Sarcasm Detection with Naive Bayes Classification for Tweets Data

**What is Sarcasm:**

**Sarcasm**, which is both positively funny and negatively nasty, plays an important part in human social interaction.

Generally, why people do sarcasm just because they want to show negativity towards something.

**Problem:**

We have given the **Tweets Dataset** and our goal is to predict whether a given text is sarcastic or not.

The data contains 2 columns, one is text where all tweets are there and another one is label that indicates “A given text is sarcastic or not”. Score is either 1 (for positive) or 0 (for negative) which indicates whether sentence has a sarcasm or non-sarcasm.

**Code:**

**Step 1: The library which are helpful to do the analysis:**

* Readxl: That can read the excel file.
* tm: That is useful to apply text-mining.
* Wordcloud: That can plot the word cloud.
* RcolorBrewer: That can use to put ready colors to the given wordcloud.
* E1071: Function for latent class analysis.
* gmodels: That can use for various model fitting.
* navieBayes: For the requirement purpose.

**Step 2: Exploring and Preparing the Data**

tweet <- read.csv(file.choose(), header = T,fill = TRUE, stringsAsFactors = F)

tweet$sarcasm <- factor(tweet$sarcasm)

**#Check the positive and negative scores**

table(tweet$sarcasm)

0 1

1018 974

**#Creating a corpus form text**

tweet\_corpus <- VCorpus(VectorSource(tweet$text))

**#Create a document-term sparse matrix directly from the corpus**

tweet\_dtm <- DocumentTermMatrix(tweet\_corpus, control = list(

tolower = TRUE,

removeNumbers = TRUE,

stopwords = TRUE,

removePunctuation = TRUE,

stemming = TRUE

))

**# Creating training and test datasets**

tweet\_dtm\_train <- tweet\_dtm[1:1593, ]

tweet\_dtm\_test <- tweet\_dtm[1594:1992, ]

**# Also save the labels**

tweet\_train\_labels <- tweet[1:1593, ]$sarcasm

tweet\_test\_labels <- tweet[1594:1992, ]$sarcasm

**#Checking that the proportion is same for both train and test data**

prop.table(table(tweet\_train\_labels))

tweet\_train\_labels

0 1

0.517263 0.482737

prop.table(table(tweet\_test\_labels))

0 1

0.4862155 0.5137845

**#proportion is not same on train and test data**

**#We will need to use sampling**

Sample takes a sample of the specified size from the elements of x using either with or without replacement.

**# Create random samples**

set.seed(123)

train\_index <- sample(1000, 800)

tweet\_train <- tweet[train\_index, ]

tweet\_test <- tweet[-train\_index, ]

**# check the proportion of class variable**

prop.table(table(tweet\_train$sarcasm))

0 1

0.52125 0.47875

prop.table(table(tweet\_test$sarcasm))

0 1

0.5041946 0.4958054

This is much better.

train\_corpus <- VCorpus(VectorSource(tweet\_train$text))

test\_corpus <- VCorpus(VectorSource(tweet\_test$text))

**Word Cloud Visualization**

**# subset the training data into spam and ham groups**

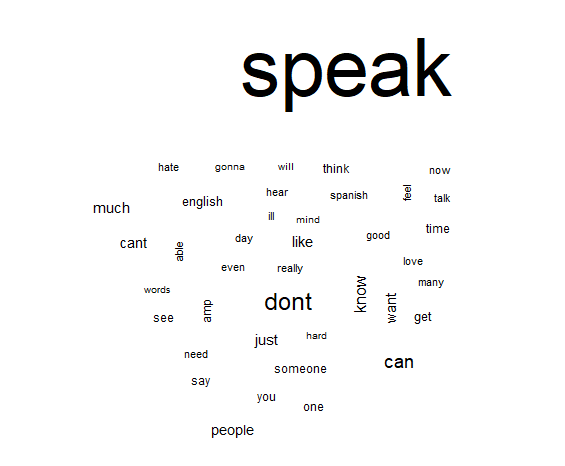
positive <- subset(tweet\_train, sarcasm == 1)

negative <- subset(tweet\_train, sarcasm == 0)

wordcloud(positive$text, max.words = 40, scale = c(5, 0.5))

