Use Naive Bayes for Classification

Dr. Nishat Mohammad

02/14/2024

Load some needed packages

```r
# Text mining
library(tm)
```

```
## Loading required package: NLP
```

```r
# Text stemming
library(SnowballC)
# Word cloud visualization
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```r
# Naive Bayes Model
library(e1071)

#library(gmodels)
```

1. Exploring and Preparing the Data

1.1. Loading the data for SMS Spam

```r
# Read csv file into sms_dt
sms_dt <- read.csv("spammsg.csv")

# Look at the structure of the data
str(sms_dt)
```

```
## 'data.frame':    5574 obs. of  2 variables:
##  $ type: chr  "ham" "ham" "spam" "ham" ...
##  $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... C
```

```r
# Look at the first few rows of the data
head(sms_dt)
```

```
##   type
## 1  ham
## 2  ham
## 3 spam
## 4  ham
## 5  ham
## 6 spam
##
## 1                                              Go until jurong point, crazy.. Available only in bugis
## 2
## 3 Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive en
## 4
## 5                                                                                               Nah I
## 6         FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you u
```

```
# A few more lines for my understanding
# Look at the dimensions
dim(sms_dt)
```

```
## [1] 5574    2
```

```
# Check for missing values
any(is.na(sms_dt))
```

```
## [1] FALSE
```

1.2. Factoring Categorical Variables.

```
# Factor type
sms_dt$type <- factor(sms_dt$type)

# Look at the structure
str(sms_dt$type)
```

```
##  Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
```

Categorical variables are to be factored and in this data the type variable is categorical diving the data into ham and spam categories.
After factoring the ham category is assigned to 1 and spam to 2.

```
# Look at the distribution of ham and spam in a table
table(sms_dt$type)
```

```
##
##  ham spam
## 4827  747
```

We check the overall distribution of spam and ham and find 747 an 4827 values respectively.

2. Data Preparation.

2.1. Cleaning and Standardizing Text Data.

```r
# Create a VCorpus Object
sms_dt_corpus <- VCorpus(VectorSource(sms_dt$text))
print(sms_dt_corpus)
```

2.1.1. Text Mining and Natural Language Processing.

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5574
```

```r
# Look at the first two documents in general
inspect(sms_dt_corpus[1:2])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 111
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 29
```

```r
# Look at the text in the first document
sms_dt_first_doc <- as.character(sms_dt_corpus[[1]])
#sms_dt_first_doc

# Look at the text for the first 2 docs
sms_dt_1st2nd_doc <- lapply(sms_dt_corpus[1:2], as.character)
#sms_dt_1st2nd_doc
```

The `VCorpus()` function from the tm Package in R is used for text mining and natural language processing, it takes a vector of text values and returns a Vcorpus object which is a collection of multiple text documents. For our data we can see by printing the VCorpus Object (sms_dt_corpus) is printed to find that we have 5574 documents. SO it acts as a container for the text data and allows for use of more methods and functions to manipulate to our choices.
Wrapping the Vcorpus Object in `inspect()` function we can see more details about the data, looking at the first 2 documents in the corpus.
There are no copus_specific metadata and no indexed attributes for the metadata.
Both doucments are Plain Text with 7 metadata attributes, the first one has 111 characters in the content while the second document has 29 characters in its content. Looking at the text in the first document using the `as.inspect()` function, let us look at it below:

"Go until jurong point, crazy.. Available only in bugis n great world la e buffet… Cine there got amore wat…".

Looking at the text for the first two documents by using `lapply()` function to apply the `as.character()` function on a list of the first two documents from the corpus. Please view them below:

"Go until jurong point, crazy.. Available only in bugis n great world la e buffet… Cine there got amore wat…", "Ok lar… Joking wif u oni…".

```
# Transform all characters to lower case
sms_dt_corpus_clean <- tm_map(sms_dt_corpus, content_transformer(tolower))
# Look at the first document in cleaned data
sms_dt_cleand_first_doc <- as.character(sms_dt_corpus_clean[[1]])
#sms_dt_cleand_first_doc
```

2.1.2. Clean, Standaradize and Transform the data. Converting all letters to lower case was done by wrapping the corpus in the `tm_map()` function and specifying `tolower` in the `content_transformer()` function.
After transforming to lower cases the first document has all lower case alphabets as was expected and looks like this:

"go until jurong point, crazy.. available only in bugis n great world la e buffet… cine there got amore wat…".

The `content_transformer()` function can be used for various purposes, from making abbreviations to full forms to finding group of words or creating content based features as part of feature engineering.

