```
## IST 707  Project Fall2018 ## Viral Events:Yellow Vests

#### Twitter ####

## set the working directory

setwd("/Users/HAG/Desktop/Fall 18/IST 707 Data
Analytics/IST 707")
getwd()
##load the libraries
library(arules)
library(rtweet)
library(twitteR)
library(ROAuth)
library(jsonlite)
#library(streamR)
library(rjson)
library(tokenizers)
library(tidyverse)
library(plyr)
library(dplyr)
library(ggplot2)
library(syuzhet)
library(stringr)
library(arulesViz) ## load last
library(qdapRegex)
##library(neuralnet)
library(tm)
library(NLP)## has conflict with arules
library(RCurl)
library(bitops)
library(digest)
library(SnowballC)
library(ggplot2)
##library(graph) #not available
##library(Rgraphviz)## not available
library(wordcloud)
library(RColorBrewer)
library(fpc)
library(topicmodels)
```

```
??twitteR
??SnowballC
## need a Twitter developer account,
## go to www.apps.twitter.com to get one.
## get an API: application programming interface
## keep your API in an Excel csv file, it is a private key
to get twitter files
##?? setup_twitter_oauth
twitter_oauth <- read.csv("twitter_oauth.csv")
##View(twitter_oauth)

## set up a twitter oauth
setup_twitter_oauth(twitter_oauth$consumer_key,
twitter_oauth$consumer_secret, twitter_oauth$access_token,
twitter_oauth$access_secret)



## search twitter
## what words people are they saying together?
W3:01:03:DR.Gates said!
## It seems pretty interesting and we can learn a lot
## a lot about what people are saying and how those
## words are in conjuction with each other!
## Let's try "#YellowVests, #paris #giletsjaunesparis
#GiletsJaunes #yellowvest" first
## "#YellowVests"
## Let's find out: what is viral

## load tweets into R
#( Search_YW<- twitteR::searchTwitter("#YellowVests",
n=1000, since = "2018-12-1"))
tweets <- Search_YW
(Search_YW)
# convert tweets to a data frame
# tweets.df <- do.call("rbind", lapply(tweets,
as.data.frame))
tweets.df <- twListToDF(tweets)
names(tweets.df)
dim(tweets.df)
str(tweets.df)
## Load NLP
```

```
## built a corpus, and define the source to be character
vectors
myCorpus <- Corpus(VectorSource(tweets.df$text))

##convert to the lower case
myCorpus <- tm_map(myCorpus, content_transformer(tolower))

## remove punctuation
myCorpus <- tm_map(myCorpus, removePunctuation)

##remove numbers
myCorpus <- tm_map(myCorpus, removeNumbers)

##remove URLs
removeURLs <- function(x) gsub("http[[:alnum:]]*","",x)
myCorpus <- tm_map(myCorpus,
content_transformer(removeURLs))
## myCorpus <- tm_map(myCorpus, removeURLs, lazy=TRUE)
## Set Stop words, add "t.co", "rt", "http", "https"
myStopWords <- c(stopwords("english"), "rt","http",
"https", "t.co", "\U0001f4a5now\U0001f4a5",
"\U0001f534\U0001f4f9\U0001f1eb\U0001f1f7",
"cody\U0001f42f" )

## remove stopwords from Mycorpus
myCorpus <- tm_map(myCorpus, removeWords, myStopWords)

## copy of a Corpus to use as a dictionary for stem
completion
myCorpusCopy <- myCorpus
myCorpus <- tm_map(myCorpus, stemDocument)

##inspect the documents(tweets)
inspect(myCorpus)

##Stem Completion
##myCorpus <- tm_map(myCorpus,
content_transformer(stemCompletion),
##                dictionary = myCorpusCopy, lazy=TRUE)
##View(myCorpus)

TermDocumentMatrix
```

```
tdm <- TermDocumentMatrix(myCorpus, control =
list(wordLengths = c(1, Inf)))
tdm
inspect(tdm)
## Document Term Matrix
(dtm <- DocumentTermMatrix(myCorpus))
inspect(dtm)

(dtm_matrix <- as.matrix(dtm))############ use for other
algorityms

(WordFreq <- colSums(as.matrix(dtm)))
head(WordFreq)
length(WordFreq)

ord <- order(WordFreq)
(WordFreq[head(ord)])
(WordFreq[tail(ord)])
## Row Sums
(Row_Sum_Per_doc <- rowSums((as.matrix(dtm))))

