MovieLens Data Analysis

Paolo De Piante

2019-10-26

1. Introduction

This is a report on MovieLens 10M dataset generated by the GroupLens research lab. The dataset can be downloaded from http://files.grouplens.org/datasets/movielens/ml-10m.zip. The study has been inspired by HarvardX & edX "Data Science: Capstone" MOOC. The gaol is to test some Machine Learing algorithms and approaches referred to the domain named "Recommandation Systems". RMSE will be used to evaluate how the model trained is far from predictions. The main challenge to address is to find the lowest RMSE to compare and to identify the better method to fill in the sparse matrix with the movie ratings closer to each single user profile. Just to know, the highest predicted movie ratings are used to suggest some movies not voted or never watched by a specific user yet. However, this last goal is not part of this study. Of the many methods available, the most suitable were chesen to carry out this study, based on their compatibility with the hardware used and the elaboration time.

Note: some code lines comes from lessons provided by Professor Rafael Irizarry in his Data Science Certification Program provided by edX.

2. Analysis

2.1 Data wrangling

I used the following library:

```
library(tidyverse)
library(lubridate)
library(Matrix)
library(recommenderlab)
library(Matrix.utils)
library(irlba)
library(recosystem)
library(knitr)
```

Basically two main data partitions will be used: "edx" to train and "validation" to validate the model. The data partition has been provided by edX "Data Science: Capstone" course mentioned in the introduction section. As follows I report the code provided:

```
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
```

```r
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip


dl <- tempfile()
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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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


# Validation set will be 10% of MovieLens data

set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]


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

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)
```

2.2 Data exploration

Exploring the edx dataset properties, the following points were considered: the dimension of the data matrix (users, movies) = [69878, 10677]; the data frame has 9000055 observations (number of ratings); the sparse matrix has many empty cells (movies not voted); ratings range goes from 0.5 to 5.

```r
# Exploring the data set edx
class(edx)
```

```
## [1] "data.frame"
```

```
head(edx)
```

```
##   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
```

```
# str(edx) # not reported
dim(edx)
```

```
## [1] 9000055       6
```

```
length(unique(edx$movieId))  # number of different movies
```

```
## [1] 10677
```

```
length(unique(edx$userId))   # number of differen users
```

```
## [1] 69878
```

```
summary(edx)
```

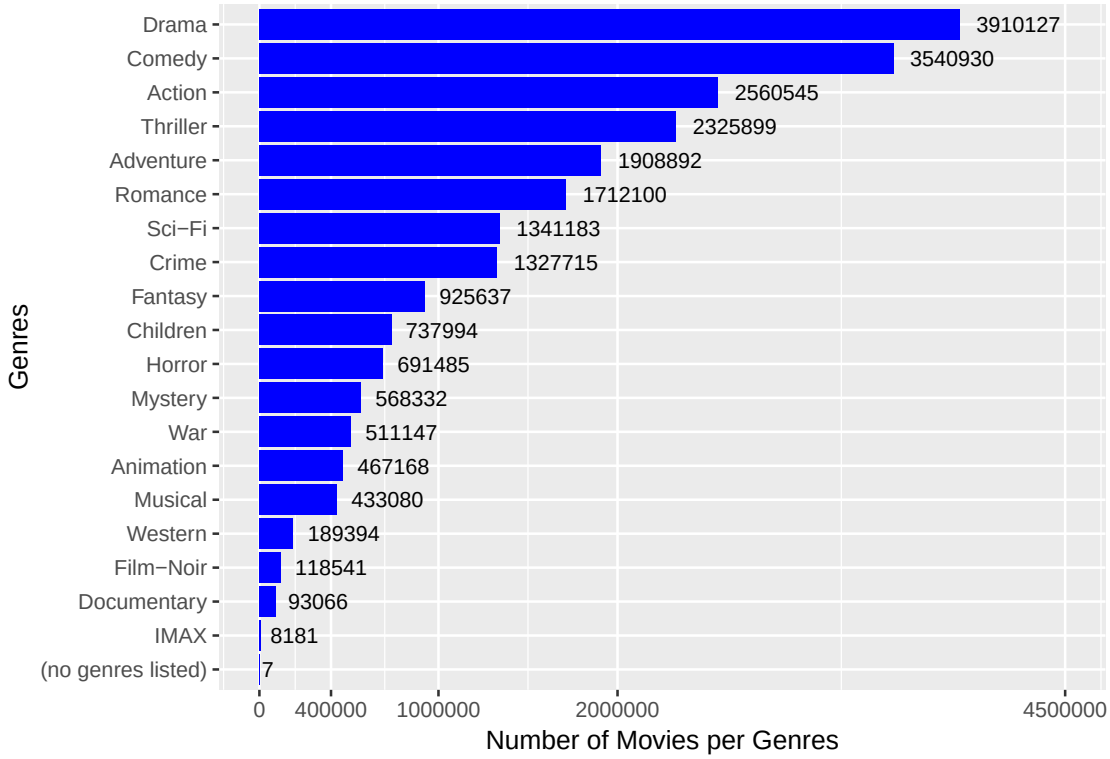
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 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
##
##
##
```

Continuing to explore edx dataset, I used descriptive statistics to analyze some variables as follows:

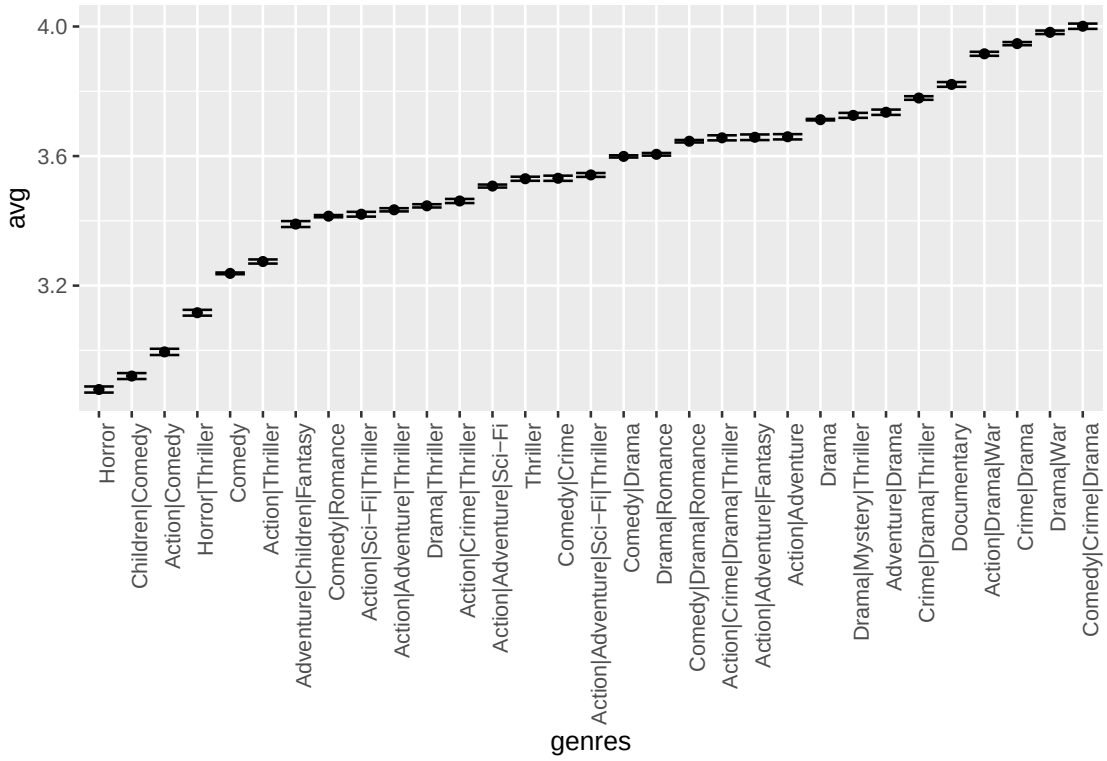
- **Genres**: for each genre, the number of ratings has been counted. It seems the Genres have effects on rating distribution. Also condidering classes of genres by the average ratings, the effect is confirmed;

- **Time**: there is no marked evidence of a time effect on average rating. It is mostly stable also considering different time frames (weeks, months, etc.). I decided to exclude "Timestamp" in my analysis;
- **Ratings**: it seems that both, movies and users, have effects on rating distribution. There are movies with many votes and movies with a lower rate of votes. There are users that usually provide a vote to the movies they see and on the other hand there are users that provide very few votes;
- **Movies mostly rated**: accordingly to the previous point I just obtained the top forty voted movies.

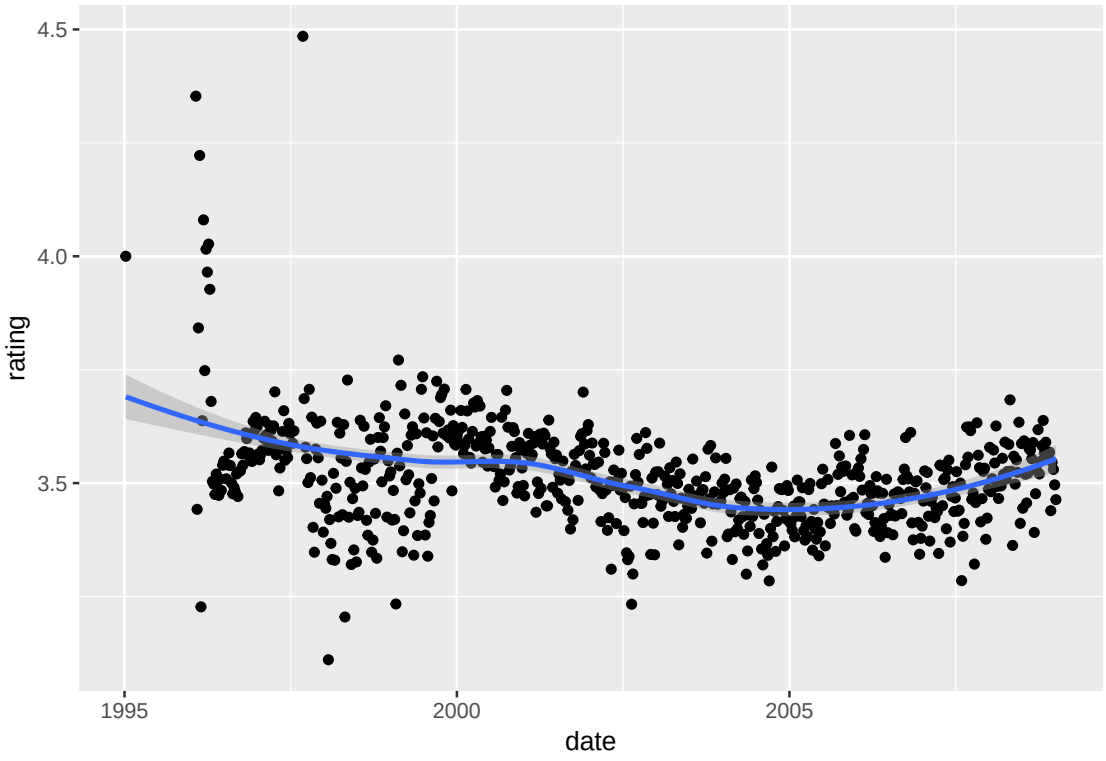
```
# Explore Genres

