text_mining.r

Fra

Fri Nov 23 00:07:14 2018

```r
rm(list = ls())

library(rvest)
```

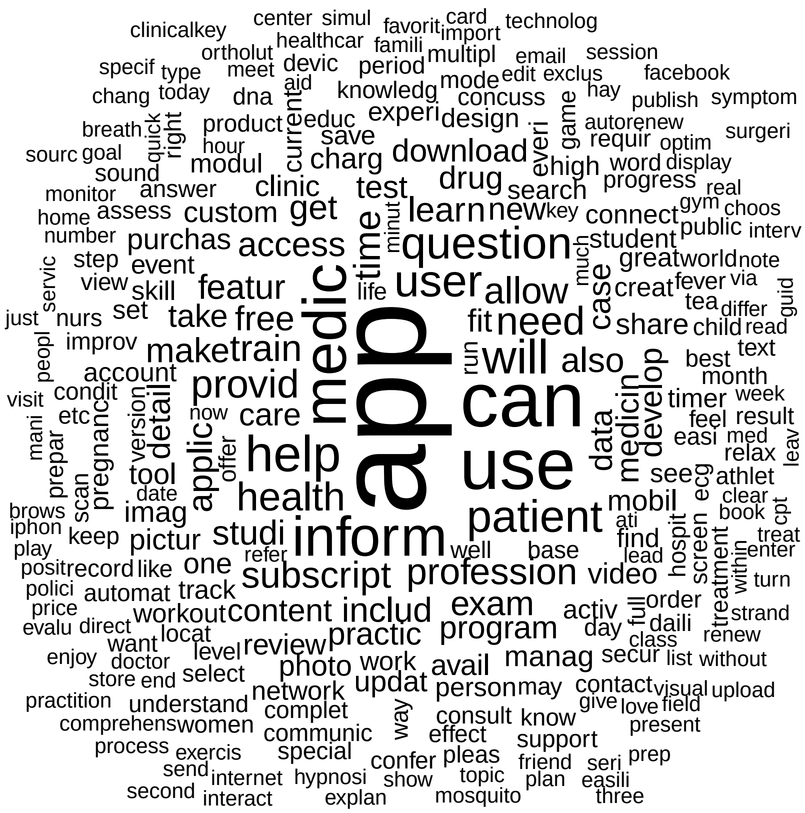
```
## Loading required package: xml2

##
## Attaching package: 'rvest'

## The following object is masked from 'package:purrr':
##
##     pluck

## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```r
library(tidyverse)
library(tm)

training_test_4<-read.csv2("training_test_set_group_4.csv",header = T)

t4_train<-training_test_4[which(training_test_4$X=="Test"),c(1,4,5)]

t4_train$NC.1.0<-factor(t4_train$NC.1.0)

table(t4_train$NC.1.0)
```

```
##
##  0  1
## 37 38
```

```r
#Data preparation â€" cleaning and standardizing text data####

#The first step in processing text data involves creating a corpus, which is a collection
# of text documents.

descr_corpus <- VCorpus(VectorSource(t4_train$Description))

#Our first order of business will be to standardize the messages to use only lowercase
# characters

descr_corpus_clean<-tm_map(descr_corpus, content_transformer(tolower))

# Let's continue our cleanup by removing numbers

descr_corpus_clean<-descr_corpus_clean%>%tm_map(.,removeNumbers)


# next task is to remove filler words such as to, and, but, and or from our SMS
```

```
# messages. These terms are known as stop words and are typically removed prior to
# text mining. This is due to the fact that although they appear very frequently, they do
# not provide much useful information for machine learning.

descr_corpus_clean<-tm_map(descr_corpus_clean,removeWords,stopwords())%>%
  tm_map(., removePunctuation) #This line remove punctuation


# Another common standardization for text data involves reducing words to their root
# form in a process called stemming. The stemming process takes words like learned,
# learning, and learns, and strips the suffix in order to transform them into the base
# form, learn. This allows machine learning algorithms to treat the related terms as a
# single concept rather than attempting to learn a pattern for each variant.

library(SnowballC)


descr_corpus_clean<-tm_map(descr_corpus_clean,stemDocument)


# The final step in our text cleanup process is to remove
# additional whitespace, using the built-in stripWhitespace()

descr_corpus_clean<-tm_map(descr_corpus_clean,stripWhitespace)

#Data preparation â€" splitting text documents into words####

# Now that the data are processed to our liking, the final step is to split the messages
# into individual components through a process called tokenization. A token is a
# single element of a text string; in this case, the tokens are words.

descr_dtm<-DocumentTermMatrix(descr_corpus_clean)

# Data preparation â€" creating training and test datasets ####

#75% train, 25% test
descr_train<-descr_dtm[1:57,]
descr_test<-descr_dtm[58:75,]

descr_train_labels<-t4_train[1:57,]$NC.1.0
descr_test_labels<-t4_train[58:75,]$NC.1.0


# Visualizing text data â€" word clouds ####
library(wordcloud)
wordcloud(descr_corpus_clean,min.freq = 7, random.order = F )
```