## Normalization and weighting:TF – IDF
## I want to divide each element in each row by the sum of
the elements
## in that row. I will test this on a small matrix first to
make
## sure that it is doing what I want.
##  It would be always a good idea to test models in  small
cases.
## Create a small pretend matrix

## Using 1 in apply does rows, using a 2 does columns
(mymat = matrix(1:12,3,4))
freqs2 <- apply(mymat, 1, function(i) i/sum(i))
freqs2
## This re-organizes the matrix - so I need to transpose
back
(t(freqs2))
## OK - so this works. Now I can use this to control the
normalization of
## my matrix...
```

```
## Create a normalized version of dtm
yellowVest_M <- as.matrix(dtm)
yellowVest_M_N1 <- apply(yellowVest_M, 1, function(i)
round(i/sum(i),3))
## View(yellowVest_M_N1)
str(yellowVest_M_N1)
## transpose
yellowVest_Matrix_Norm <- t(yellowVest_M_N1)
#View(yellowVest_Matrix_Norm)


colnames(yellowVest_Matrix_Norm)[1]

## Have a look at the original and the norm to make sure
(yellowVest_M[c(1:3),c(11:13)])
(yellowVest_Matrix_Norm[c(1:3),c(11:13)])
## From the line of code
(Row_Sum_Per_doc <- rowSums((as.matrix(dtm))))


## Convert to matrix and view
yellowVest_dtm_matrix = as.matrix(dtm)
str(yellowVest_dtm_matrix)
(yellowVest_dtm_matrix[c(1:3),c(2:4)])

## Also convert to DF
yellowVest_DF <- as.data.frame(as.matrix(dtm))
str(yellowVest_DF)
##(yellowVest_DF$barricad)

## Check the data
(nrow(yellowVest_DF))  ## Each row is a revivew
rownames(yellowVest_DF)
#View(yellowVest_DF)
str(yellowVest_DF)
dim(yellowVest_DF)

## dtm_matrix[10:15, 40:45]############################
## Frequency words and association
idx <- which(dimnames(tdm)$Terms == "yellowvest")
inspect(tdm[idx + (200:203), 101:110])
```

```
##inspect frequent words
(freq.terms <- findFreqTerms(tdm, lowfreq = 15))

term.freq <- rowSums(as.matrix(tdm))
term.freq <- subset(term.freq, term.freq>=10)
df <- data.frame(term=names(term.freq), freq=term.freq)
str(df)
summary(df)
## lets's barplot
ggplot(data = df, aes(x = term, y = freq)) + geom_bar(stat
= "Identity") + xlab("Terms") + ylab("Count") +
coord_flip()