Genres <- edx %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>%
  summarize(count = n()) %>% arrange(desc(count))
Genres %>% ggplot(aes(x=reorder(genres, count), y=count)) +
  geom_bar(stat="identity", fill="blue") +
  coord_flip(y=c(0, 4500000)) +
  labs(x="Genres", y="Number of Movies per Genres") +
  scale_y_continuous(breaks = c(0,400000,1000000,2000000,4500000)) +
  geom_text(aes(label=count), hjust=-0.2, size=3)
```



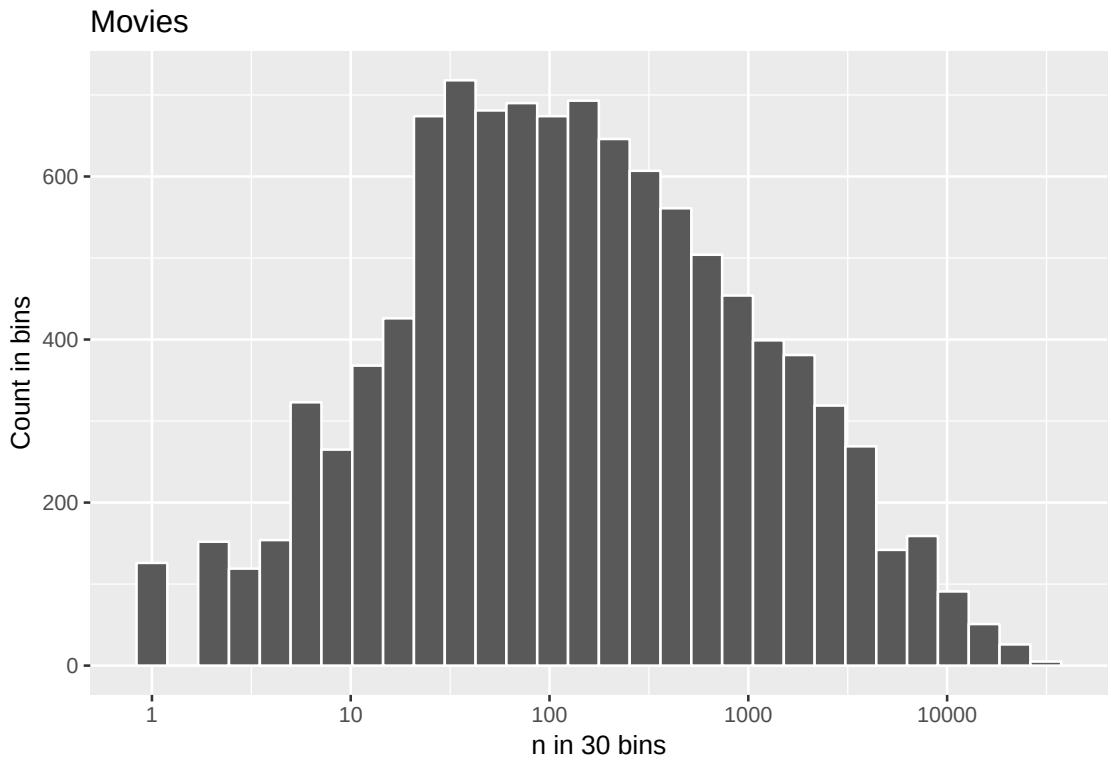