```
# Take numbers off form data
sms_dt_corpus_clean <- tm_map(sms_dt_corpus_clean, removeNumbers)
```

Passing the cleaned orpus to `tm_map` function with `removeNumbers` option takes off the numbers in the data.

```
# Remove filler words
sms_dt_corpus_clean <- tm_map(sms_dt_corpus_clean, removeWords, stopwords())
```

Wrapping the corpus in `tm_map()` function with the `removeWords` option and `stopwords()` function takes off the filler words in the data.

```
# Take off punctuations
sms_dt_corpus_clean <- tm_map(sms_dt_corpus_clean, removePunctuation)

# Function to adjy=ust any settings in the remove punctuation tasks
replacePunctuation <- function(x) {
    gsub("[[:punct:]]+", " ", x)
}
```

Using the `removePunctuation` option in `tm_map()` function takes off the punctuations in the data. The punctuations to be removed can be specified using regex through the gsub() function as shown in the code above.

```
# remove same words in different tenses or forms
sms_dt_corpus_clean <- tm_map(sms_dt_corpus_clean, stemDocument)
```

Using the `stemDocument` option in `tm_map()` removes various forms of the same word such as plurals, tenses among others.

```r
sms_dt_corpus_clean <- tm_map(sms_dt_corpus_clean, stripWhitespace)
```

We take care of white spaces using the `stripWhitespace` option in `t_map()` function.

```r
# Look at the cleaned data again
print(sms_dt_corpus_clean)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 5574
```

```r
# Look at the first two docs
inspect(sms_dt_corpus_clean[1:2])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 76
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 21
```

```r
sms_dt_clean1st2nd_doc <- lapply(sms_dt_corpus_clean[1:2], as.character)
#sms_dt_clean1st2nd_doc
```

After these steps we have arrived at the following for the first 2 docments:

"go jurong point crazi avail bugi n great world la e buffet cine got amor wat", "ok lar joke wif u oni".

2.2. Splitting Text Document into Words.

```r
# Create a DTM
sms_dt_dtm <- DocumentTermMatrix(sms_dt_corpus_clean)
sms_dt_dtm
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6592)>>
## Non-/sparse entries: 42608/36701200
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

The `DocumentTermMatrix()` function from the `tm` package is used to create a sparse matrix called the document-term matrix (DTM) by taking a corpus object and making the documents the rows and the words will become the columns.
The cleaned data was passed to the function and can be viewed above.

```r
# Function to carry out all the earlier steps
sms_dt_dtm2 <- DocumentTermMatrix(sms_dt_corpus, control = list(
    tolower = TRUE,
    removeNumbers = TRUE,
    stopwords = TRUE,
    removePunctuation = TRUE,
    stemming = TRUE
))
sms_dt_dtm2
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6995)>>
## Non-/sparse entries: 43713/38946417
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

The original corpus can be passed to the function and options specified therein to clean the data this can cause a slight differenc edue to the sequence of cleaning steps. Please view the corpus cleaned with the `DocumentTermMatrix()` function above.

2.3. Creating Training and Test Sets.

```r
# Get 75% of the rows
tot_rows <- nrow(sms_dt_dtm)

# Limits for training and test
train_limit <- round(0.75 * tot_rows, 0)
test_limit <- tot_rows - train_limit


# Training set
sms_dt_dtm_train <- sms_dt_dtm[1:train_limit, ]
# Test set
sms_dt_dtm_test <- sms_dt_dtm[(train_limit+1):test_limit, ]

# Labels
sms_dt_train_labels <- sms_dt[1:train_limit, ]$type
sms_dt_test_labels <- sms_dt[(train_limit+1):test_limit, ]$type