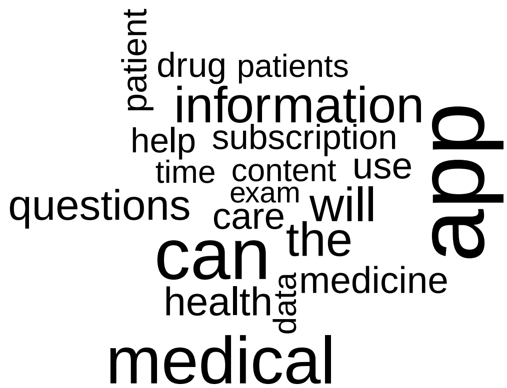
```r
#subset of NC=0 and NC=1
NC<-subset(t4_train ,NC.1.0==0 )
med<-subset(t4_train,NC.1.0==1 )

# This time, we'll use the max.words parameter to look at the 10 most common words in
# each of the two sets. The scale parameter allows us to adjust the maximum and
# minimum font size for words in the cloud

wordcloud(NC$Description, max.words = 20, scale=c(3,0.5))
```

```
## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation):
## transformation drops documents
```

```
## Warning in tm_map.SimpleCorpus(corpus, function(x) tm::removeWords(x,
## tm::stopwords())): transformation drops documents
```


```r
wordcloud(med$Description, max.words = 20, scale=c(3,0.5))
```

```
## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation):
## transformation drops documents

## Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation):
## transformation drops documents
```




```
# Data preparation â€" creating indicator features for
# frequent words ####

# The final step in the data preparation process is to transform the sparse matrix into a
# data structure that can be used to train a Naive Bayes classifier

# Currently, the sparse matrix includes over 2,500 features; this is a feature for every word that appe
# least one description

# It's unlikely that all of these are useful for classification.

#To reduce the number of features, we will eliminate any word that appear in less than 0.1 percent of t

# Finding frequent words requires use of the findFreqTerms() function in the
# tm package. This function takes a DTM and returns a character vector containing
# the words that appear for at least the specified number of times.

descr_freq_words<-findFreqTerms(descr_train, 5)

# We now need to filter our DTM to include only the terms appearing in a specified
# vector
descr_dtm_freq_train<-descr_train[ ,descr_freq_words]
descr_dtm_freq_test<-descr_test[ ,descr_freq_words]

# The Naive Bayes classifier is typically trained on data with categorical features.
# This poses a problem, since the cells in the sparse matrix are numeric and measure
```

```
# the number of times a word appears in a message. We need to change this to a
# categorical variable that simply indicates yes or no depending on whether the
# word appears at all.

convert_counts<-function(x){
  x<-ifelse(x>0,"Yes","No")
}

# We now need to apply convert_counts() to each of the columns in our sparse
# matrix.

descr_train<-apply(descr_dtm_freq_train,MARGIN = 2, convert_counts)
descr_test<-apply(descr_dtm_freq_test,MARGIN = 2, convert_counts)

# The result will be two character type matrixes, each with cells indicating "Yes" or
# "No" for whether the word represented by the column appears at any point in the
# message represented by the row.

# Training a model on the data ####

# Now that we have transformed the raw SMS messages into a format that can be
# represented by a statistical model, it is time to apply the Naive Bayes algorithm. The
# algorithm will use the presence or absence of words to estimate the probability that a
# given app's description is not medical

library(e1071)

descr_classifier<-naiveBayes(descr_train,descr_train_labels)

# Evaluating model performance

# To evaluate, we need to test its predictions on
# unseen descriptions

# The predict() function is used to make the predictions

descr_test_pred<-predict(descr_classifier, descr_test)

# To compare the predictions to the true values, we'll use the CrossTable() function
# in the gmodels package

library(gmodels)

CrossTable(descr_test_pred, descr_test_labels,
           prop.chisq = F, prop.t = F,
           dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
```

```
## |-------------------------|
##
##
## Total Observations in Table:  18
##
##
##              | actual
##    predicted |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |         6 |         4 |        10 |
##              |     0.600 |     0.400 |     0.556 |
##              |     0.750 |     0.400 |           |
## -------------|-----------|-----------|-----------|
##            1 |         2 |         6 |         8 |
##              |     0.250 |     0.750 |     0.444 |
##              |     0.250 |     0.600 |           |
## -------------|-----------|-----------|-----------|
## Column Total |         8 |        10 |        18 |
##              |     0.444 |     0.556 |           |
## -------------|-----------|-----------|-----------|
##
##
```

```r
# This time we'll build a Naive Bayes model as done
# earlier, but this time set laplace=1

descr_classifier2<-naiveBayes(descr_train, descr_train_labels,laplace = 1)

descr_test_pred2<-predict(descr_classifier2,descr_test)

CrossTable(descr_test_pred2, descr_test_labels,
           prop.chisq = F, prop.t = F,
           dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  18
##
##
##              | actual
##    predicted |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |         8 |         5 |        13 |
##              |     0.615 |     0.385 |     0.722 |
##              |     1.000 |     0.500 |           |
## -------------|-----------|-----------|-----------|
##            1 |         0 |         5 |         5 |
```

```
##              |     0.000 |     1.000 |     0.278 |
##              |     0.000 |     0.500 |           |
## -------------|-----------|-----------|-----------|
## Column Total |         8 |        10 |        18 |
##              |     0.444 |     0.556 |           |
## -------------|-----------|-----------|-----------|
##
##
```