

**DATA SCIENCE PROJECT**  
  
REPORT

*Elie Bensoussan*

*Malo Estéoule*

*Giulia Rampa*

*Léa Repaux-Coustet*



SCRIPT R - Recommender

STEP 1 : Set up

We set up the frame of the project and start to input the data from the CSV, so we specify the directory we want to go and start sourcing in the files.

We started the cleaning of the unnecessary data presented in the files like time stamping and start to display the relevant data, i.e. user ID, movie ID and ratings.

STEP 2 : Display of the data

Then, the display of the data gave us a general idea of the quantity and the features of the data in order to process it.

STEP 3 : Code the variables

in order to start the algorithm, each observation of each variable is coded with figures in order to be used in the loop

STEP 4 : TRAIN/TEST SPLIT

In this step, there is three different stage in the process :

Preparing a training set

Applying the same preparation to a testing set

Controlling that train and test sets have the same shape

In order to simulate a train and test set we are going to split randomly this data set into 70% train and 30% test

STEP 5 : better understand the data

We start to display a matrix of the ratings so we can assess popularity of the different movies and some histograms to get a clear view of the data. In order to have a closer shot of the rating matrix, we converted it as a list and then a data frame, then we proceed to a normalisation of the matrix in order to remove any bias.

STEP 6 : create the recommender algorithm

The matrix is then transformed into a 0-1 matrix (binarized) in order to create the recommender algorithm.

The method use the method based on three different algorithms :

if the film was popular (POPULAR)

a random way (RANDOM)

user-based

STEP 7 : Evaluation of the algorithm

Then there is the evaluation and the computation of the predicted ratings in order to access the error between the prediction and the unknown part of the test data. use of the function evalutionScheme for the evaluation of the predicted ratings. The aim of the algorithm is to be able to predict the withheld items. then we can compare the prediction of the algorithm and the unknown part of the test data to see how well it worked.

STEP 8 : Outcome

once the loop is finished, a curve is displayed for recommender method POPULAR.

Finally, the code is reversed and the name of the movies are converted back to display the name.

SCRIPT R - Recommender

getwd()

setwd("/Users/Lea/Desktop/Projet Big Data/project\_bigdata")

library(readr)

movies\_group13 <- read\_csv("movies\_group13.csv")

ratings\_group13 <- read\_csv("ratings\_group13.csv")

# Get rid of column "timestamp" and "X1"

ratings\_group13 = ratings\_group13[ , -which(names(ratings\_group13) %in% c("timestamp"))]

data <- ratings\_group13[ , - c(1)]

colnames(data) = c("user\_id", "movie\_id", "rating")

#See data

str(data)

data = as.data.frame(data)

str(data)

summary(data)

hist(data$rating)

# DATA SPARSITY

Number\_Ratings = nrow(ratings\_group13)

Number\_Movies = length(unique(ratings\_group13$movieId))

Number\_Users = length(unique(ratings\_group13$userId))

# SPLITTING DATA RANDOMLY TRAIN/TEST

library(caTools)

spl = sample.split(data$rating, 0.7)

train = subset(data, spl == TRUE)

test = subset(data, spl == FALSE)

# RECOMMENDER

library("recommenderlab")

r <- as(data, "realRatingMatrix")

r

r\_train <- as(train, "realRatingMatrix")

r\_train

getRatingMatrix(r)

getRatingMatrix(r\_train)

# Understand the data better

as(r[1,], "list")

rowMeans(r[1,])

rowCounts(r[1,])

hist(getRatings(r), breaks=100)

hist(rowCounts(r), breaks=50)

hist(colMeans(r), breaks=20)

# Convert the rating matrix into a list of users with their ratings for closer inspection

as(r, "list")

# The rating matrix can converted into a data.frame with user/item/rating tuples.

head(as(r, "data.frame"))

head(as(r\_train, "data.frame"))

# Normalization of the rating matrix to remove biases

r\_m <- normalize(r)

r\_m

getRatingMatrix(r\_m)

# The rating matrix can be transformed into a 0-1 matrix

r\_b <- binarize(r, minRating=4)

r\_b

# Information about interesting recommandation methods

recommenderRegistry$get\_entries(dataType = "realRatingMatrix")

###Popular

rr\_pop <- Recommender(r\_train[1:10000], method = "POPULAR")

names(getModel(rr\_pop))

getModel(rr\_pop)

recom\_pop <- predict(rr\_pop, r\_train[10233:10234], n=5)

recom\_pop

recom\_pop@ratings

as(recom\_pop, "list")