# Check the distribution between ham and spam
prop.table(table(sms_dt_train_labels))
```

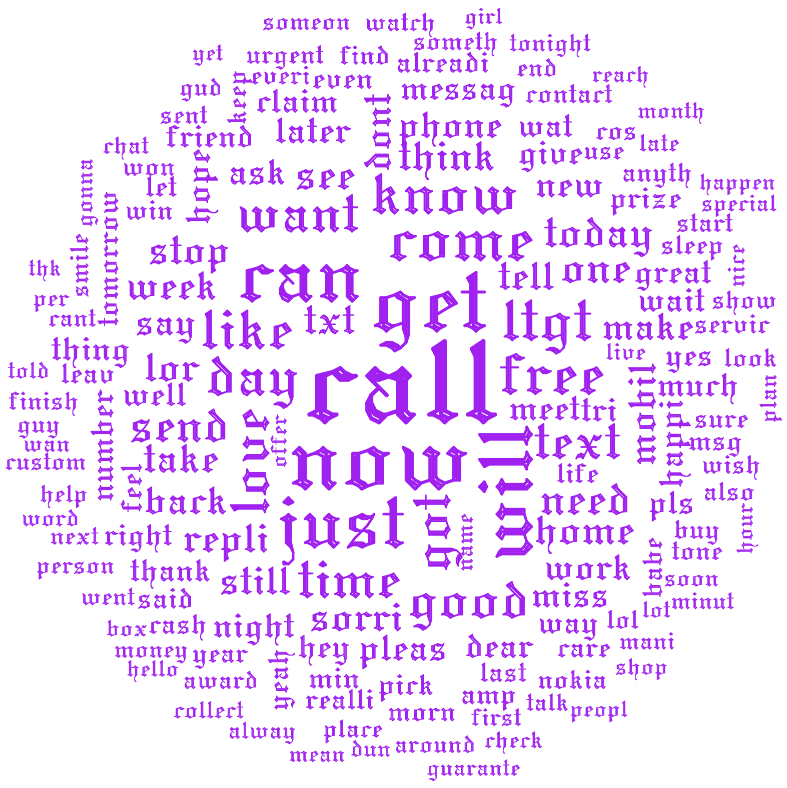
```
## sms_dt_train_labels
##       ham      spam
## 0.8648325 0.1351675
```

Here the data was split into training and test sets based on 75:25 ratio respectively. The proportion for ham and spam were checked using the `prop.table()` function. The ham got 86% and the spam took about 13%.

3. Visualization of text data Using Word Clouds.

3.1. Word cloud for the cleaned data

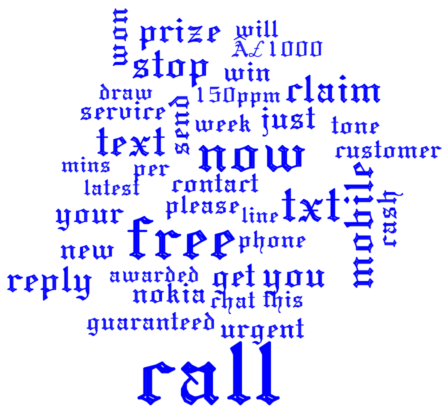
```r
wordcloud(sms_dt_corpus_clean, min.freq = 50, random.order = FALSE, colors = "purple", vfont=c("gothic
```



The figure above shows all the words in the data after cleaning

3.2. Word Cloud for Spam.

```r
# Sample spam out of the main data
spam_dt <- subset(sms_dt, type == "spam")
wordcloud(spam_dt$text, max.words = 40, scale = c(3, 0.5), colors = "blue", vfont=c("gothic english","p
```



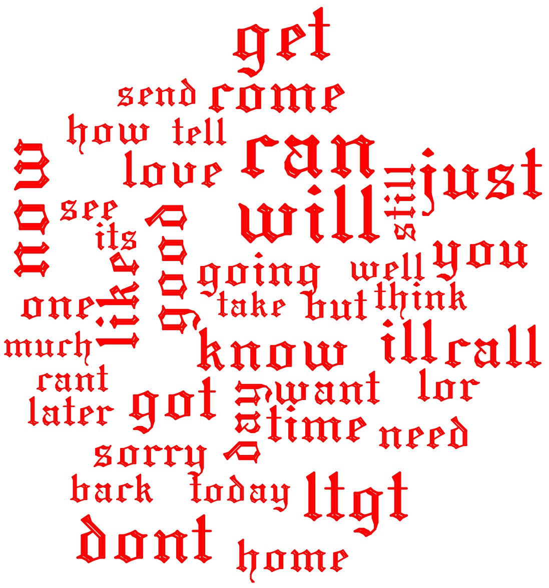
The figure above shows the word cloud for the spam data alone. R has dropped some documents probably due to its own cleaning mechanisms. The most frequent words here are `call`, `free`, `txt`, `mobile`.

3.3. Word Cloud for Ham.