```
# Explore Genres by the average ratings.

edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 50000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



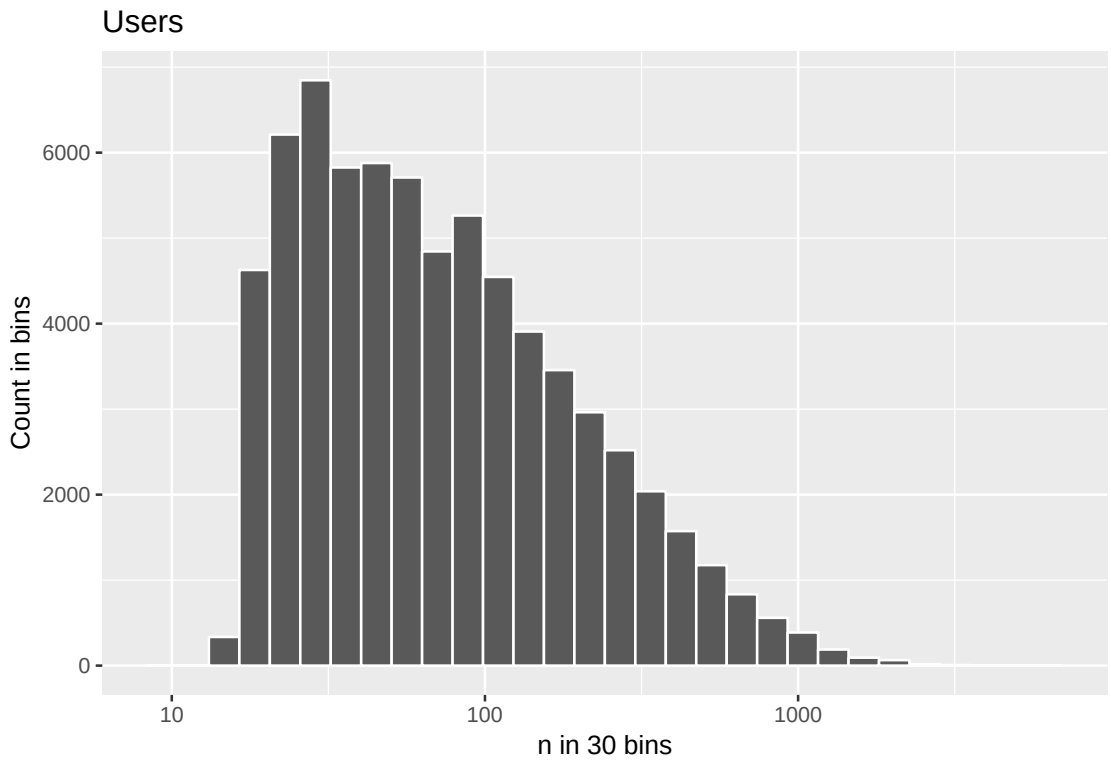

```
# Explore Time

Times <- mutate(edx, date = as_datetime(timestamp))
Times %>% mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```



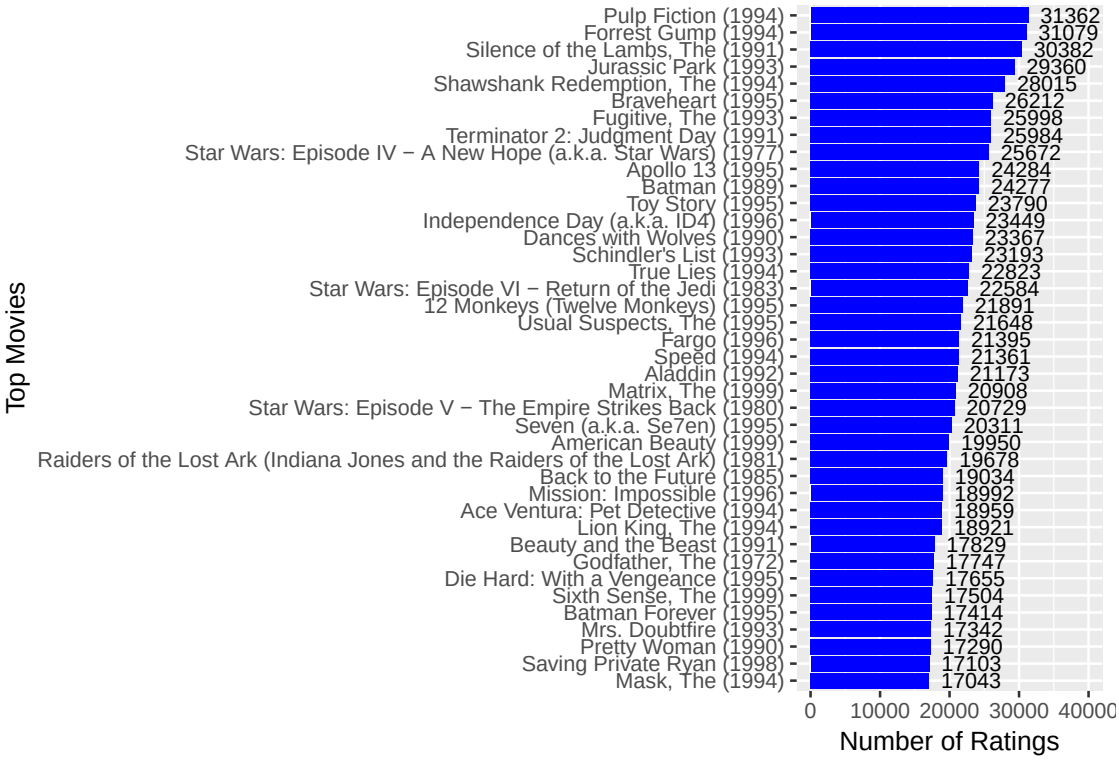

```
# Explore number of ratings by movieId and UserId.

edx %>%  count(movieId) %>% ggplot(aes(n)) +
  geom_histogram(bins=30, color="white") +
  scale_x_log10() +
  ggtitle("Movies") +
  labs(x="n in 30 bins", y="Count in bins")
```




```
edx %>% count(userId) %>% ggplot(aes(n)) +
geom_histogram(bins=30, color="white") +
scale_x_log10() +
ggtitle("Users") +
labs(x="n in 30 bins", y="Count in bins")
```




```
  # Explore the Movies with highest number of ratings.

