# Mobile SMS Spam Detection

*Durga Gaddam*

*August 25, 2016*

**Objective**

The main objective of this article is to use Navie Bayes Classification algorithm concept in Machine Learning(Supervised Learning) to detect which messages are spam. This algorithm will automatically detect the incoming message by reading the keywords to determine it as spam.

**What is Navie Bayes Classification**

This concept uses probability i.e likely hood of happening of an event to categorize data. Conditional probability is mostly used in Navie Bayes classification and is considered as one of the strongest method used for classification learning tasks(Lantz,2015)

Algorithm is a sequence of procedures or rules given to a computer, when followed guarantees the result

In every Machine learning model, five steps are required to complete the model.

**Step-1: Collecting the Data**

**Step-2: Exploring and Preparing the data**

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

**Step-4: Evaluating model performance**

**Step-5: Improving model perfomance**

**Step-1: Collecting the Data**

The present data is collected from http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

**Description of Data**

The extracted data consists of Short Message Services(SMSs) collected from mobile phone users.

**Ham(Non-spam) examples:**

1) *Better. Made up for Friday and stuffed myself like a pig yesterday. Now I feel bleh. But at least its not writhing pain kind of bleh.*
2) *No we sell it all so we'll have tons if coins. Then sell our coins to someone thru paypal. Voila! Money back in life pockets:)*

**Spam examples:**

1) *URGENT! We are trying to contact U. Todays draw shows that you have won a Â£800 prize GUARANTEED. Call 09050001808 from land line. Claim M95. Valid12hrs only',1*

2) *Please call our customer service representative on FREEPHONE 0808 145 4742 between 9am-11pm as you have WON a guaranteed Â£1000 cash or Â£5000 prize!*

As you can see that the ham messages do not contain any words like 'FREE', 'URGENT' and other Money related excitements, There is a likelyhood that the messages containing these words are spams. And it is less likely that messages with this words are hams.

**Step-2: Exploring and Preparing the data**

```
sms <- read.csv("F:/R PRACTICE/Smsfilter/sms_spam.csv", stringsAsFactors = FALSE)

head(sms,2)
```

```
##   Type                                               Text
## 1  ham Hope you are having a good week. Just checking in
## 2  ham                             K..give back my thanks.
```

```
tail(sms,1)
```

```
##      Type                              Text
## 5559  ham Shall call now dear having food
```

```
str(sms)
```

```
## 'data.frame':    5559 obs. of  2 variables:
##  $ Type: chr  "ham" "ham" "ham" "spam" ...
##  $ Text: chr  "Hope you are having a good week. Just checking in" "K..give back my thanks." "Am also
```

The give data set contains 5559 observations. Each sms is categorized into ham or spam. There are two variables "type" and "text"

```
sapply(sms,class)
```

```
##        Type        Text
## "character" "character"
```

The given variable "type" data is in the form of character vector and needs to be converted into factor variable to get levels.

```
sms$Type <- as.factor(sms$Type)
class(sms$Type)
```

```
## [1] "factor"
```

```
str(sms$Type)
```

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

```
table(sms$Type)
```

```
##
##  ham spam
## 4812  747
```

### cleaning the data set

Currently the data set is in raw form and needs to be cleaned. Understanding each punctuation mark and word is done using a package called *tm* in R.

```
###install.packages("tm")
### library(tm)
```

### Corpus:

Corpus is a collection of text data or documents, which is a reference of text that we need to understand.

In our context, sms_spam.csv is the data source and a variable needs to be created to read the data and understand it.

