Machine Leaning and Data Mining Assignment Project Report

PAU

PAN AFRICAN UNIVERSITY

INSTITUTE FOR BASIC SCIENCES,

TECHNOLOGY AND INNOVATION

JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

TECHNOLOGY FOR DEVELOPMENT

Name: Haruna Jallow

Subject: Machine Leaning and Data Mining

Major: MSc Data Science

Lecturer: Prof Waititu

30th October 2023

Contents

1 Abstract

In this report, I did a comprehensive literature review of KNN and Naive Bayes algorithms and also delve into the analysis and classification of SMS messages, with a focus on differentiating between spam and ham messages. The primary objective of this study is to build and evaluate a Naïve Bayes classification model on a dataset of SMS messages. The report encompasses detailed explanations of the two algorithms and a comprehensive exploratory data analysis, model development, and performance evaluation. Descriptive statistics, including word clouds, pie charts, frequency tables, and bar graphs, offer insight into the characteristics of the dataset. The report also addresses the determination of the optimal Laplace smoothing parameter, overall accuracy, Kappa, Mcnemar's Test P-Value, sensitivity, specificity, and detection rate. These findings contribute to a better understanding of the effectiveness of the Naïve Bayes classifier in distinguishing between spam and ham messages in the SMS dataset.

2 Introduction

The report reports a comprehensive literature review of two machine learning algorithms; KNN and Naive Bayes. This report also aims to classify SMS messages as spam or ham using a Naïve Bayes model. Spam messages are unsolicited and may pose security and privacy risks, while ham messages are legitimate and desired. Machine learning techniques, such as text classification, can help detect spam messages effectively.
The report consists of two parts:
The first part consist of the literature review of two machine learning algorithms in detailed. The second part is also divided into two parts; descriptive statistics and model evaluation. The dataset contains labelled SMS messages, which are analyzed and visualized using word clouds, pie charts, frequency tables, and bar graphs. These methods reveal the characteristics and distribution of spam and ham messages.
The report then develops and evaluates a Naïve Bayes model, which is a probabilistic classifier that uses the frequency of words to predict the class of a message. The report explores the optimal Laplace smoothing parameter, which is a technique to avoid zero probabilities in the model. The report also assesses the model's performance using metrics such as accuracy, Kappa, Mcnemar's Test P-Value, sensitivity, specificity, and detection rate.
The report provides valuable insights into the classification of SMS data, demonstrating the performance and suitability of the Naïve Bayes model for this task. The report also discusses the challenges and limitations of such models in spam detection.

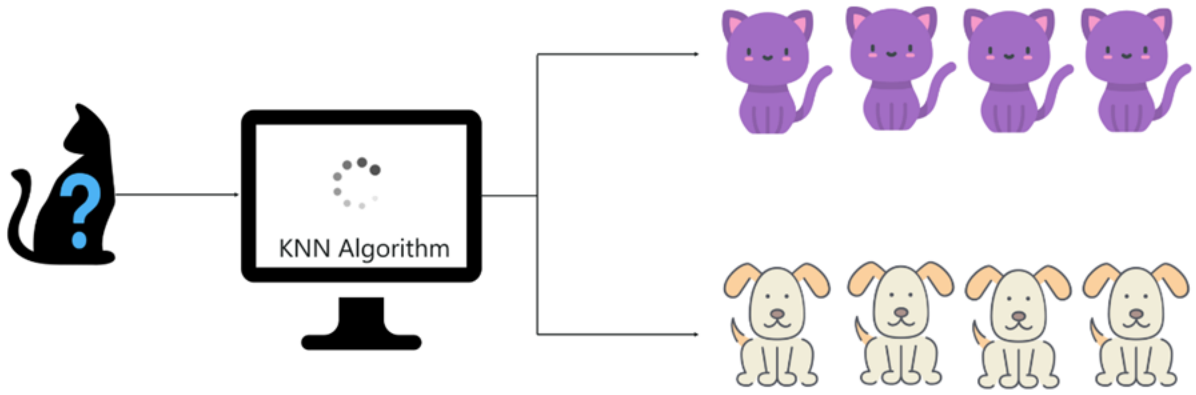
3 Literature Review

3.1 K-Nearest Neighbors (K-NN)

k-NN is short for k-Nearest Neighbor classifier, which is a type of 'lazy learners' method. It was first proposed in the 1950s and gained popularity in the 1960s [3]. kNN is one of the most widely used data-mining algorithms because it is simple and efficient. It can be applied to various data-mining tasks, such as classification, regression, and missing value imputation [4], [3].

This method measures the similarity of the training records that are closest to it. The value k indicates how many neighbors are used for comparison. The value of k determines how many k nearest data are involved in the comparison [4], [3].