Movie_rating_count <- edx %>% group_by(title) %>% summarize(count=n()) %>%
  top_n(40, count) %>% arrange(desc(count))
Movie_rating_count %>% ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat="identity", fill="blue") +
  coord_flip(y=c(0, 40000)) +
  labs(x="Top Movies", y="Number of Ratings") +
  scale_y_continuous(breaks = c(0,10000,20000,30000,40000)) +
  geom_text(aes(label=count), hjust=-0.2, size=3)
```



2.3 Pre-processing: defining RMSE function

Before the analysis, I define the "Root Mean Square Error" function that will be used to evaluate how good the model fits with the validation data. The formula is:

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

where

- $\hat{y}_{u,i}$ is the predicted values
- $y_{u,i}$ is the observed value
- N is the number of elements not null in the matrix or data.frame

```
# Function Definition for RMSE calculation

RMSE <- function(true_rating, predicted_rating){
  sqrt(mean((true_rating - predicted_rating)^2))}
```

2.4 Pre-processing: Building a sparse matrix for data analysis

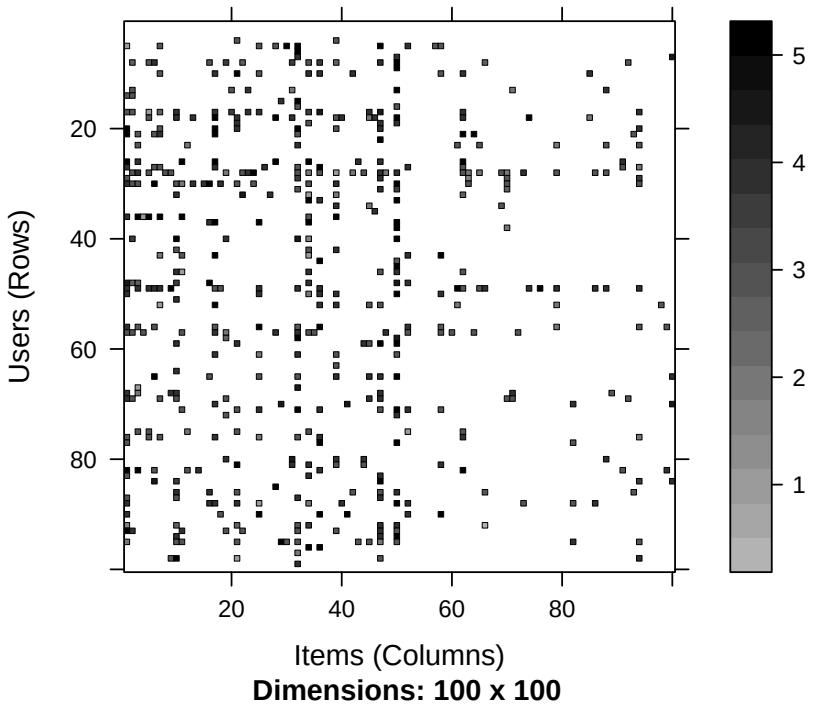
I am going to implement the sparseMatrix function from Matrix library to build the matrix. Before doing that I:

- copied the original edX data frame to leave it available for further uses;

- transformed userId and movieId as factor data type because factors are used to group the units under observation (in this case users and movies) and this is required by further analysis.

After that, starting from the sparse matrix, I created a "realRatingMatrix" object from recommanderlab package. Just to see how the sparse matrix looks like, I generated an image of it for only 100 x 100 cells.

```
# Obtain a sparse Matrix in class realRatingMatrix

edx2 <- edx
edx2$userId <- as.factor(edx2$userId)
edx2$movieId <- as.factor(edx2$movieId)

edx2$userId <- as.numeric(edx2$userId)
edx2$movieId <- as.numeric(edx2$movieId)

sparse_ratings <- sparseMatrix(i = edx2$userId, j = edx2$movieId, x = edx2$rating,
                               dims = c(length(unique(edx2$userId)),
                                        length(unique(edx2$movieId))))

ratingMat <- new("realRatingMatrix", data = sparse_ratings)
image(ratingMat[1:100, 1:100])
```



2.5 Facing with dimensionality issues

Sparsity of ratings and the high number of 'NULL' values highlights a very challenging problem in terms of resources required to calculate a single algorithm or a bunch of them as an ensemble. There are two main techniques to reduce the dimension of the sparse matrix: "PCA" and "SVD". Looking at the svd method I

used the Irlba package that implements a memory-efficient way to compute a partial SVD. According to what was suggested by Rafael Irizarry in the lesson on Matrix factorization, the SVD can help us to decompose a Matrix M[R,C] with R<C in as follows:

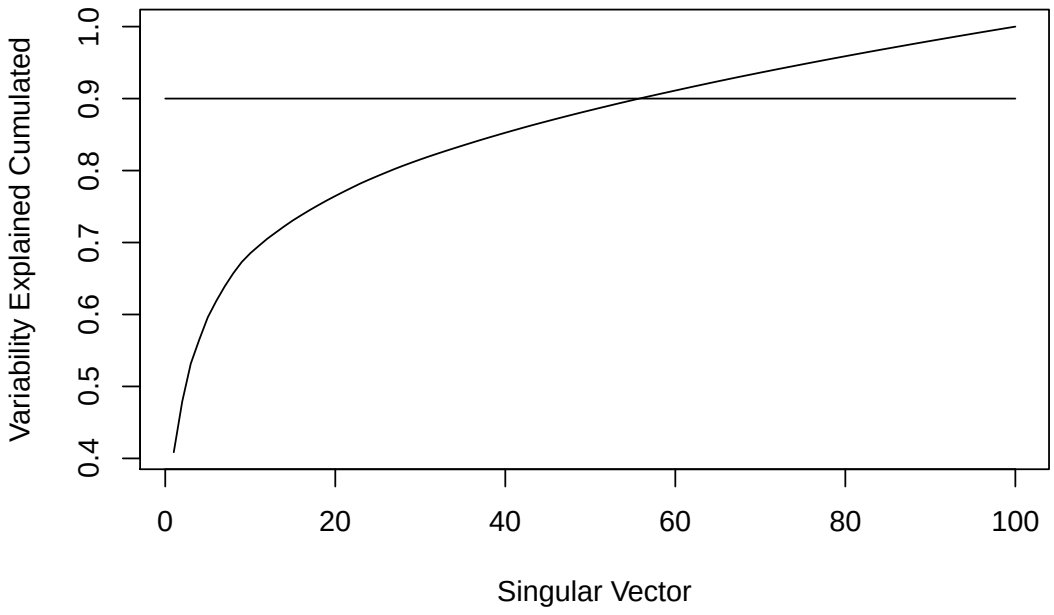
$$M = U \times D \times V'$$

where

- U : orthogonal matrix of dimensions $R \times m$
- D : diagonal matrix containing the singular values of the original matrix, $m \times m$
- V' : orthogonal matrix transposed of dimensions $m \times C$

Irlba algorithm shows the k parameter equal to the number of Singular vectors with 90% of variability explained. It is also shown into the plot. In this way I could say that our original matrix made of 9000055 ratings becomes of $(69878 \times 55) + (55 \times 55) + (55 \times 10677) = 4,433,550$ cells. So about the 50% less than the original one. Now the question was: is it possible to do better? So this is a key point to address. How to reduce the sparse matrix without losing important data to train the model? I also tried PCA method in this study.