The *tm* Packages contains two functions *Vcorpus()* and *Pcorpus()*. Vcorpus stands for Volatile Corpus is stored in RAM and is temprory, Pcorpus is Permanent Corpus which can be stored in harddisk. *VectorSource()* is another function used to read the data to corpus

```
require(tm)
```

```
## Loading required package: tm
```

```
## Warning: package 'tm' was built under R version 3.3.1
```

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

```
sms_corpus <- VCorpus(VectorSource(sms$Text))
```

Each text message is stored as a document in Corpus. *tm* contains a function called *inspect()* which is used to summarize the data in required document

```
inspect(sms_corpus[20])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 157
```

```
as.character(sms_corpus[[20]])
```

```
## [1] "U can WIN £100 of Music Gift Vouchers every week starting NOW Txt the word DRAW to 87066 TsCs w
```

### tm_map()

tm_map() function in *tm* package is used to clean the data.

### Standardizing the data

To standardize the data, all the characters or words are to be converted to lower case, this can be done using a fuction called tolower() to transform this we also use a function called content_transformer(). we also need to remove the numbers, to do this we need to use *removeNumbers* function in *tm*

```
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))

as.character(sms_corpus[[500]])
```

```
## [1] "Jordan got voted out last nite!"
```

```
as.character(sms_corpus_clean[[500]])
```

```
## [1] "jordan got voted out last nite!"
```

```
sms_corpus_clean <- tm_map(sms_corpus_clean,removeNumbers)



as.character(sms_corpus[[5]])
```

```
## [1] "okmail: Dear Dave this is your final notice to collect your 4* Tenerife Holiday or #5000 CASH a
```

```
as.character(sms_corpus_clean[[5]])
```

```
## [1] "okmail: dear dave this is your final notice to collect your * tenerife holiday or # cash award!
```

### StopWords

Stopwords are frequently used words in a language. Here are the list of Stop words. These are to be eliminated from the text before text mining. This transformation is done thorugh a function in *tm* called *removeWords*

```
stopwords()
```

```
##   [1] "i"          "me"         "my"         "myself"     "we"
##   [6] "our"        "ours"       "ourselves"  "you"        "your"
##  [11] "yours"      "yourself"   "yourselves" "he"         "him"
##  [16] "his"        "himself"    "she"        "her"        "hers"
##  [21] "herself"    "it"         "its"        "itself"     "they"
##  [26] "them"       "their"      "theirs"     "themselves" "what"
```

```
##  [31] "which"      "who"        "whom"       "this"       "that"
##  [36] "these"      "those"      "am"         "is"         "are"
##  [41] "was"        "were"       "be"         "been"       "being"
##  [46] "have"       "has"        "had"        "having"     "do"
##  [51] "does"       "did"        "doing"      "would"      "should"
##  [56] "could"      "ought"      "i'm"        "you're"     "he's"
##  [61] "she's"      "it's"       "we're"      "they're"    "i've"
##  [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
##  [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
##  [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
##  [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
##  [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
##  [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
##  [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
## [101] "who's"      "what's"     "here's"     "there's"    "when's"
## [106] "where's"    "why's"      "how's"      "a"          "an"
## [111] "the"        "and"        "but"        "if"         "or"
## [116] "because"    "as"         "until"      "while"      "of"
## [121] "at"         "by"         "for"        "with"       "about"
## [126] "against"    "between"    "into"       "through"    "during"
## [131] "before"     "after"      "above"      "below"      "to"
## [136] "from"       "up"         "down"       "in"         "out"
## [141] "on"         "off"        "over"       "under"      "again"
## [146] "further"    "then"       "once"       "here"       "there"
## [151] "when"       "where"      "why"        "how"        "all"
## [156] "any"        "both"       "each"       "few"        "more"
## [161] "most"       "other"      "some"       "such"       "no"
## [166] "nor"        "not"        "only"       "own"        "same"
## [171] "so"         "than"       "too"        "very"
```

```
sms_corpus_clean <- tm_map(sms_corpus_clean,removeWords,stopwords())

as.character(sms_corpus[[5]])
```

```
## [1] "okmail: Dear Dave this is your final notice to collect your 4* Tenerife Holiday or #5000 CASH au
```

```
as.character(sms_corpus_clean[[5]])
```

```
## [1] "okmail: dear dave    final notice  collect  * tenerife holiday  # cash award! call   landline. t
```

```
sms_corpus_clean <- tm_map(sms_corpus_clean,removePunctuation)
as.character(sms_corpus_clean[[5]])
```

```
## [1] "okmail dear dave    final notice  collect   tenerife holiday   cash award call   landline tcs sa
```