## which words are associated with 'yellowvest'?
findAssocs(tdm, "yellowvest", corlimit = 0.18)

##findAssocs(x = tdm, terms = "yellowvest",corlimit = 0.2 )
## which words are associated with 'Emmanuel Macron'
findAssocs(tdm, "pari", corlimit = 0.15)
##plot(tdm, terms = freq.terms, corThreshold = 0.12,
weighting = TRUE)
library(wordcloud)
##word Cloud
m <- as.matrix(tdm)
## calculate the frequency of words and sort it by
frequency
word.freq <- sort(rowSums(m), decreasing = TRUE)
wordcloud(words = names(word.freq), freq = word.freq,
min.freq = 10, random.order = FALSE)

## plot(tdm, term = freq.terms, corThreshold = 0.12,
weighting = T)

##################Clustering################
#############################################
## remove sparse terms
tdm2 <- removeSparseTerms(tdm, sparse = 0.95)
m2 <- as.matrix(tdm2)
## cluster term
##distance Measure
distMatrix <- dist(scale(m2))
fit_yw <- hclust(distMatrix, method = "ward.D")
```

```
plot(fit_yw)
rect.hclust(fit_yw, 6) ## cut 3 into  clusters
## rect.hclust(tree =fit_yw,k = 3)

####### k means clustering
## Create a normalized version of dtm
yellowVest_M <- as.matrix(dtm)
yellowVest_M_N1 <- apply(yellowVest_M, 1, function(i)
round(i/sum(i),3))
## View(yellowVest_M_N1)
str(yellowVest_M_N1)
## transpose
yellowVest_Matrix_Norm <- t(yellowVest_M_N1)
View(yellowVest_Matrix_Norm)

library(factoextra)
m_norm <- yellowVest_Matrix_Norm

distance1 <- get_dist(m_norm,method = "manhattan")#
fviz_dist(distance1, gradient = list(low = "#00AFBB", mid =
"white", high = "#FC4E07"))

distance2 <- get_dist(m_norm,method = "pearson")#
fviz_dist(distance2, gradient = list(low = "#00AFBB", mid =
"white", high = "#FC4E07"))

distance3 <- get_dist(m_norm,method = "canberra")#
fviz_dist(distance3, gradient = list(low = "#00AFBB", mid =
"white", high = "#FC4E07"))

distance4 <- get_dist(m_norm,method = "spearman") #kind of
ok
fviz_dist(distance4, gradient = list(low = "#00AFBB", mid =
"white", high = "#FC4E07"))

## transpose the matrix to cluster documents (tweets)
m3 <- t(m2)
## set a fixed random seed
set.seed(122)
k <- 3 #number of cluster
kmeansResult <- kmeans(m3, k)
round(kmeansResult$centers, yv = 3) ## cluster centers
```

```
## partitioning around medoids with estimation of number of
clusters
pamResults <- pamk(m3, metric="manhattan")
pamResults

k <- pamResults$nc # number of clusters identified
pamResults <- pamResults$pamobject

## Print cluster medoids
for (i in 1:k) {
  cat("cluster", i, ": ",
      colnames(pamResults$medoids)
[which(pamResults$medoids[i,]==1)], "\n")
}

##plot clustering result
layout(matrix(c(1, 2), 1,2)) ## two graph per page
plot(pamResults, col.p = pamResults$clustering)
layout(mat = 1)## return to normal screen

#### Topic Model
??`topics,TopicModel-method`
dtm <- as.DocumentTermMatrix(tdm)
lda <- LDA(dtm, k = 8) # find 8 topics
term <- terms(lda, 4) #first 4 terms of every topic
term

term <- apply(term, MARGIN = 2, paste, collapse = ", ")
## first topic identified for every document (tweet)
require(data.table) ## fore IDate

topic <- topics(lda, 1)
topics <- data.frame(date=as.IDate(tweets.df$created),
topic)
qplot(date, ..count.., data = topics, geom = "density",
      fill=term[topic], position = "stack") # not useful,
we have only one day


## Next, our current matrix does NOT have the columns as
the docs
```

```
## so we need to transpose it first....
## Run the following twice...
m_norm <- t(m_norm)
## Now scale the data
m_norm <- scale(m_norm)
str(m_norm)
## k means

kmeansFIT_1 <- kmeans(m_norm,centers=3)
#(kmeansFIT1)
summary(kmeansFIT_1)
(kmeansFIT_1$cluster)
fviz_cluster(kmeansFIT_1, data = m_norm)
## ----------------------------------------