```
set.seed(1)
Red_ratings <- irlba(sparse_ratings, tol=1e-4, verbose=TRUE, nv= 100, maxit=1000)
plot(cumsum(Red_ratings$d^2/sum(Red_ratings$d^2)), type="l", xlab="Singular Vector",
     ylab="Variability Explained Cumulated")
lines(x=c(0,100), y= c(.90, .90))
```



```
k = max(which(cumsum(Red_ratings$d^2/sum(Red_ratings$d^2)) <= .90))
k # Number of Singular vectors with 90% of variability explained
```

```
## [1] 55
```

```
U <- Red_ratings$u[, 1:k]
D <- Diagonal(x = Red_ratings$d[1:k])
V <- t(Red_ratings$v)[1:k,]

# U%*%D%*%V this provide us the original matrix at 90%
# U%*%V this provides the predicted ratings
```

The idea is to reduce the matrix knowing that many users give few ratings and also that many movies do not have many ratings. So I could cut the movies and the rows using these criteria. I calculated quantiles and in particular the 90th percentile that leaves on its left 90% of the elements of the distribution. I did that for both, movies and users. After that, I took into consideration just the row counts (number of movies voted for each user) and column counts (number of users that provided a vote for each movie) of the matrix that are greater than two quantiles.

```
# Different approach to reduce the original matrix (realRatingMatrix)

min_movies <- quantile(rowCounts(ratingMat),0.9)
min_movies
```

```
## 90%
## 301
```

```
min_users <- quantile(colCounts(ratingMat), 0.9)
min_users
```

```
##    90%
## 2150.2
```

```
ratings_movies <- ratingMat[rowCounts(ratingMat) > min_movies,
                            colCounts(ratingMat) > min_users]
ratings_movies
```

```
## 6978 x 1068 rating matrix of class 'realRatingMatrix' with 2313148 ratings.
```

So the new matrix (named ratings_movies) dimension is made of 2,313,148 ratings on 7,452,504 (elements or cells).

2.6. Machine Learning implementation

Based on pre-processing data and information I tried some Machine Learning methods. For each one the model was validated using RMSE function.

2.6.1 Linear regression models

The first family models are related to the linaer regression ones using movies, users and genres effects. This approach comes from lessons provided by Irizarry, R. in Recommandation Systems topic. I followed an incremental approach starting from a simple linear regression model including the same rating (the general average "mu") for all users and movies plus an error. So the true rating value for the 'u' user and the 'i' movie is:

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

As follows, the code shows how I added (one by one) to this "naive" starting model the following terms:

- b_i is the avegrage ranking form movies i
- b_u is equal to the mean of (rating - mu - b_i) the user effect
- b_g is equal to the mean of (rating - mu - b_i - b_u) the genre effect

to enrich the model. For each one, step by step, the rmse has been calculated. The time has not been included because as we saw in exploratory phase it doesn't have a marked effect on ratings.

```
# Linear regression models using just the mean, movies, users and genres effect

mu <- mean(edx$rating)
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.061202
```

```
movie_avg <- edx %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
predicted_rating <- mu + validation %>% left_join(movie_avg, by = "movieId") %>% .$b_i
movie_rmse <- RMSE(validation$rating, predicted_rating)
movie_rmse
```

```
## [1] 0.9439087
```

```
user_avg <- edx %>% left_join(movie_avg, by = "movieId") %>% group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_rating <- validation %>% left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>% .$pred
movie_user_rmse <- RMSE(validation$rating, predicted_rating)
movie_user_rmse
```

```
## [1] 0.8653488
```

```
genres_avg <- edx %>% left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
predicted_rating <- validation %>% left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>% left_join(genres_avg, by="genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>% .$pred
movie_genres_rmse <- RMSE(validation$rating, predicted_rating)
movie_genres_rmse
```

```
## [1] 0.8649469
```

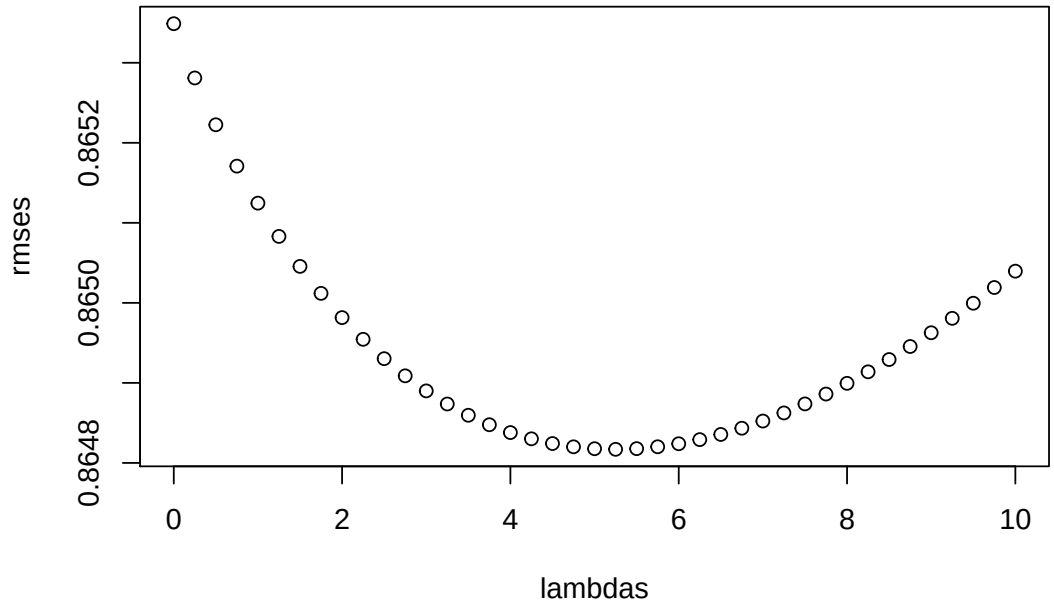
2.6.1.1 Regularization

The question is: why when I calculated just the movie effect the rmse was only about 0.94 respect to a good variation shown in the "Movie" distribution? The reguralization allows us to reduce the wrong effect due to the fact that supposed best and worst movies have been rated by very few users. So in other words larger estimates of b_i are more likely when fewer users rate the movies. It's a bias. So large errors increase the rmse. Using this approach "we can penalize large estimates that come from small sample sizes... The general idea is to add a penalty for large values of b to the sum of square equations that we minimize" (by Irizarry, R.)

$$\frac{1}{N}\sum(\hat{y}_{u,i} - y_{u,i})^2 + \lambda\sum_i b_i^2$$

the second part is the penalty component of the equation. For details, please refer to Recommandation System section of Machine Learning Course by Prof. Irizarry, R. I don't use genres in the regularization approach because I saw that including it, the rmse didn't improve so much (just to 0.8644546 for λ equal to 5). In the following code I found the λ as a tuning parameter trying numbers from 0 to 10 by 0.25. I found the λ that minimize the rmse and the minimum rmse for that value of λ.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mu <- mean(edx$rating)
  b_i <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>% left_join(b_i, by='movieId') %>% group_by(userId) %>%
    summarize(b_u = sum(rating -b_i -mu)/(n()+l))
  predicted_rating <- validation %>% left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %>% .$pred
  return(RMSE(validation$rating, predicted_rating))})

plot(lambdas, rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.25
```

```
min(rmses)
```

```
## [1] 0.864817
```