**This are the Positive words**

wordcloud(negative$text, max.words = 40, scale = c(5, 0.5))

**This are the negative words**

**# create a document-term sparse matrix directly for train and test**

train\_dtm <- DocumentTermMatrix(train\_corpus, control = list(

tolower = TRUE,

removeNumbers = TRUE,

stopwords = TRUE,

removePunctuation = TRUE,

stemming = TRUE

))

test\_dtm <- DocumentTermMatrix(test\_corpus, control = list(

tolower = TRUE,

removeNumbers = TRUE,

stopwords = TRUE,

removePunctuation = TRUE,

stemming = TRUE

))

train\_dtm

<<DocumentTermMatrix (documents: 800, terms: 2218)>>

Non-/sparse entries: 6856/1767544

Sparsity : 100%

Maximal term length: 61

Weighting : term frequency (tf)

test\_dtm

<<DocumentTermMatrix (documents: 1192, terms: 2867)>>

Non-/sparse entries: 9897/3407567

Sparsity : 100%

Maximal term length: 46

Weighting : term frequency (tf)

**# create function to convert counts to a factor**

convert\_counts <- function(x) {

x <- ifelse(x > 0, "Yes", "No")

}

**# apply() convert\_counts() to columns of train/test data**

train\_dtm\_binary <- apply(train\_dtm, MARGIN = 2, convert\_counts)

test\_dtm\_binary <- apply(test\_dtm, MARGIN = 2, convert\_counts)

## **Step 3: Training a model on the data**

tweet\_classifier <- naiveBayes(as.matrix(train\_dtm\_binary), tweet\_train$sarcasm)

## **Step 4: Evaluating model performance**

tweet\_test\_pred <- predict(tweet\_classifier, as.matrix(test\_dtm\_binary))

head(tweet\_test\_pred)

[1] 1 0 1 0 0 0

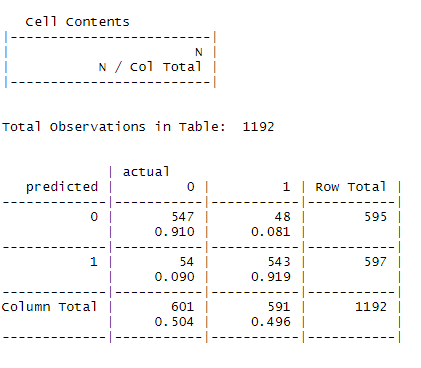
Levels: 0 1

CrossTable(tweet\_test\_pred, tweet\_test$sarcasm,

prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,

dnn = c('predicted', 'actual'))

**#accuracy is 0.91**



## **Step 5: Improving model performance**

**Use Laplace smoothing because the train document term matrix does not contain the terms from the test data**.

tweet\_classifier2 <- naiveBayes(as.matrix(train\_dtm\_binary), tweet\_train$sarcasm, laplace = 1)

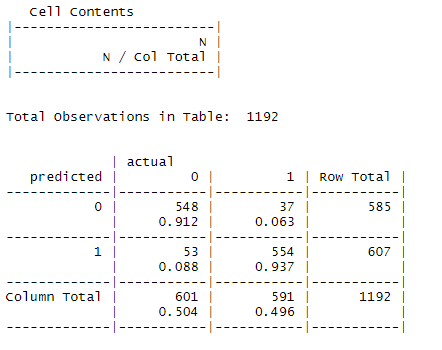
tweet\_test\_pred2 <- predict(tweet\_classifier2, as.matrix(test\_dtm\_binary))

CrossTable(tweet\_test\_pred2, tweet\_test$sarcasm,

prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,

dnn = c('predicted', 'actual'))

**#accuracy is 0.92**



**#improving accuracy**

tweet\_classifier3 <- naiveBayes(as.matrix(train\_dtm\_binary), tweet\_train$sarcasm, laplace = .5)

tweet\_test\_pred3 <- predict(tweet\_classifier3, as.matrix(test\_dtm\_binary))

CrossTable(tweet\_test\_pred3, tweet\_test$sarcasm,

prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,

dnn = c('predicted', 'actual'))

**#accuracy 0.91**