###Random

rr\_random <- Recommender(r\_train[1:10000], method = "random")

names(getModel(rr\_random))

getModel(rr\_random)

recom\_random <- predict(rr\_random, r\_train[10233:10234], n=5)

recom\_random

recom\_random@ratings

as(recom\_random, "list")

###UBCF

rr\_ubcf <- Recommender(r\_train[1:10000], method = "UBCF")

names(getModel(rr\_ubcf))

getModel(rr\_ubcf)

recom\_ubcf <- predict(rr\_ubcf, r\_train[11243:11244], n=5)

recom\_ubcf

recom\_ubcf@ratings

as(recom\_ubcf, "list")

# Evaluation of predicted ratings

r\_train\_bis <- r\_train[which(rowCounts(r\_train)>15),]

e <- evaluationScheme(r\_train\_bis[1:1000], method="split", train=0.7, given=15, goodRating=3)

e

r1 <- Recommender(getData(e, "train"), "Popular")

r1

r2 <- Recommender(getData(e, "train"), "Random")

r2

r3 <- Recommender(getData(e, "train"), "UBCF")

r3

# Compute predicted ratings for the known part of the test data

p1 <- predict(r1, getData(e, "known"), type="ratings")

p1

p2 <- predict(r2, getData(e, "known"), type="ratings")

p2

p3 <- predict(r3, getData(e, "known"), type="ratings")

p3

# Error between the prediction and the unknown part of the test data

error <- rbind(

Popular = calcPredictionAccuracy(p1, getData(e, "unknown")),

Random = calcPredictionAccuracy(p2, getData(e, "unknown")),

UBCF = calcPredictionAccuracy(p3, getData(e, "unknown"))

)

error

# Evaluation of a top-N recommender algorithm

?evaluationScheme

scheme <- evaluationScheme(r\_train\_bis[1:1000], method="cross", k=4, given=15, goodRating=3)

scheme

#Use the created evaluation scheme to evaluate the recommender methods top-3, top-5

results\_pop <- evaluate(scheme, method="POPULAR", type = "topNList",

n=c(3,5))

results\_pop

results\_random <- evaluate(scheme, method="random", type = "topNList",

n=c(3,5))

results\_random

results\_UBCF <- evaluate(scheme, method="UBCF", type = "topNList",

n=c(3,5))

results\_UBCF

# confusion matrices for the 1st run

getConfusionMatrix(results\_pop)[[1]]

getConfusionMatrix(results\_random)[[1]]

getConfusionMatrix(results\_UBCF)[[1]]

# Average confusion matrices for all the 4 runs

avg(results\_pop)

avg(results\_random)

avg(results\_UBCF)

# ROC curve for recommender method POPULAR

plot(results\_pop, annotate=TRUE)

plot(results\_random, annotate=TRUE)

plot(results\_UBCF, annotate=TRUE)

# Precision-recall plot for method POPULAR.

plot(results\_pop, "prec/rec", annotate=TRUE)

plot(results\_random, "prec/rec", annotate=TRUE)

plot(results\_UBCF, "prec/rec", annotate=TRUE)

#Choosing methog Popular as it is the most efficient

rr\_pop <- Recommender(r\_train[1:10000], method = "POPULAR")

recom\_pop <- predict(rr\_pop, r\_train[12458:12459], n=5)

recom\_pop

recom\_pop@ratings

as(recom\_pop, "list")

#Converting Movie IDs to Titles

nb\_users\_to\_recomand = 2

nb\_movies\_to\_recomand = 5

for (i in 1: nb\_users\_to\_recomand){

print(paste("Je recommande les films suivant pour l'utilisateur",i))

films<-vector()

for(j in 1: nb\_movies\_to\_recomand){

films[j]<-as.character(subset(movies\_group13, movies\_group13$movieId==as.numeric(as(recom\_pop, "list")[[i]])[j])$title)

}

print(films)

}

SCRIPT R - Sentiment analysis

getwd()

setwd("/Users/Lea/Desktop/Projet Big Data/sentiment\_analysis")

#Require

require("base64enc")

require("twitteR")

require('ggplot2')

require('gridExtra')

require('ROAuth')

require('bitops')

require('RCurl')

library('ggplot2')

library('gridExtra')

library('tm')

library('stringr')

library('Matrix')

library('igraph')

library('plyr')

library('RColorBrewer')

library('wordcloud')

require('stopwords')

library("SnowballC")

library('rvest')