2.6.2 RecommenderLab Engines

Referring to https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf is possible to implement and test some recommender algorithms for rating data and 0-1 data in a unified framework. Three of them have been selected: singular Value Decomposition (SVD); User-Based Collaborative Filtering (UBCF); Item-Based Collaborative Filtering (IBCF). For these algorithms it was not possible to use train and validation set. So the rmse is just a good estimation of it. The method used for evaluation scheme was "Split". The "rating_movies" (a "realRatingMatrix" object) is split in train, known (to predict) and unknown (to validate) data. The lower rmse is obtained by SVD method.

```
# RECOMMENDER ENGINES

set.seed(1)
eval <- evaluationScheme(ratings_movies, method="split", train=0.9, given=-5)

model_svd <- Recommender(getData(eval,"train"), method="SVD")
pred_svd <- predict(model_svd, getData(eval,"known"), type="ratings")
```

```
rmse_svd <- calcPredictionAccuracy(pred_svd, getData(eval,"unknown"))[1]
rmse_svd
```

```
##      RMSE
## 0.8454439
```

```
model_UBCF <- Recommender(getData(eval,"train"), method="UBCF",
                          param=list(normalize="center", method="cosine", nn=50))
pred_UBCF <- predict(model_UBCF, getData(eval,"known"), type="ratings")
rmse_UBCF <- calcPredictionAccuracy(pred_UBCF, getData(eval,"unknown"))[1]
rmse_UBCF
```

```
##      RMSE
## 0.8600749
```

```
model_IBCF <- Recommender(getData(eval,"train"), method="IBCF",
                          param=list(normalize="center", method="cosine", k=350))
pred_IBCF <- predict(model_IBCF, getData(eval,"known"), type="ratings")
rmse_IBCF <- calcPredictionAccuracy(pred_IBCF, getData(eval,"unknown"))[1]
rmse_IBCF
```

```
##      RMSE
## 0.9650579
```

2.6.3 Parallel Stochastic Gradient Descendent

In the Matrix Factorization domain, I noticed this method. As follows, some useful links for further information:

- about recosystem Package 1
- about recosystem Package 2
- about SGD

The main tasks performed in the code are:

- for the original train and validation data set only three variables have been considered: “user”;“item”;“rating”. Both are converted in a Matrix format (this is required by the recosystem package);
- to avoid memory issues both the model for prediction and validation output can be written on two different flat files;
- it is possible to create the model using Reco(), tune() method to set parameters, train() and predict() to train and calculate predicted values.

Please note that the algorithm takes long time to run (not less than 30 minutes on my laptop). The rmse is 0.7829514.

```
# RECOMMENDER ENGINES - Matrix Factorization with parallel stochastic gradient descendent

edx.tiny <- edx %>% select(-c("genres","title","timestamp"))
names(edx.tiny) <- c("user","item","rating")
validation.tiny <- validation %>% select(-c("genres", "title", "timestamp"))
names(validation.tiny) <- c("user","item","rating")
edx.tiny <- as.matrix(edx.tiny)
validation.tiny <- as.matrix(validation.tiny)

write.table(edx.tiny, file = "traindata.txt", sep=" ", row.names=FALSE,
            col.names=FALSE)
write.table(validation.tiny, file="validationdata.txt", sep=" ", row.names=FALSE,
            col.names=FALSE)

set.seed(55)
train_set <- data_file("traindata.txt")
validation_set <- data_file("validationdata.txt")

r=Reco()
opts=r$tune(train_set, opt=list(dim=c(10,20,30), lrate=c(0.1,0.2), costp_l1=0,
                                costq_l1=0, nthread=1, niter=10))
r$train(train_set , opts=c(opts$min, nthread=1, niter=20))
```

```
## iter      tr_rmse          obj
##    0       0.9736  1.2061e+007
##    1       0.8709  9.8619e+006
##    2       0.8394  9.1755e+006
##    3       0.8182  8.7642e+006
##    4       0.8024  8.4796e+006
##    5       0.7903  8.2823e+006
##    6       0.7805  8.1308e+006
##    7       0.7723  8.0080e+006
##    8       0.7654  7.9096e+006
##    9       0.7596  7.8317e+006
##   10       0.7543  7.7607e+006
##   11       0.7497  7.7054e+006
##   12       0.7456  7.6561e+006
##   13       0.7418  7.6105e+006
##   14       0.7384  7.5721e+006
##   15       0.7353  7.5372e+006
##   16       0.7324  7.5061e+006
##   17       0.7297  7.4785e+006
##   18       0.7273  7.4535e+006
##   19       0.7251  7.4331e+006
```

```
pred_file=tempfile()
r$predict(validation_set, out_file(pred_file))
```

```
## prediction output generated at C:\Users\PADEP\AppData\Local\Temp\Rtmp2D2k0r\file3024518a3b45
```

```
scores_real <- read.table("validationdata.txt", header=FALSE, sep=" ")$V3
scores_pred <- scan(pred_file)
```

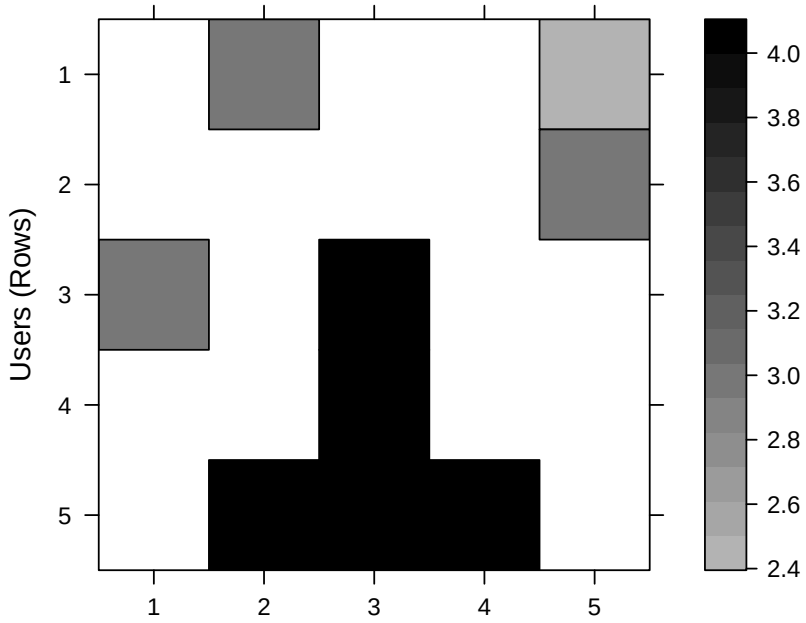
```
rmse_mf <- RMSE(scores_real, scores_pred)
rmse_mf
```

```
## [1] 0.7829514
```