We can use a simple example to explain the KNN algorithm. Suppose we want a machine to tell apart images of cats and dogs. We need to give the machine a dataset of cat and dog images and train it to recognize the animals based on some features. For instance, features like pointy ears can help the machine identify cats and likewise, we can distinguish dogs based on their long ears [2].

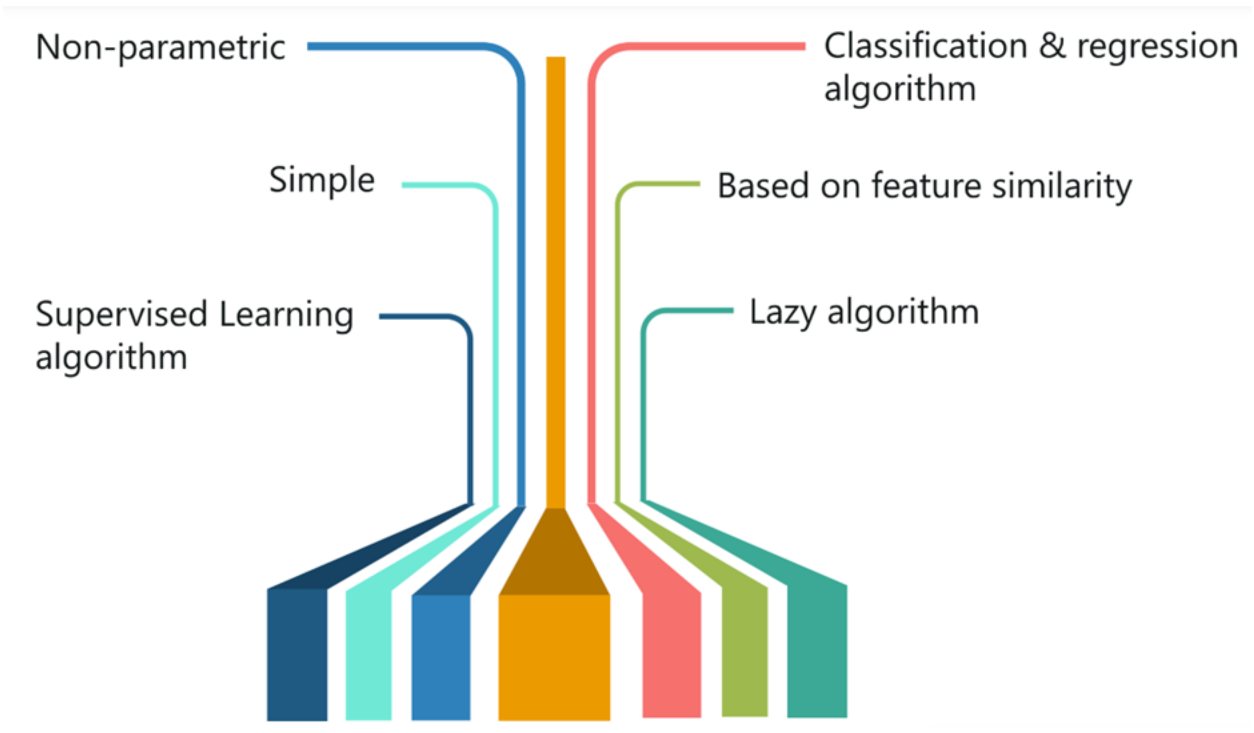


The KNN algorithm learns from the dataset in the training phase, and then it assigns a label to a new image based on how similar it is to the features of the existing images. So, if the new image has pointy ears, it will label it as a cat because it matches the cat images. This is how the KNN algorithm groups data points according to their similarity to their neighboring data points.
Now let's discuss the features of the KNN algorithm.

3.1.1 Features Of KNN Algorithm

The KNN algorithm has the following features:

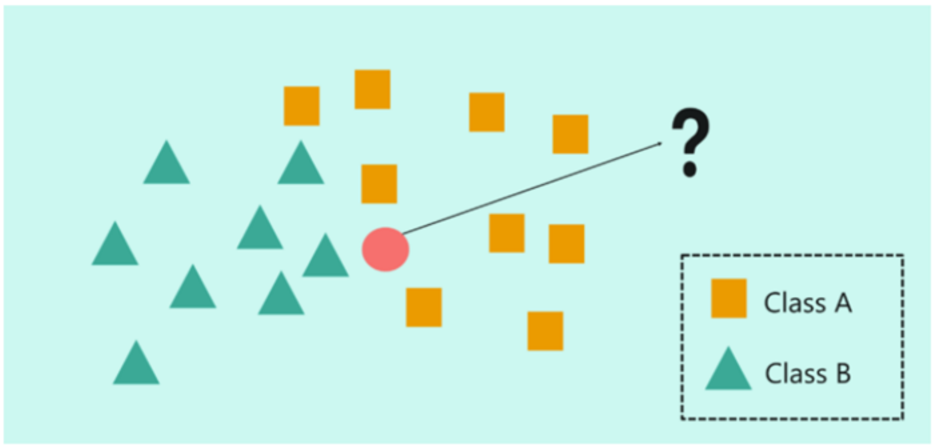
- KNN is a Supervised Learning algorithm that uses a labeled input data set to predict the output of the data points.
- It is one of the simplest Machine learning algorithms and can be easily implemented for various problems.
- It is mainly based on feature similarity. KNN checks how similar a data point is to its neighbor and classifies the data point into the class it is most like.


