# Exp 5 - Naive Bayes Classifier

February 11, 2022

The **Gaussian Naive Bayes classifier** or Naive Bayes, is a technique that simplifies predictive modelling tasks and avoids the dimensionality curse. It frequently yields very accurate and stable models with relatively small sample sets. The fundamental assumption of Nave Bayes is that **all independent variable characteristics are conditionally independent**.

## 1 Experimental Description

### 1.1 Objective

To perform Naive Bayesian classifier on a set of documents that needs to be classified. Find the conditional probability of attributes of the training data using Bayes Theorem by following the steps of the algorithm as mentioned in the experiment. Compute accuracy, precision, and recall of the generated model using the test dataset.

### 1.2 Algorithm

1. Convert the given dataset into a frequency table.
2. Create a likelihood table by finding the probabilities.
3. Calculate the posterior probability of each feature with respect to the class.
4. If for a certain feature the probability evaluates to zero use feature smoothing for correction.
5. Classify the example into the class for which the probability is highest.

### 1.3 Procedure

In Naive Bayes for document classification, we extract numerical features from text content by:

- Tokenizing strings and giving an integer ID for each possible token.
- Counting the occurrences of tokens in each document.
- Treating each token occurrence frequency as a feature.
- Considering the vector of all the token frequencies (for a given document) as a multivariate sample.

### 1.4 System Requirements

Windows/Linux OS/Mac OS with R. Required packages are **tm**, **Snowball**, **wordcloud**, **e1071** and **caret**.

### 1.5 Dataset Summary

For this project, we obtained structured data of SMS messages in a CSV format having 2 variables, namely, 'category' and 'message'. The dataset consists of 87% ham messages and 13% spam messages. **message** contains the SMS text to be classified, and **category** contains information on message type (spam or ham). There are 5572 rows in the original dataset.

## 2 Code and Output

```
rm(list = ls())
version$version.string
```

```
## [1] "R version 4.2.0 (2022-04-22 ucrt)"
```

```
# Importing data
sms_raw <- read.csv("spam.csv", header = FALSE, stringsAsFactors = FALSE)

# Manipulating data
sms_raw <- sms_raw[,1:2]
colnames(sms_raw) <- c("Type", "Text")
sms_raw$Type <- factor(sms_raw$Type)
```

```
# Checking the structure of the dataset
str(sms_raw)
```

```
## 'data.frame':    5573 obs. of  2 variables:
##  $ Type: Factor w/ 3 levels "Category","ham",..: 1 2 2 3 2 2 3 2 2 3 ...
##  $ Text: chr  "Message" "Go until jurong point, crazy.. Available only in bugis n great
```

```
# Checking the number of "Spam" and "Ham" messages
table(sms_raw$Type)
```

```
##
## Category      ham     spam
##        1     4825      747
```

```
prop.table(table(sms_raw$Type))
```

```
##
##     Category          ham         spam
## 0.0001794366 0.8657814463 0.1340391172
```

```
# Install the "tm" package by uncommenting and running the following command
# install.packages("tm")

# Loading the "tm" package
library(tm)
```

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

```
# Creating a volatile corpus which is a collection of texts
sms_corpus <- VCorpus(x = VectorSource(sms_raw$Text))

# Printing corpus
sms_corpus
```

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

```
## Content:  documents: 5573
```

```
# Checking the text in some messages and their associated label
sms_corpus[5:8]
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 4
```

```
sms_raw$Type[5:8]
```

```
## [1] ham  ham  spam ham
## Levels: Category ham spam
```

```
# Creating a copy of the corpus
corpus_clean <- sms_corpus

# Removing numbers
corpus_clean <- tm_map(x = corpus_clean, FUN = removeNumbers)

# Removing punctuation
corpus_clean <- tm_map(x = corpus_clean, FUN = removePunctuation)
```

```
# Printing stop words
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"
```

```
# Removing stop words from the corpus
corpus_clean <- tm_map(x = corpus_clean, FUN = removeWords, stopwords())
```

```
# Install the "SnowballC" package by uncommenting and running the following command
# install.packages("SnowballC")

# Loading package
library(SnowballC)

# Testing the package
wordStem(words = c("cooks", "cooking", "cooked"))
```

```
## [1] "cook" "cook" "cook"
```

```
# Stemming words in corpus
corpus_clean <- tm_map(x = corpus_clean, FUN = stemDocument)
```

```
# Removing extra white spaces
corpus_clean <- tm_map(x = corpus_clean, FUN = stripWhitespace)
```

```
# Creating Document Term Matrix (DTM)
DTM <- DocumentTermMatrix(x = corpus_clean)

# Viewing the generated matrix
DTM
```

```
## <<DocumentTermMatrix (documents: 5573, terms: 7241)>>
## Non-/sparse entries: 46148/40307945
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
# Install the "wordcloud" package by uncommenting and running the following command
# install.packages("wordcloud")

# Loading package
library(wordcloud)

# Creating word cloud for the whole dataset
wordcloud(words = corpus_clean,
          min.freq = 100, # Minimum number of times a word must be present
          random.order = FALSE, # Arrange most frequent words in the center
          color = (colors = c("#4575b4","#74add1","#abd9e9","#e0f3f8","#fee090","#fdae61",
          "#f46d43","#d73027")))
```


```
# Creating training dataset
DTM_train <- DTM[1:round(nrow(DTM)*0.80, 0), ]

# Creating test dataset
DTM_test <- DTM[(round(nrow(DTM)*0.80, 0)+1):nrow(DTM), ]

# Creating vectors with labels for the training and test datasets
train_labels <- sms_raw[1:round(nrow(sms_raw)*0.80, 0), ]$Type
test_labels <- sms_raw[(round(nrow(sms_raw)*0.80, 0)+1):nrow(DTM), ]$Type

# Checking the proportion of "ham" and "spam" in the training and test datasets
prop.table(table(train_labels))
```

```
## train_labels
##     Category          ham         spam
## 0.0002243158 0.8647375505 0.1350381337
```

```
prop.table(table(test_labels))
```

```
## test_labels
##  Category       ham      spam
## 0.0000000 0.8699552 0.1300448
```

```
# Sub-setting most frequent words
threshold <- 0.1 # In percentage
min_freq = round(DTM$nrow*(threshold/100),0) # Calculate minimum frequency
min_freq
```

```
## [1] 6
```

```
# Creating vector of most frequent words
frequent_words <- findFreqTerms(x = DTM, lowfreq = min_freq)

# Checking the structure of the generated vector
str(frequent_words)
```

```
##  chr [1:1293] "abiola" "abl" "about" "abt" "accept" "access" "accident" ...
```

```
# Filtering both training and test DTMs
DTM_train_most_frequent <- DTM_train[, frequent_words]
DTM_test_most_frequent <- DTM_test[, frequent_words]

# Checking dimension of both training and test DTM
dim(DTM_train_most_frequent)
```

```
## [1] 4458 1293
```

```
dim(DTM_test_most_frequent)
```

```
## [1] 1115 1293
```

```
# Creating a function to output "Yes" if a word is present and "No" if it is absent in the
document
is_present <- function(x) {
  x <- ifelse(test = x > 0, yes = "Yes", no = "No")
}

# Testing the function
x <- is_present(c(1, 0, 3, 4, 0, 0))
x
```