################## Expectation Maximization ---------
## When Clustering, there are many options.
## I cannot run this as it requires more than 20 Gigs...
##library(mclust)
##ClusFI <- Mclust(m_norm,G=3)
##(ClusFI)
##summary(ClusFI)
##plot(ClusFI, what = "classification")
TweetTrans <- read.transactions(TransactionTweetsFile,
rm.duplicates = FALSE, format = "basket", sep = ",")
str(TweetTrans)
## warning(TweetTrans)
inspect(TweetTrans)


###################################
### Association Rule Mining####### Not responding
###################################
TweetTrans_rules = arules::apriori(yellowVest_Matrix_Norm,
                                   parameter =
list(support=0.01, confidence=0.01,

minlen=2, maxlen=4))

inspect(yellowVest_Matrix_Norm[1:30])

##sorted
```

```
##Support
SortedRules_sup <- sort(TweetTrans_rules, by="support",
decreasing = TRUE)
inspect(SortedRules_sup[1:20])

## Confidence
SortedRules_conf <- sort(TweetTrans_rules, by="confidence",
decreasing = TRUE)
inspect(SortedRules_conf[1:20])

## Lift
SortedRules_lift<- sort(TweetTrans_rules, by="lift",
decreasing = TRUE)
inspect(SortedRules_lift[1:20])

library(randomForest)
#### yv_fit_RF <- randomForest(x =
yellowVest_DF$yellowvest)

(dim(yellowVest_DF))
colnames(yellowVest_DF)
rownames(yellowVest_DF)
##create a train and test sets
every7_rows<-seq(1,nrow(yellowVest_DF),7)

yv_Test=yellowVest_DF[every7_rows, ]
yv_Train=yellowVest_DF[-every7_rows, ]

nrow(yv_Test)
nrow(yv_Train)
nrow(yv_Test_noLabel)
nrow(yv_Test_justLabel)
##  take the labels out of the testing data
(head(yv_Test))
yv_Test_noLabel<--c(yv_Test$yellowvest)
yv_Test_justLabel<-yv_Test$yellowvest
str(yv_Test_noLabel)
str(yv_Test_justLabel)
nrow(yv_Test_justLabel)
nrow(yv_Test_noLabel)
dim(yv_Test_noLabel)
yv_Test_noLabel
```

```
class(yv_Test_justLabel)
dim(yv_Train)
class(yv_Test_noLabel)

#### Let's apply e1071 package first
library(e1071)
## formula is yellowvest ~ x1 + x2 + …  NOTE that
yellowvest ~. is "use all to create model"
yv_NB_e1071<-naiveBayes(yv_Train$yellowvest~.,
data=yv_Train)
yv_NB_e1071_Pred <- predict(yv_NB_e1071, yv_Test_justLabel)
str(yv_NB_e1071)
yv_NB_e1071_Pred
yv_Test_
(table(yv_NB_e1071_Pred))

##Visualize
## plot(yv_NB_e1071, ylab = "Density", main = "Naive Bayes
Plot")

## using naivebayes package
library(naivebayes)
yv_NB_object<- naive_bayes(yv_Train$yellowvest~.,
data=yv_Train)
yv_NB_prediction<-predict(yv_NB_object, yv_Test_noLabel,
type = c("class"))
## head(predict(yv_NB_object, yv_Test_noLabel, type =
"prob"))
table(yv_NB_prediction)
## plot(yv_NB_object, legend.box = TRUE)

## Decision Tree Classification
library(rpart)
Treefit <- rpart(yv_Train$yellowvest ~ ., data = yv_Train,
method="class")
summary(Treefit)
predicted= predict(Treefit,yv_Test, type="class")
(Results <- data.frame(Predicted=predicted,Actual=yv_Test))
(table(Results))
fancyRpartPlot(Treefit)

######### Support Vector Machine Classifier –SVM #######
```

```
SVM_yv <- svm(yv_Train$yellowvest ~ ., data = yv_Train,
                  kernel="polynomial", cost=100,
                  scale = FALSE)
print(SVM_yv)

## Confusion Matrix for training data to check model
(SVM_pred_yv <- predict(SVM_yv,yv_Test, type="class"))
round(table(SVM_pred_yv),1)  ## pretty accurate results

######## linear WARNING: reaching max number of iterations
SVM_yv_L <- svm(yv_Train$yellowvest ~ ., data = yv_Train,