Non-parametric
Simple
Supervised Learning algorithm
Classification & regression algorithm
Based on feature similarity
Lazy algorithm

- Unlike most algorithms, KNN is a non-parametric model, which means it does not make any assumptions about the data set. This makes the algorithm more effective since it can handle realistic data.
- KNN is a lazy algorithm, this means that it memorizes the training data set instead of learning a discriminative function from the training data.
- KNN can be used for solving both classification and regression problems.

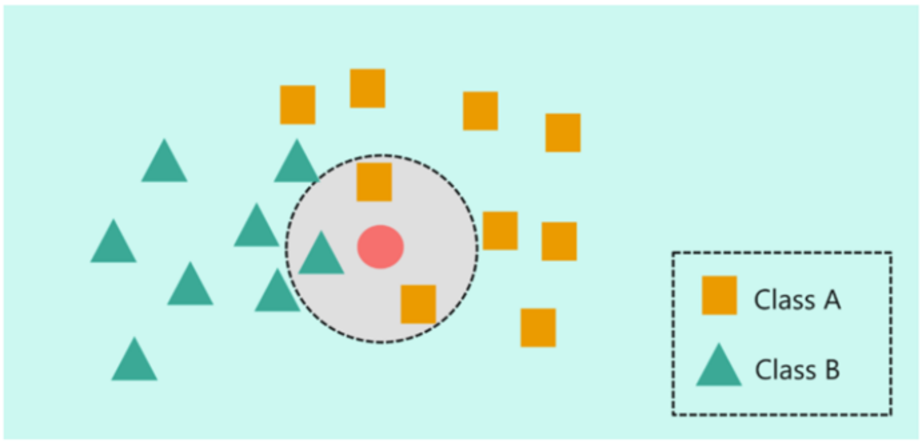
3.1.2 KNN Algorithm Example

To understand how KNN algorithm works, let's consider the following scenario:

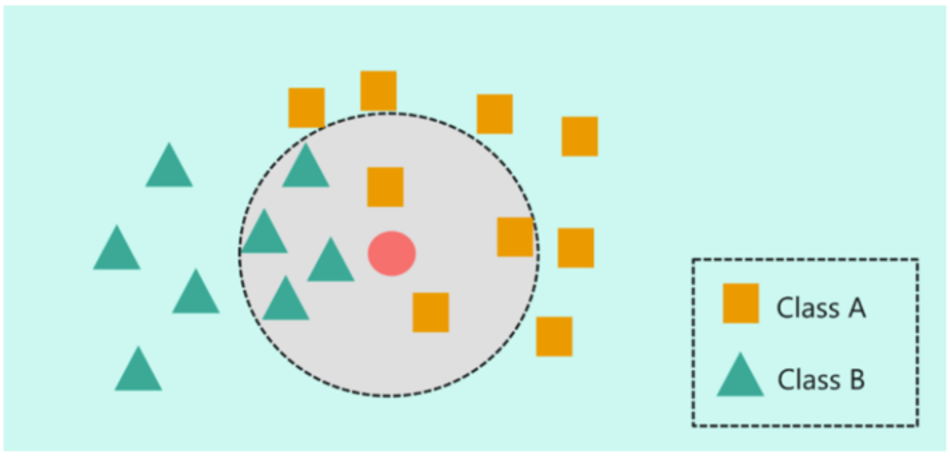


- In the above image, we have two classes of data, namely Class A (squares) and Class B (triangles)

- The problem statement is to assign the new input data point to one of the two classes by using the KNN algorithm.
- The first step in the KNN algorithm is to define the value of ‘K’. But what does the ‘K’ in the KNN algorithm stand for?
- ‘K’ stands for the number of Nearest Neighbors hence the name K Nearest Neighbors (KNN).