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

```
# Applying is_present() function to training and test DTMs
DTM_train_most_frequent <- apply(X = DTM_train_most_frequent,
MARGIN = 2, # Apply function to columns
FUN = is_present) # Specify function to be used
DTM_test_most_frequent <- apply(X = DTM_test_most_frequent,
MARGIN = 2, # Apply function to columns
FUN = is_present) # Specify function to be used
```

```
# Install the "e1071" package by uncommenting and running the following command
# install.packages("e1071")

# Loading package
library(e1071)

# Creating model using the training dataset
spam_classifier <- naiveBayes(x = DTM_train_most_frequent, y = train_labels)

# Printing probability tables for some words
spam_classifier$tables$call
```

```
##             call
## train_labels         No        Yes
##     Category 1.00000000 0.00000000
```

```
##     ham      0.94422827 0.05577173
##     spam     0.57308970 0.42691030
```

```
spam_classifier$tables$friend
```

```
##             friend
## train_labels         No        Yes
##     Category 1.00000000 0.00000000
##     ham      0.98184176 0.01815824
##     spam     0.98172757 0.01827243
```

```
spam_classifier$tables$free
```

```
##             free
## train_labels         No        Yes
##     Category 1.00000000 0.00000000
##     ham      0.98806744 0.01193256
##     spam     0.77574751 0.22425249
```

```
# Making predictions on test dataset
test_predictions <- predict(object = spam_classifier, newdata = DTM_test_most_frequent)

# Install the "caret" package by uncommenting and running the following command
# install.packages("caret")

# Loading caret package
library(caret)

# Creating confusion matrix
confusionMatrix(data = test_predictions, reference = test_labels, positive = "spam", dnn =
c("Prediction", "Actual"))
```

```
## Confusion Matrix and Statistics
##
##           Actual
## Prediction Category ham spam
##   Category        0   0    0
##   ham             0 964   13
##   spam            0   6  132
##
## Overall Statistics
##
##                Accuracy : 0.983
##                  95% CI : (0.9735, 0.9897)
##     No Information Rate : 0.87
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9231
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Category Class: ham Class: spam
## Sensitivity                       NA     0.9938      0.9103
## Specificity                        1     0.9103      0.9938
```

```
## Pos Pred Value                    NA     0.9867      0.9565
## Neg Pred Value                    NA     0.9565      0.9867
## Prevalence                         0     0.8700      0.1300
## Detection Rate                     0     0.8646      0.1184
## Detection Prevalence               0     0.8762      0.1238
## Balanced Accuracy                 NA     0.9521      0.9521
```

```
# Calculating precision and recall
temp = as.matrix(table(test_labels, test_predictions))
precision <- diag(temp)/colSums(temp)
precision
```

```
##  Category       ham      spam
##       NaN 0.9866940 0.9565217
```

```
recall <- diag(temp)/rowSums(temp)
recall
```

```
##  Category       ham      spam
##       NaN 0.9938144 0.9103448
```

From the output, we can interpret the following:

- The algorithm constructed tables of probabilities that were used to estimate the likelihood of new examples belonging to various classes and gave an accuracy of 98%.
- The model has a precision of 95.6% for spam messages and 98.6% for ham messages.
- The model has a recall of 91.0% for spam messages and 99.3% for ham messages.

# 3 Conclusion

**Naive Bayes Classifier** got implemented successfully over the given dataset.