***Naïve Bayes Algorithm***

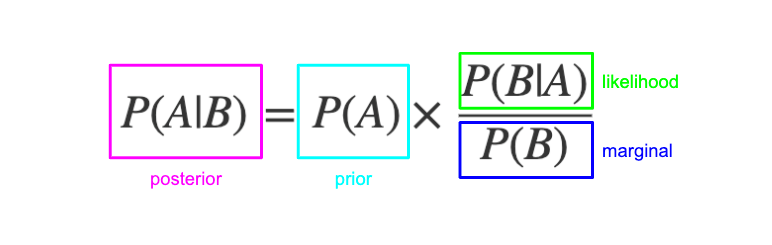
**Aim:** Spamming electronic communication channels with fraud messages is wasting people/organization time and money. It would be helpful if there is a filter that separates spam and ham messages to customer as they pay for SMS services. In this project, we are focusing on SMS to set filter for spam and ham using Naïve Bayes Algorithm. This prevents customer missing important messages in ocean of spams.

**Algorithm selected**: This problem is text classification type. Naïve Bayes works well as it is simple, fast, and very effective way to classify texts. It treats message words as independent which depend on the spam or ham classification. Though in reality words may not be independent, this algorithm performs well due to its versatility and accuracy.

This uses Bayesian probabilistic methods for classification. This uses observed probabilities to predict the most likely class for the new features. It estimates probability of events and revises based on their prior probabilities. For example, let’s take a vegetable carrot which is orange in color and 10cms long. Even if these features dependent on each other or other features, all these properties independently contribute to the probability that this vegetable is carrot and that is why it is known as ‘Naive’ Bayes.

Bayesian classification works well when there is a situation to consider large number of features simultaneously to estimate overall probability. Many ML classification algorithms like KNN ignore weak features and some missed instances, whereas Bayesian considers all the features to predict even the minute effects on outcome. It also adds small numbers to each feature to ensure nonzero probability of occurrence.

In this project, spam and ham are mutually exclusive and exhaustive events, which implies that they cannot occur at the same time and are the only two possible outcomes. Bayes Theorem is given by following formula.



Let’s dive in into the steps now

1. **DATA COLLECTION**: Data used in this project is SMS Spam Collection from <http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>.

This data set has 5559 instances and 2 features indicating SMS text messages and classification labels. All the messages are labelled as either spam or ham against each message.

1. **DATA PREPARATION**:

We will start importing the csv data in R, for this

* sms\_raw=read.csv("sms\_spam.csv", stringsAsFactors = FALSE)

#To see data structure

* str(sms\_raw)

*'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 doing in cbe only. But have to pay." "complimentary 4 STAR Ibiza Holiday or Â£10,000 cash needs your URGENT collection. 09066364349 NOW from Landline"| \_\_truncated\_\_ ...*

This is the structure of the data. The type feature is categorial variable now. As there are only 2 types, it’s better to convert into factor

#Change type feature to factors (ham and spam)

* sms\_raw$type=factor(sms\_raw$type)
* str(sms\_raw$type)

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

* table(sms\_raw$type)

*ham spam*

*4812 747*

1. **Data Preparation/Cleaning:** Cleaning of complex data needs many steps to convert it into data to be used for statistical processing. Text mining (tm) package is available in R for exclusively working on complex texts. Following are steps for processing text

**i) Creating Corpus:** Corpus is collection of text documents (here it is SMS texts). We use Vcorpus() for volatile corpus and Pcorpus() for corpus on permanent memory.

**ii)Covert all text to common case:** To avoid treating same words differently (call & Call), we must standardize messages by bringing them to common case. Here we are trying to convert it to lowercase

**iii)Remove all numbers in text:** Although some numbers may provide useful information, the majority would likely be unique to individual senders and thus will not provide useful patterns across all messages. So, removenumbers() is a bult in function to remove numbers.

**iv)Removing most frequent words-StopWords:** The commonly used English words like “a”,” is”,” the” in the tm package are referred to as stop words. These words must be eliminated to get the results more accurate. R has inbuilt function to remove stop words-stopwords().