**Stemming:**

Stemming is reduction of words using root form. Example, “parenting”,“parents”,“parent” has a root form of parent(suffix). Every word of this root are converted automatically removed. Stemming is used in R through a package called “SnowballC”

```
###install.packages("SnowballC")
### library(SnowballC)

require(SnowballC)
```

```
## Loading required package: SnowballC
```

```
wordStem(c("parenting","parents","parent"))
```

```
## [1] "parent" "parent" "parent"
```

```
### StemDocument() function is used in transforming the entire corpus text documents

sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

reducing the white spaces that were created previously by deleting words, numbers and punctuation marks

```
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)

as.character(sms_corpus[[3]])
```

```
## [1] "Am also doing in cbe only. But have to pay."
```

```
as.character(sms_corpus_clean[[3]])
```

```
## [1] " also cbe pay"
```

### Document Term Matrix

This is a process of transformation of tokens. Tokenizing is a process of splitting the message into individual words. In this process, message is termed as number in row and columns contain repeated words.

```
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)

### Alternate process
sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
                                                    tolower=TRUE,
                                                    removeNumbers=TRUE,
                                                    stopwords=TRUE,
                                                    removePunctuation=TRUE,
                                                    stemming = TRUE))

sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5559, terms: 6546)>>
## Non-/sparse entries: 42139/36347075
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
sms_dtm2
```

```
## <<DocumentTermMatrix (documents: 5559, terms: 6946)>>
## Non-/sparse entries: 43211/38569603
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

The difference in sparse entries is due to the variation in the order. It is recommended to follow the first process rather than alternate process.

```
### Creating test and training sets

sms_dtm_train <- sms_dtm[1:4169,]
sms_dtm_test <- sms_dtm[4170:5559,]


sms_train_labels <- sms[1:4169,]$Type
sms_test_labels <- sms[4170:5559, ]$Type

prop.table(table(sms_train_labels))
```

```
## sms_train_labels
##       ham      spam
## 0.8647158 0.1352842
```

```
prop.table(table(sms_test_labels))
```

```
## sms_test_labels
##       ham      spam
## 0.8683453 0.1316547
```

### Word Cloud

This provides the frequently used words. Word Colud is used to determine the trend in social media websites.

```
###install.packages("wordcloud")
###library(wordcloud)
require(wordcloud)
```

```
## Loading required package: wordcloud
```

```
## Warning: package 'wordcloud' was built under R version 3.3.1
```

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

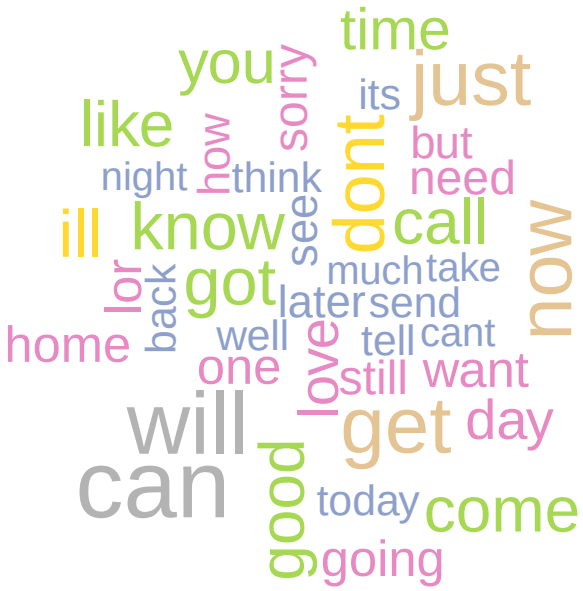
```
pal2 <- brewer.pal(9,"Set1")
pal3 <- brewer.pal(8,"Set2")

wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE,rot.per=.15, colors=pal2)
```


```
spam <- subset(sms,Type=="spam")
ham <- subset(sms,Type=="ham")

wordcloud(spam$Text,max.words = 40, scale = c(3,0.5), rot.per = 0.2, colors=pal3)
```


```
wordcloud(ham$Text,max.words = 40, scale = c(3,0.5), rot.per = 0.2, colors=pal3)
```