2.6.4 PCA method based on genres

Just to follow again the aim of Matrix Factorization that relies on the fact that an important source of variation depends on similar patterns between groups of movies and users, now I am going to work on Genres and Principal Component Analysis (PCA). The hypothesis to verify was: if I select from the original edx data frame just "Drama" (as genre type) and after that I try to reduce the matrix using PCA, can I obtain a lower rmse? If this were the case, I could apply the same routine for other genres. At that point I could suggest the higher ratings for a specific user picking the higher rating movies from each genre. So, the following code implements these steps just for Drama (using it alone or in combination with other genres). I reuse previous pieces of code to have a sparse matrix and to reduce just movies with very few ratings. After that, with a "for" cycle I fill in the empty elements of the matrix with the column average to permit PCA computation. I obtain the variation cumulated explained by all items of the matrix. I decided to cut the variation at 60% with 76 items. Then I calculated the correlation between the eigenvectors and the original items. The correlation matrix has been exported in a .csv file. Then I found the most correlated and reported them in PCs. Just to give the process for the first one (movieId = 619) I found that max correlation between all movies and the first principal component is equal to -0.4535. It correspond to movieId 619. So using these 76 principals components to filter the original edx data set, I re-ran some methods explored in this study.

```
# Reduction based on Genres and PCA

Genres <- edx %>% filter(str_detect(genres, "Drama"))

edx.copy <- Genres
edx.copy$userId <- as.factor(edx.copy$userId)
edx.copy$movieId <- as.factor(edx.copy$movieId)

edx.copy$userId <- as.numeric(edx.copy$userId)
edx.copy$movieId <- as.numeric(edx.copy$movieId)

sparse_ratings <- sparseMatrix(
  i = edx.copy$userId,
  j = edx.copy$movieId,
  x = edx.copy$rating,
  dims = c(length(unique(edx.copy$userId)),
  length(unique(edx.copy$movieId))),
       dimnames = list(paste("u",1:length(unique(edx.copy$userId)), sep=""),
       paste("m",1:length(unique(edx.copy$movieId)), sep="")))

ratingMat <- new("realRatingMatrix", data = sparse_ratings)

min_n_movies <- quantile(rowCounts(ratingMat),0.95)
min_n_users <- quantile(colCounts(ratingMat), 0.95)
ratings_movies <- ratingMat[rowCounts(ratingMat) > min_n_movies,
                            colCounts(ratingMat) > min_n_users]

# Just to take a look to a little portion of sparse matrix
image(ratings_movies[1:5,1:5])
```



Dimensions: 5 x 5

```
getRatingMatrix(ratings_movies[1:5,1:5])
```

```
## 5 x 5 sparse Matrix of class "dgCMatrix"
##     m2 m4 m5 m8  m9
## u8   .  3  .  . 2.5
## u28  .  .  .  . 3.0
## u57  3  .  4  . .
## u70  .  .  4  . .
## u88  .  4  4  4 .
```