**v)Remove Punctuation:** Punctuations aren’t helpful for ML algorithms. So, we cease punctuation by removepunctuation() function in tm package

**vi)Stemming:** Stemming is the process of reducing words of similar origin into one word for example “dial”, “dials”, “dialed”. Stemming helps us increase accuracy in our mined text by removing suffixes and reducing words to their basic forms. We shall use the wordstem() in SnowballC library for this.

**vii)Remove white spacing:** After removing numbers, stop words, and punctuation as well as performing stemming, the text messages are left with the blank spaces that previously separated the now-missing pieces. For this, stripwhitespace in tm package helps.

**viii)Tokenization:** After the cleaning process, we are left with independent terms that exist throughout the document. These are stored in a matrix that shows each of their occurrence. This matrix logs the number of times the term appears in our clean data set, so it is called as document term matrix in which rows indicate documents (SMS messages) and columns indicate terms (words).

#TEXT PROCESSING

* sms\_corpus=VCorpus(VectorSource(sms\_raw$text))

#Summary of texts

* inspect(sms\_corpus[1:2])

*Metadata: corpus specific: 0, document level (indexed): 0*

*Content: documents: 2*

*[[1]]*

*<<PlainTextDocument>>*

*Metadata: 7*

*Content: chars: 49*

*[[2]]*

*<<PlainTextDocument>>*

*Metadata: 7*

*Content: chars: 23*

* as.character(sms\_corpus[[1]])

*[1] "Hope you are having a good week. Just checking in"*

* lapply(sms\_corpus[1:2], as.character)

*$`1`*

*[1] "Hope you are having a good week. Just checking in"*

*$`2`*

*[1] "K..give back my thanks."*

#Cleaning the corpus by transforming text to lowercase

* sms\_corpus\_clean=tm\_map(sms\_corpus,content\_transformer(tolower))
* as.character(sms\_corpus[[1]])
* as.character(sms\_corpus\_clean[[1]])

*[1] "hope you are having a good week. just checking in"*

* library(tm)

#Remove all numbers from corpus

* sms\_corpus\_clean=tm\_map(sms\_corpus\_clean, removeNumbers)

#Remove stopwords

* sms\_corpus\_clean=tm\_map(sms\_corpus\_clean,removeWords, stopwords())

#Remove punctuation

* sms\_corpus\_clean=tm\_map(sms\_corpus\_clean, removePunctuation)

#Stemming

* sms\_corpus\_clean=tm\_map(sms\_corpus\_clean, stemDocument)

#Strip whitespace after all text cleaning

* sms\_corpus\_clean=tm\_map(sms\_corpus\_clean, stripWhitespace)

# Text docs to words-tokenization

* sms\_dtm=DocumentTermMatrix(sms\_corpus\_clean)

1. **TRAINING & TEST DATA**: Dividing the data into training and test data to predict the model accuracy. For this, we are dividing 75%(4169) instances as training data and 25% (1390)data as test data. As the data here is randomly ordered we are directly dividing them, but when they are not randomly ordered its better to do resampling techniques.

* sms\_dtm\_train=sms\_dtm[1:4169, ]
* sms\_dtm\_test=sms\_dtm[4170:5559, ]

For training model, We have to include labels. We will use these in the next steps of training and evaluating our classifier

* sms\_train\_labels=sms\_raw[1:4169, ]$type
* sms\_test\_labels=sms\_raw[4170:5559, ]$type
* prop.table(table(sms\_train\_labels))

*sms\_train\_labels*

*ham spam*

training and test data contain about 13 % spam. This means that the spam messages were divided evenly between the two datasets.

*0.8647158 0.1352842*

* prop.table(table(sms\_test\_labels))

*sms\_test\_labels*

*ham spam*

*0.8683453 0.1316547*

**DATA VISUALIZATION:** A word cloud is a way to visually represent the frequency at which words appear in text data. The cloud is composed of words scattered somewhat randomly around the figure. Words appearing more often in the text are shown in a larger font, while fewer common terms are shown in smaller fonts. This is done by wordcloud() in wordcloud package.