```r
# Sample ham form main data
ham_dt <- subset(sms_dt, type == "ham")
wordcloud(ham_dt$text, max.words = 40, scale = c(3, 0.5), colors = "red", vfont=c("gothic english","pla
```

```
## 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
intersect(ham_dt,spam_dt)
```

```
## list()
```

This figure shows the ham data in a word cloud. R has dropped documents here as well. The most frequent words are `can`, `get`, `will`, `just`, `know`.
Thus Naive Bayes has picked up key words form the data.

4. More Data Preparation by Creating Indicator Features for frequent words.

4.1 The frequent words

```r
# Get the frequent words in training set
sms_dt_freq_words <- findFreqTerms(sms_dt_dtm_train, 5)
# Look at the structure
str(sms_dt_freq_words)
```

```
##  chr [1:1161] "£wk" "abiola" "abl" "abt" "accept" "access" "account" ...
```

Frequent words are extracted using the training dtm data and the structure can be seen above. There are 861 frequent words.

4.2. Assign features to train and test set.

```
sms_dt_dtm_freq_train <- sms_dt_dtm_train[ , sms_dt_freq_words]
print(sms_dt_dtm_freq_train)
```

```
## <<DocumentTermMatrix (documents: 4180, terms: 1161)>>
## Non-/sparse entries: 25259/4827721
## Sparsity           : 99%
## Maximal term length: 13
## Weighting          : term frequency (tf)
```

```
sms_dt_dtm_freq_test <- sms_dt_dtm_test[ , sms_dt_freq_words]
print(sms_dt_dtm_freq_test)
```

```
## <<DocumentTermMatrix (documents: 2788, terms: 1161)>>
## Non-/sparse entries: 16670/3220198
## Sparsity           : 99%
## Maximal term length: 13
## Weighting          : term frequency (tf)
```

The training set has 4180 documents and the test set has 2788 documents.

4.3. Change Numeric Naive Bayes to Categorical.

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

sms_traind <- apply(sms_dt_dtm_freq_train, MARGIN = 2,
    convert_counts)
sms_testd  <- apply(sms_dt_dtm_freq_test, MARGIN = 2,
    convert_counts)
```

Before modelling, the numeric values in the train and test dtms were changed to Yes for 0 and No for 1 by using the covert function and applying it over the columns of the frequency train and test sets by using `MARGIN = 2`. This makes the data uniform in class.

5. Train Naive Bayes Model on the Data.

```
# Use naive bayes from e1071 package
sms_dt_classifier <- naiveBayes(sms_traind, sms_dt_train_labels)
```

To Create the model, the `naiveBayes()` function from the `e1071` package is used in the code chunk above by wrapping the frequency train set and the train set labels in it.

6. Evaluate Model

6.1. Apply the Test Data for Evaluation

```r
sms_dt_test_predict <- predict(sms_dt_classifier, sms_testd)
```

A vital step in modelling is evaluation and here it has been carried out by using the `predict()` function with the model and the frequency test set passed to it.

6.2. Create Cross Table (Confusion Table).

```r
# Load package
library(gmodels)

# Get cross table
CrossTable(sms_dt_test_predict,
           sms_dt_test_labels,
           prop.chisq = FALSE,
           prop.c = FALSE,
           prop.r = FALSE,
           dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  2788
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      2420 |        37 |      2457 |
##              |     0.868 |     0.013 |           |
## -------------|-----------|-----------|-----------|
##         spam |         5 |       326 |       331 |
##              |     0.002 |     0.117 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      2425 |       363 |      2788 |
## -------------|-----------|-----------|-----------|
##
##
```

The cross table above shows actual ham are 2425 and the model predicted a total of 2457 ham, slightly higher to take note of.