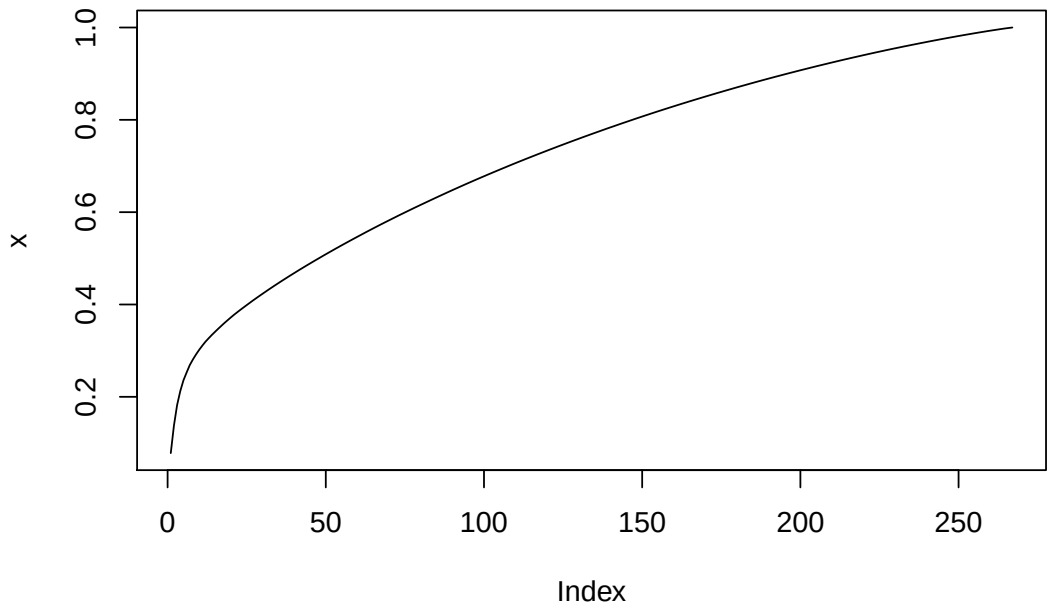
```
# Adding col_mean values instead of NAs
Genres_Matrix <- as(ratings_movies, "matrix")
Genres_Matrix[is.na(Genres_Matrix)] <- 0
mean_col <- colMeans(Genres_Matrix)

for (i in 1:nrow(Genres_Matrix)) {
  for (j in 1:ncol(Genres_Matrix))
    if (Genres_Matrix[i,j]==0) Genres_Matrix[i,j]=mean_col[j]
}

cp <- prcomp(Genres_Matrix)

dim(cp$rotation)
```

```
## [1] 267 267
```

```r
dim(cp$x)
```

```
## [1] 3471  267
```

```r
#summary(cp) - explained variability set as 60% with 76 principal components
x <- summary(cp)$importance[3,]
plot(x, type="l")
```




```r
var_explained <- cumsum(cp$sdev^2/sum(cp$sdev^2))
var_explained
```

```
##   [1] 0.07807265 0.13740727 0.18200163 0.21222616 0.23560489 0.25253737
##   [7] 0.26868584 0.28089558 0.29199383 0.30204096 0.31112488 0.31970705
##  [13] 0.32703584 0.33413012 0.34085954 0.34745064 0.35391990 0.36019199
##  [19] 0.36618887 0.37203455 0.37775210 0.38317842 0.38847735 0.39361849
##  [25] 0.39872764 0.40378426 0.40877567 0.41362570 0.41844985 0.42320673
##  [31] 0.42791868 0.43259941 0.43720081 0.44174479 0.44623292 0.45069067
##  [37] 0.45507751 0.45943568 0.46375452 0.46803880 0.47226479 0.47648577
##  [43] 0.48064004 0.48473797 0.48881909 0.49288339 0.49690287 0.50090472
##  [49] 0.50486716 0.50880766 0.51273843 0.51665206 0.52050629 0.52434158
##  [55] 0.52813396 0.53191831 0.53567113 0.53937998 0.54308056 0.54676576
##  [61] 0.55044070 0.55409589 0.55773128 0.56134155 0.56490672 0.56844808
##  [67] 0.57196681 0.57547079 0.57894787 0.58240505 0.58584758 0.58926892
##  [73] 0.59268172 0.59604459 0.59939681 0.60273532 0.60605355 0.60935758
##  [79] 0.61262041 0.61585921 0.61909493 0.62232095 0.62554306 0.62875234
##  [85] 0.63195019 0.63512417 0.63828783 0.64142746 0.64455961 0.64767780
```

```
##  [91] 0.65077920 0.65385288 0.65690546 0.65994797 0.66296395 0.66595684
##  [97] 0.66894468 0.67191534 0.67487292 0.67781060 0.68072949 0.68362633
## [103] 0.68651616 0.68937625 0.69223395 0.69508417 0.69792605 0.70075318
## [109] 0.70356813 0.70636683 0.70914426 0.71189098 0.71462885 0.71736319
## [115] 0.72007998 0.72277512 0.72546327 0.72813547 0.73078745 0.73343276
## [121] 0.73606431 0.73868534 0.74129720 0.74389939 0.74649157 0.74905453
## [127] 0.75160856 0.75414896 0.75668687 0.75921572 0.76172612 0.76422666
## [133] 0.76671076 0.76917874 0.77163710 0.77409178 0.77653935 0.77897258
## [139] 0.78137513 0.78377663 0.78616405 0.78854678 0.79091360 0.79327054
## [145] 0.79562077 0.79795980 0.80028304 0.80259782 0.80489232 0.80718097
## [151] 0.80946685 0.81173856 0.81399514 0.81623226 0.81845025 0.82066702
## [157] 0.82286930 0.82505867 0.82723657 0.82940572 0.83156064 0.83370492
## [163] 0.83584298 0.83797444 0.84009183 0.84219789 0.84427896 0.84635664
## [169] 0.84842385 0.85048371 0.85252733 0.85455677 0.85658408 0.85859944
## [175] 0.86059922 0.86259799 0.86458084 0.86655896 0.86852809 0.87049179
## [181] 0.87244139 0.87436079 0.87627649 0.87818719 0.88009293 0.88198454
## [187] 0.88386689 0.88573942 0.88759570 0.88944768 0.89128203 0.89310897
## [193] 0.89492030 0.89672340 0.89852049 0.90030823 0.90209113 0.90385507
## [199] 0.90561206 0.90735801 0.90910047 0.91082147 0.91253412 0.91423798
## [205] 0.91592702 0.91760880 0.91928664 0.92095764 0.92261028 0.92425654
## [211] 0.92589947 0.92752596 0.92914045 0.93074625 0.93234338 0.93393152
## [217] 0.93550159 0.93706123 0.93861681 0.94016617 0.94170158 0.94322413
## [223] 0.94474040 0.94623728 0.94773177 0.94922309 0.95070860 0.95217366
## [229] 0.95362983 0.95505279 0.95647340 0.95788676 0.95929303 0.96068057
## [235] 0.96205957 0.96342987 0.96479421 0.96614815 0.96749150 0.96882166
## [241] 0.97014686 0.97146834 0.97278785 0.97407684 0.97536279 0.97663769
## [247] 0.97789948 0.97914013 0.98037324 0.98159441 0.98280931 0.98400831
## [253] 0.98519122 0.98636676 0.98753511 0.98868407 0.98983027 0.99094364
## [259] 0.99205027 0.99312939 0.99419951 0.99524231 0.99628254 0.99728621
## [265] 0.99823370 0.99915045 1.00000000
```

```
EV <- cp$x[,1:76]
r<-cor(Genres_Matrix,EV)

#write.table(r, file = "PCA_Corr.csv",row.names=TRUE, na="",col.names=TRUE, sep=",")

PCs <- c("619","3324","1822","425","2736","513","1498","664","270","623","1891","121",
         "1101","616","970","582","1382","1379","613","647","164","967","651","1141",
         "669","265","596","630","15","407","941","644","607","638","837","1274","2746",
         "143","274","244","519","466","181","242","3880","615","262","636","559","543",
         "149","184","286","968","276","4785","1476","36","1095","20","172","916","1328",
         "2483","1570","532","57","1968","10","631","524","145","1791","946","137","29")

edx.copy <- edx %>% filter(movieId %in% PCs)
edx.copy$userId <- as.factor(edx.copy$userId)
edx.copy$movieId <- as.factor(edx.copy$movieId)

edx.copy$userId <- as.numeric(edx.copy$userId)
edx.copy$movieId <- as.numeric(edx.copy$movieId)

sparse_ratings <- sparseMatrix(i = edx.copy$userId,
                               j = edx.copy$movieId,
                               x = edx.copy$rating,
                               dims = c(length(unique(edx.copy$userId)),
```

```
                                        length(unique(edx.copy$movieId))))

ratingMat <- new("realRatingMatrix", data = sparse_ratings)

min_movies <- quantile(rowCounts(ratingMat),0.9)
#min_users <- quantile(colCounts(ratingMat), 0.9)
ratings_movies <- ratingMat[rowCounts(ratingMat) > min_movies]

set.seed(1)
eval <- evaluationScheme(ratings_movies, method="split", train=0.8, given=-1)

model_svd <- Recommender(getData(eval,"train"), method="SVD")
pred_svd <- predict(model_svd, getData(eval,"known"), type="ratings")
rmse_pca_svd <- calcPredictionAccuracy(pred_svd, getData(eval,"unknown"))[1]
rmse_pca_svd
```

```
##      RMSE
## 0.9630308
```

```
model_svdf <- Recommender(getData(eval,"train"), method="SVDF")
pred_svdf <- predict(model_svdf, getData(eval,"known"), type="ratings")
rmse_pca_svdf <- calcPredictionAccuracy(pred_svdf, getData(eval,"unknown"))[1]
rmse_pca_svdf
```

```
##      RMSE
## 0.9444094
```

```
model_UBCF <- Recommender(getData(eval,"train"), method="UBCF",
                          param=list(normalize="center", method="cosine", nn=50))
pred_UBCF <- predict(model_UBCF, getData(eval,"known"), type="ratings")
rmse_pca_UBCF <- calcPredictionAccuracy(pred_UBCF, getData(eval,"unknown"))[1]
rmse_pca_UBCF
```

```
##      RMSE
## 0.9609646
```

```
model_IBCF <- Recommender(getData(eval,"train"), method="IBCF",
                          param=list(normalize="center", method="cosine", k=350))
pred_IBCF <- predict(model_IBCF, getData(eval,"known"), type="ratings")
rmse_pca_IBCF <- calcPredictionAccuracy(pred_IBCF, getData(eval,"unknown"))[1]
rmse_pca_IBCF
```

```
##     RMSE
## 1.049547
```

3. Results

In the following table, I recap all rmse values calculated with the analyses presented in this study.

Methods	RMSE
Naive	1.0612018
Movie	0.9439087
Movie & User	0.8653488
Movie & Users & Genres	0.8649469
Movie & USers with Regularization	0.8648170
SVD	0.8454439
UBCF	0.8600749
IBCF	0.9650579
Parallel Stochastic Gradient Descendent	0.7829514
PCA & SVD	0.9630308
PCA & SVDF	0.9444094
PCA & UBCF	0.9609646
PCA & IBCF	1.0495471

The use of PCA based on genres, didn't provide the expected results. The lower RMSE is related to the **"Parallel Stochastic Gradient Descendent"** with a value equal to **0.78295**.

4. Conclusions

Using RMSE as a criteria for evaluating the algorithms, it is possible to see that the two best methods are matrix factorization with stochastic gradient descent and the regression model with regularized users and films. SVD is also good (but just as an estimate of RMSE). It looks quite good using the recommenderLab package. It may be worth using "Ensemble Methods" to get more powerful results. With the many excellent algoritms available in the web, in the future, with the use of more powerful computers (parallel computing that uses different CPUs, GPUs and much more RAM or quantum computers), it will be possible to improve the outputs and obtain better performance.

Special thanks to Professor Rafael Irizarry and to the edX staff.

Sincerely,

Paolo De Piante