**Using frequently used words for training Data structure**

We need to eliminate the words that have appeared in less than 0.1 percent of the records.This can be done through a function called findFreqTerms()

```
sms_freq_words <- findFreqTerms(sms_dtm_train,5)

sms_dtm_freq_train <- sms_dtm_train[, sms_freq_words]

sms_dtm_freq_test <-  sms_dtm_test[,sms_freq_words]


convert_counts<- function(x) {

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


sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
sms_test <- apply(sms_dtm_freq_test, MARGIN=2, convert_counts)

sms_train[50:55]
```

```
## [1] "No" "No" "No" "No" "No" "No"
```

### Step3- Training The Model

```
###install.packages("e1071")
###library(e1071)

require(e1071)
```

```
## Loading required package: e1071
```

```
## Warning: package 'e1071' was built under R version 3.3.1
```

```
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

### Step-4 Evaluating Model Perfmance

```
sms_test_pred <- predict(sms_classifier, sms_test)

length(sms_test_pred)
```

```
## [1] 1390
```

```
length(sms_test_labels)
```

```
## [1] 1390
```

```
###install.packages("gtools")
###library(gtools)
###install.packages("gmodels")
###library(gmodels)
require(gmodels)
```

```
## Loading required package: gmodels
```

```
## Warning: package 'gmodels' was built under R version 3.3.1
```

```
CrossTable(sms_test_pred, sms_test_labels, prop.chisq = FALSE, prop.t=FALSE,
           dnn=c('predicted', 'actual'))
```

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

```
##
## Total Observations in Table:  1390
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1201 |        30 |      1231 |
##              |     0.976 |     0.024 |     0.886 |
##              |     0.995 |     0.164 |           |
## -------------|-----------|-----------|-----------|
##         spam |         6 |       153 |       159 |
##              |     0.038 |     0.962 |     0.114 |
##              |     0.005 |     0.836 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1207 |       183 |      1390 |
##              |     0.868 |     0.132 |           |
## -------------|-----------|-----------|-----------|
##
##
```

**Step5 Improving Model performance**

```
sms_classifier2 <- naiveBayes(sms_train,sms_train_labels, laplace = 2)
sms_test_pred2 <- predict(sms_classifier2, sms_test)
CrossTable(sms_test_pred2, sms_test_labels, prop.chisq = FALSE, prop.t=FALSE,
           dnn=c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1390
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1204 |        34 |      1238 |
##              |     0.973 |     0.027 |     0.891 |
##              |     0.998 |     0.186 |           |
## -------------|-----------|-----------|-----------|
##         spam |         3 |       149 |       152 |
##              |     0.020 |     0.980 |     0.109 |
##              |     0.002 |     0.814 |           |
## -------------|-----------|-----------|-----------|
```

```
## Column Total |      1207 |       183 |      1390 |
##              |     0.868 |     0.132 |           |
## -------------|-----------|-----------|-----------|
##
##
```

**Conclusion:**

We have used the followin procedure and functions:

1)Vcorpus() 2)content_transformer() 3)tolower() 4)tm_map() 5)removeNumbers,removePunctuation 6)stopwords() 7)stemDocument() 8)stripwhitespace() 9)DocumentTermMatrix() 10)Wordcloud 11)subset() 12)naiveBayes() 13)CrossTable

Reference:

Lantz Brett (2015) *Machine Learning with R: Second Edition*