     316      5        1
## 5      1     329      5        1
## 6      1     355      5        1
```

```r
validation <- validation %>%
    inner_join(movieGenres, by = "movieId")
head(validation)
```

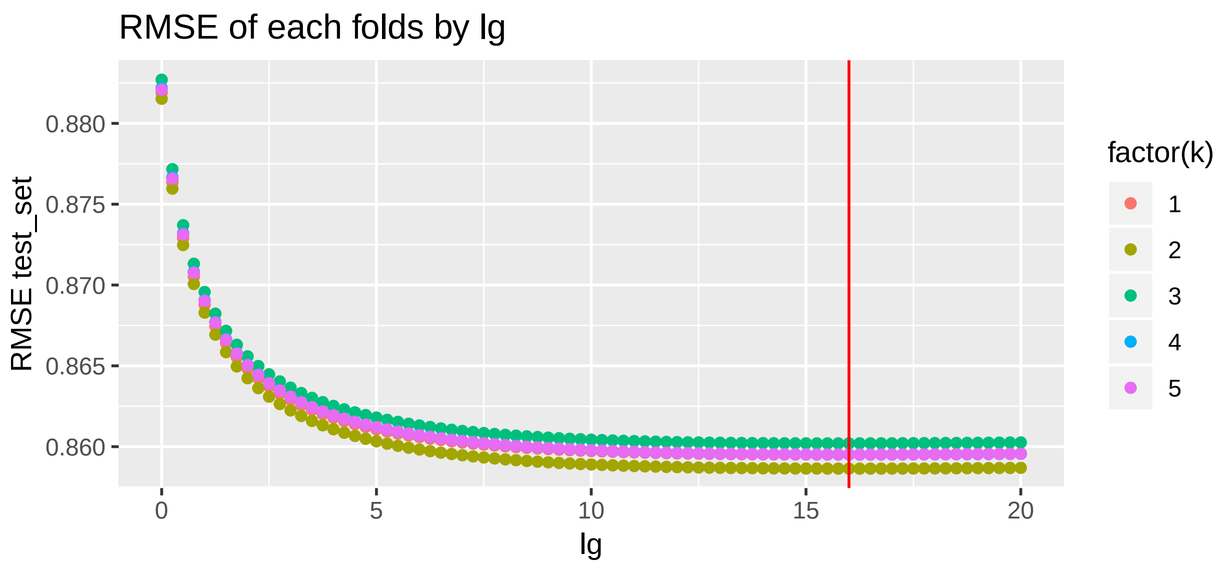
```
##   userId movieId rating gcluster
## 1      1     231      5        3
## 2      1     480      5        1
## 3      1     586      5        1
## 4      2     151      3        1
## 5      2     858      2        6
## 6      2    1544      3        4
```

Genres effect

We can now compute the genres effect on the `edx`dataset.
As stated in the introduction of this section, we will use the residuals of the previous model (we use li and lu values of the model 3 and do not recompute them) and we will use 5-folds cross validation to compute the penalization lambda of the genre (lg).

The best value for `lg` is:

lg	minRMSE	meanRMSE	maxRMSE
16	0.8586403	0.8594774	0.8602032



Compute the full model values (mu, b__i, b__u, b__g)

We can now compute the full model on the edx dataset:

```
print(paste('li=',li, 'lu=', lu, 'lg=' , lg))
```

```
## [1] "li= 4.5 lu= 5 lg= 16"
```

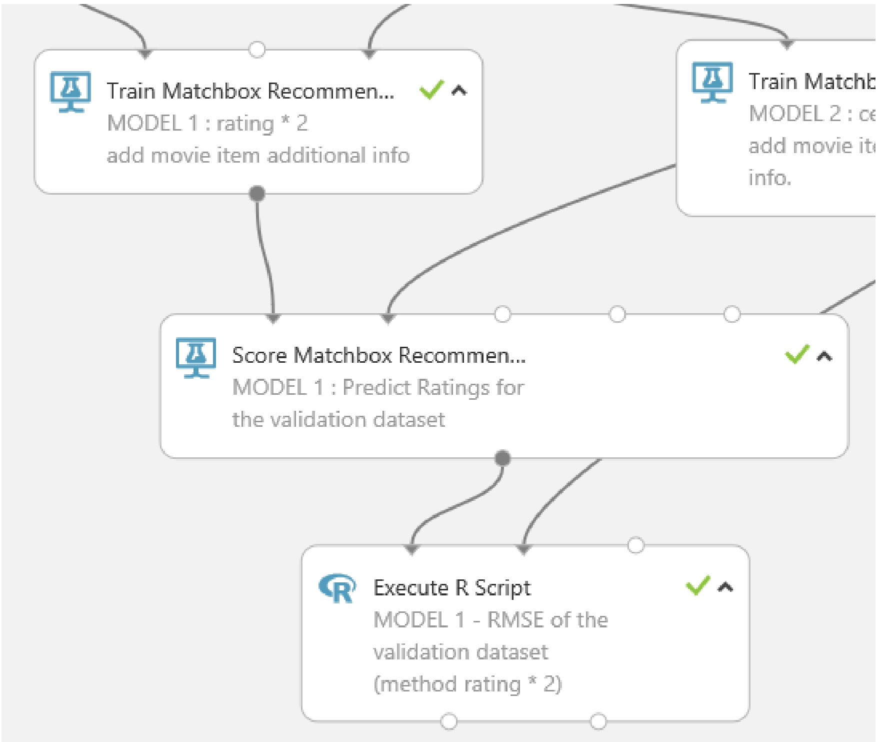
```
mu <- mean(edx$rating)

b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+li))

b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lu))

b_g <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(userId, gcluster) %>%
    summarize(b_g = sum(rating - b_i -b_u - mu)/(n()+lg))
```

Results

We can now try the finalized models on our validation dataset.

Azure ML (model 1 and 2)

To compute the final RMSE, we use the validation dataset and compute its predicted ratings using the Score Matchbox module.
The final RMSE is computed in the last R script module.



The RMSE is available in the output dataset.

```
# Map 1-based optional input ports to variables
pred <- maml.mapInputPort(1) # class: data.frame
val <- maml.mapInputPort(2) # class: data.frame

error <- val$rating - pred$Rating
myrmse <- sqrt(mean(error^2))

# Select data.frame to be sent to the output Dataset port
myreturn<- data.frame(RMSE=myrmse)
maml.mapOutputPort("myreturn");
```

method	RMSE
Azure ML Matchbox - ratings*2	1.015422
Azure ML Matchbox - ceiling	0.939061

Regularized Movie + User Effect Model (model 3)

```
predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u ) %>%
```

```
    .$pred

rmse_model3 <- RMSE(predicted_ratings, validation$rating)

rmse_results <- dplyr::bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = rmse_model3))

rmse_model3
```

```
## [1] 0.8648192
```

Regularized Movie + User Effect + Genre effect Model (model 4)

For the genre effect, we just need to ensure that if the user / genrecluster does not exist, the value of the average b_g is 0.

```
predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = c("userId","gcluster")) %>%
    mutate(b_g = replace_na(b_g, 0)) %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    .$pred

rmse_model4 <- RMSE(predicted_ratings, validation$rating)

rmse_results <- dplyr::bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect + Genre effect Model",
                                     RMSE = rmse_model4))
rmse_model4
```

```
## [1] 0.8579097
```

Conclusion

The 4 RMSEs for the `validation` dataset are the following:

method	RMSE
Azure ML Matchbox - ratings*2	1.0154220
Azure ML Matchbox - ceiling	0.9390610
Regularized Movie + User Effect Model	0.8648192
Regularized Movie + User Effect + Genre effect Model	0.8579097

As expected from the average of the cross validations, the model with the best result is the 'Regularized Movie + User Effect + Genre effect Model' with an RMSE of **0.8579**. This value is near the ones computed using 5-folds cross validation on the `edx` dataset: average= 0.8595 (min= 0.8586, max= 0.8602).

I wish i had a better hardware to test more methods (SVM,...).