* library(wordcloud)
* wordcloud(sms\_corpus\_clean, min.freq = 50, random.order = FALSE)
* spam=subset(sms\_raw, type == "spam")
* ham= subset(sms\_raw, type == "ham")
* wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
* wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))

Text

Description automatically generated

Word Cloud for raw data

Text

Description automatically generatedText, letter

Description automatically generated

Word Cloud for cleaned spam messages

Word Cloud for cleaned ham messages

Frequent words like free, call, reply is in spam. Naive Bayes model will use these strong key words to differentiate between the spam/ham classes

There are many features that may not be as useful in ML algorithm. So, to reduce the complexity, we will eliminate any word that appear in less than 5 SMS messages, or in less than about 0.1 percent of the records in the training data. Naive Bayes classifier is typically trained on data with categorical features. This raises issue as the cells in the DTM 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.

* findFreqTerms(sms\_dtm\_train, 5)
* sms\_freq\_words=findFreqTerms(sms\_dtm\_train, 5)
* str(sms\_freq\_words)
* 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)

**5.TRAINING THE MODEL**: Unlike KNN, Naïve Bayes actually trains the model. We use ‘e1071’ package in R which provides naïve bayes function. naiveBayes() function in e1071 package helps in implementation of algorithm.

* sms\_classifier=naiveBayes(sms\_train, sms\_train\_labels)
* sms\_test\_pred=predict(sms\_classifier, sms\_test)

**6.EVALUATING THE MODEL:** Cross table() function in gmodels package helps in evaluating the NaiveBayes. The output will be the confusion matrix indicating the prediction accuracies and anomalies. Executing this will give results as below.

* library(gmodels)
* CrossTable(sms\_test\_pred, sms\_test\_labels,prop.chisq = FALSE, prop.t = FALSE,dnn = c('predicted', 'actual'))

Table

Description automatically generated with medium confidence

Accuracy rate of this model is 97.4%. We must decide which errors are dangerous based on the problem we have. In this case, there are 6 False Positive. These are costly in this problem as customer tend to miss some important information.

There are 30 False Negative. The predicted value is ham where the actual it is spam. This doesn’t really is so serious, but it hampers the accuracy of the model. But when compared to FN it is not as dangerous as FP was.

We didn’t add Laplace estimator in our model now adding it to improve the performance of model

**6.IMPROVING THE MODEL**: Let’s explore on improvisations of the model. Our aim is to reduce the costly False Positive

Applying smoothing technique in case of zero frequency issue, adding Laplace=1 and monitor the results

* sms\_classifier2=naiveBayes(sms\_train, sms\_train\_labels,laplace = 1)
* sms\_test\_pred2=predict(sms\_classifier2, sms\_test)
* CrossTable(sms\_test\_pred2, sms\_test\_labels,prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,dnn = c('predicted', 'actual'))

Table

Description automatically generated

Accuracy rate of this model is 97.6% False Positive reduced by 1 and false negative from 30 to 28. But we can’t increase Laplace estimator because it makes model more aggressive. Overall, model showed improvement than base model.

* Instead of taking independent feature words trying with feature word pairs, combinations may yield better accuracy of the model. (I am not sure of how to control the words pairs in this model)

**Conclusion**: Spam filtration model was built using Naïve Bayes and achieved an accuracy on 97.6%. With improvement false positive decreased by 1 count by adding smoothening factor (Laplace=1). Here false positives are costly as they may lead to customer information loosing. Tweaking the model so much may lead to overfitting of the model.

**Limitations of Naïve Bayes:**

* [Naive Bayes](https://courses.analyticsvidhya.com/courses/naive-bayes?utm_source=blog&utm_medium=naive-bayes-explained) is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.
* For improving the model, some classifier combination techniques like bagging and boosting methods would not help as their purpose is to reduce variance. Naive Bayes has no variance to minimize.