#connection to twitter

api\_key <- "L4PEeMPUvQJey7dYwiDPrzhz4";

api\_secret <- "tl8hWT2BV3SsaCioXGuf1g1OJEBSMsbGVVFokqve2yS27TkDbP";

token <- '2832222369-w1l5PYrVDktDIhwVhpzUaEE7VCUymXAo0hE8qxd';

token\_secret <-'AUx3UlMFeTde8n1Jkk7bFmujGztsH9TGJty45uPhvtcQ0';

setup\_twitter\_oauth(api\_key, api\_secret, token, token\_secret)

#TwitterMining

source("socialMiningFunctions.R")

search.string <- "netflix recommendation"

retryOnRateLimit=1

Twitter.netflix <-searchTwitter(search.string, n=5000,lang="en", retryOnRateLimit = retryOnRateLimit);

tweets.df <-twListToDF(Twitter.netflix)

docs <- Corpus(VectorSource(tweets.df$text))

inspect(docs)

toSpace <- content\_transformer(function (x, pattern ) gsub(pattern, " ", x))

docs <- tm\_map(docs, toSpace, "/")

docs <- tm\_map(docs, toSpace, "@")

docs <- tm\_map(docs, toSpace, "\\|")

docs <- tm\_map(docs, removeWords, "said")

docs <- tm\_map(docs, content\_transformer(tolower))

docs <- tm\_map(docs, removeNumbers)

docs <- tm\_map(docs, removeWords, stopwords("english"))

docs <- tm\_map(docs, removePunctuation)

docs <- tm\_map(docs, stripWhitespace)

###########Sentiment Score#########

positive.words <- scan("positive-words.txt",what='charachter',comment.char=";")

negative.words <- scan("negative-words.txt",what='charachter',comment.char=";")

plus.minus.score <- score.sentiment(tweets.df$text, positive.words, negative.words)

ggplot(plus.minus.score, aes(x=score))+

geom\_histogram(binwidth=1, colour="red", fill="black", alpha=.5) +

scale\_x\_continuous(name = "Score", breaks=(-5:5), labels=(-5:5), limits=c(-5,5)) +

scale\_y\_continuous(name = "No. of Tweets") +

ggtitle('Netflix Recommendation')

#Wordcloud

dtm <- TermDocumentMatrix(docs)

m <- as.matrix(dtm)

v <- sort(rowSums(m),decreasing=TRUE)

d <- data.frame(word = names(v),freq=v)

head(d, 10)

wordcloud(words = d$word, freq = d$freq, min.freq = 2, max.words=60, rot.per=0.3, scale=c(2,1), random.order=FALSE, colors=brewer.pal(8, "RdGy"))

######################

search.string.net <- "netflix"

retryOnRateLimit=1

Twitter.netflix.net <-searchTwitter(search.string.net, n=8000,lang="en", retryOnRateLimit = retryOnRateLimit);

tweets.df.net <-twListToDF(Twitter.netflix.net)

docs.net <- Corpus(VectorSource(tweets.df.net$text))

inspect(docs.net)

toSpace <- content\_transformer(function (x, pattern ) gsub(pattern, " ", x))

docs.net <- tm\_map(docs, toSpace, "/")

docs.net <- tm\_map(docs, toSpace, "@")

docs.net <- tm\_map(docs, toSpace, "\\|")

docs.net <- tm\_map(docs, removeWords, c("said", "for", "the", "it", "&amp;", "it's", "https"))

docs.net <- tm\_map(docs, content\_transformer(tolower))

docs.net <- tm\_map(docs, removeNumbers)

docs.net <- tm\_map(docs, removeWords, stopwords("english"))

docs.net <- tm\_map(docs, removePunctuation)

docs.net <- tm\_map(docs, stripWhitespace)

###########Sentiment Score#########

plus.minus.score.net <- score.sentiment(tweets.df.net$text, positive.words, negative.words)

ggplot(plus.minus.score.net, aes(x=score))+

geom\_histogram(binwidth=1, colour="red", fill="black", alpha=.5) +

scale\_x\_continuous(name = "Score", breaks=(-5:5), labels=(-5:5), limits=c(-5,5)) +

scale\_y\_continuous(name = "No. of Tweets") +

ggtitle('Netflix')

#Wordcloud

dtm.net <- TermDocumentMatrix(docs.net)

m.net <- as.matrix(dtm.net)

v.net <- sort(rowSums(m.net),decreasing=TRUE)

d.net <- data.frame(word = names(v.net),freq=v.net)

head(d.net, 10)

wordcloud(words = d.net$word, freq = d$freq, min.freq = 2, max.words=60, rot.per=0.3, scale=c(3,1), random.order=FALSE, colors=brewer.pal(8, "RdGy"))

Flow IBM Watson - Chatbot