The actual spam are 363 but the model predicted 331 spam, slightly lower, implying that the model is picking spam as ham incorrectly.
Let us get some probabilties with the code chunk below.

```r
# Probability the model predicts spam
Pspam1 <- (5 +37)/2788
# Probability the model predicts spam incorrectly
PIncorrect_spam1 = 37/363
# Probability the model predicts ham
Pham1 <- (2420+326)/2788
# Probability the model predicts ham incorrectly
PIncorrect_ham1 = 5/2425
# Consider ham is the positive observation
TP1<- 2420
TN1 <-326
FP1 <- 37
FN1 <- 5
total_obs <- 2788
# Accuracy of model
Accuracy1 <- (TP1 +TN1)/ total_obs
Sensitivity1 <- TP1/(TP1+FN1)
Specifictiy1 <- TN1/(TN1 + FP1)
```

The probability that this model will predict spam is:
P(spam) = “0.01506456241033”.
The probability that this model will predict ham is:
P(ham) = “0.98493543758967”.

The probability that the model will predict ham incorrectly is:
Incorrect ham = “0.00206185567010309”.
The probability that the model will predict spam incorrectly is:
Incorrect spam = “0.101928374655647”.

Accuracy of this model is “0.98493543758967”.
Sensitivity of this model is “0.997938144329897”.
Specificity of this model is “0.898071625344353”.

7. Improve Model Performance.

```r
# Train new model
sms_dt_classifier2 <- naiveBayes(sms_traind, sms_dt_train_labels, laplace = 1)

# Test the new model
sms_dt_test_pred2 <- predict(sms_dt_classifier2, sms_testd)

# Create cross table

CrossTable(sms_dt_test_pred2, sms_dt_test_labels,
    prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
    dnn = c('predicted', 'actual'))
```

```
##
```

```
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  2788
##
##
##               | actual
##     predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      2400 |        20 |      2420 |
##              |     0.861 |     0.007 |           |
## -------------|-----------|-----------|-----------|
##         spam |        25 |       343 |       368 |
##              |     0.009 |     0.123 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      2425 |       363 |      2788 |
## -------------|-----------|-----------|-----------|
##
##
```

The cross table above is created after using la Place constant to improve prediction.
The total number of ham predicted by this second model is 2420 where as the actual are 2425. The total number of spam predicted by the second model are 368 whereas the actual are 363.
Let us examine the corss table with the code chunk below.

```
# Consider ham is the positive observation
TP2<- 2200
TN2 <-343
FP2 <- 20
FN2 <- 25

# Accuracy of model
Accuracy2 <- (TP2 +TN2)/ total_obs
Sensitivity2 <- TP2/(TP2+FN2)
Specifictiy2 <- TN2/(TN2 + FP2)
```

Accuracy of the second model is "0.912123385939742".
Sensitivity of the second model is "0.98876404494382".
Specificity of the second model is "0.944903581267218".

```
# Organize data
analytic_methods <- c("Accuracy", "Sensitivity", "Specificity")
mod1data <- c(Accuracy1,Sensitivity1, Specifictiy1)
mod2data <- c(Accuracy2, Sensitivity2,Specifictiy2)
# Tabulate data
compare_models_table <- data.frame(analytic_methods,mod1data, mod2data)
knitr::kable(compare_models_table)
```

analytic_methods	mod1data	mod2data
Accuracy	0.9849354	0.9121234
Sensitivity	0.9979381	0.9887640
Specificity	0.8980716	0.9449036

From the table above, although both models are highly competitive on the accuracy and sensitivity and specificity:
The second model surpasses the first model on specificity alone.
The first model remains more accurate than the second and also more sensitive.
I thought about the word "now" being common to both ham and spam categories, and if it would affect the outcome of the Naive Bayes prediction, but since it is only one word that is common I inferred that it would not be relevant enough to consider this as blocker to accuracy of the models.
Both models can be relied upon for different purposes though based on the business needs and priorities, since they are both highly accurate.