                    kernel="linear", cost=100,
                    scale = FALSE)
print(SVM_yv_L)

## Confusion Matrix for training data to check model
(SVM_pred_yv_L <- predict(SVM_yv_L,yv_Test, type="class"))
(table(SVM_pred_yv_L)) ##
######## Radial
##WARNING: reaching max number of iterations

SVM_yv_r <- svm(yv_Train$yellowvest ~ ., data = yv_Train,
                    kernel="radial", cost=100,
                    scale = FALSE)
print(SVM_yv_r)

## Confusion Matrix for training data to check model
(SVM_pred_yv_r <- predict(SVM_yv_r,yv[-1], type="class"))
(table(SVM_pred_yv_r, yv$label)) ## pretty accurate results

########### k-Nearest Neighbors algorithm—kNN
(yv_Test)
(head(yv_Train, n=15))
plot(yv_Train$label)
yv_Train
# get a guess for k
k_guess <- round(sqrt(nrow(yv_Train)))
kNN_yv <- class::knn(train = (-c(yv_Train$yellowvest),
                         test = (-c(yv_Train$yellowvest),
                         cl = yv_Train$yellowvest,
                         k = k_guess, prob = FALSE)
(print(kNN_yv))
```

```
## Check the classification accuracy
##(table(kNN_yv, yv_Train$label))## This model is working
pretty well!

##str(yv_Test)

## Let's test it on the test set now
#kNN_yv2 <- class::knn(train = yv_Train[-1],
 #                          test = yv_Test_noLabel,
  #                         cl = yv_Train$label,
   #                        k = k_guess, prob = FALSE)

## Check the classification accuracy
##(table(kNN_yv2, yv_Test_justLabel))## This model is ok
## CrossTable(x = yv_Test$label, y = kNN_yv2, prop.chisq =
F

#######  Set up Random Forest -----------------

#yv_RF <- randomForest(yv_Train$yellowvest ~ ., data =
yv_Train)
#print(yv_RF)
#yv_pred_RF <- predict(yv_RF, yv_Test_noLabel)
#(table(yv_pred_RF, yv_Test_justLabel))
#attributes(yv_RF)
#(yv_RF$confusion) ## Seems working pretty well!
(yv_RF$classes)



## Some TwitterR fuctions

??getTrends
availableTrendLocations()
getTrends(615702) ## Trends in Paris

getTrends(44418) ## Trends in London

getTrends(929398) ## Trends in Odesa Ukraine

getTrends(2122265) ## Trends in Moscow
```

```
## try sentiment analysys
## load syuzhet

##remove hash and convert to matrix
cleaned_tweets <- rm_hash(Search_DF$text)
cleaned_tweets <- as.matrix(cleaned_tweets)

##cleaning tags and convert to matrix
cleaned_tweets <- rm_tag(cleaned_tweets)
cleaned_tweets <- as.matrix(cleaned_tweets)

## remove url and convert to matrix
cleaned_tweets <- rm_url(cleaned_tweets)
cleaned_tweets <- as.matrix(cleaned_tweets)




### Practice on other useful functions of TwitterR
str(cleaned_tweets)
library(syuzhet)
movies_sentiment <- get_sentiment(cleaned_tweets)
movies_sentiment ## Positive sign positive words, Negative
sign for negative words
summary(movies_sentiment) ## summary helpfull if you have a
massive amount of data

## DF
movies_analysis <- data.frame(cleaned_tweets,
movies_sentiment)
View(movies_analysis)

??getTrends
availableTrendLocations()
getTrends(615702) ## Trends in Paris

getTrends(2343999) ## Trends in Istanbul

## User info
```

```
??getUser
movies_user <- getUser("movies")
movies_user$description
movies_user$getFollowersCount ## follower numbers
movies_user$getFriends(n = 5) ## following friends
movies_user$getFavorites()


??userTimeline
science_timeline <- userTimeline("science")
(science_timeline) ## science's most recent tweets
```