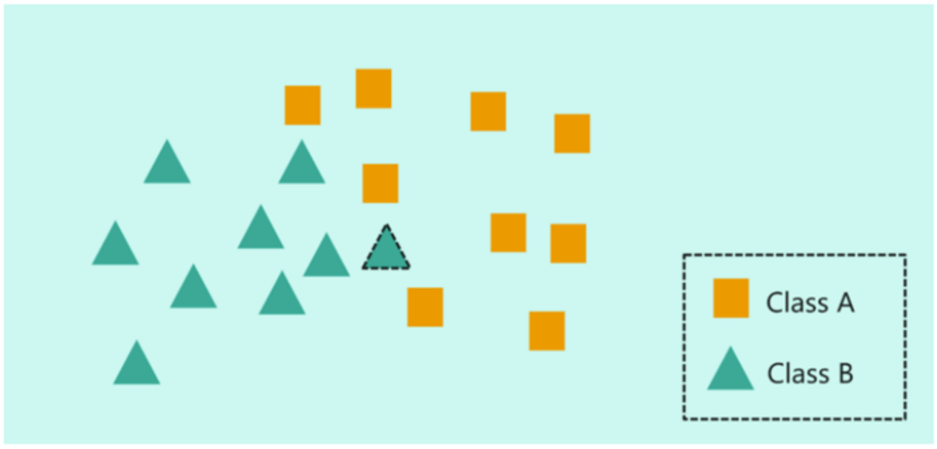


- In the above image, I’ve defined the value of ‘K’ as 3. This means that the algorithm will consider the three neighbors closest to the new data point to decide the class of this new data point.
- The closeness between the data points is calculated by using measures such as Euclidean and Manhattan distance, which I’ll explain below.
- At ‘K’ = 3, the neighbors include two squares and 1 triangle. So, if I were to classify the new data point based on ‘K’ = 3, then it would be assigned to Class A (squares).

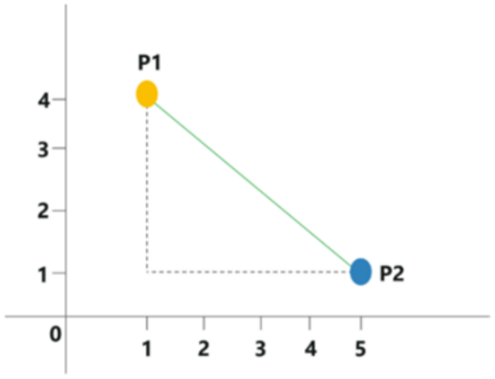


- But what if the ‘K’ value is set to 7? Here, I’m basically telling my algorithm to look for the seven nearest neighbors and classify the new data point into the class it is most like.

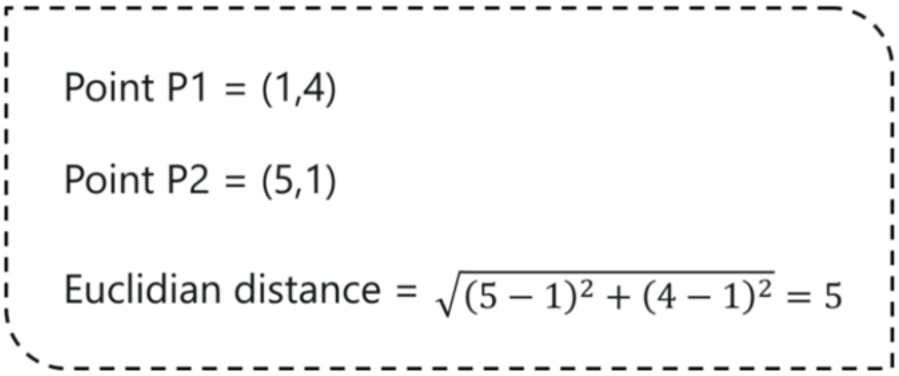
- At 'K' = 7, the neighbors include three squares and four triangles. So, if I were to classify the new data point based on 'K' = 7, then it would be assigned to Class B (triangles) since many of its neighbors were of Class B.



Earlier I mentioned that KNN uses Euclidean distance as a measure to check the distance between a new data point and its neighbors, let's see how.



- Consider the above image, here we're going to measure the distance between P_1 and P_2 by using the Euclidian Distance measure.
- The coordinates for P_1 and P_2 are $(1, 4)$ and $(5, 1)$ respectively.
- The Euclidian Distance can be calculated like so:



Point P1 = (1,4)

Point P2 = (5,1)

Euclidian distance = $\sqrt{(5-1)^2 + (4-1)^2} = 5$

It is as simple as that! KNN makes use of simple measures to solve complex problems, this is one of the reasons why KNN is such a commonly used algorithm.
To sum it up, let's look at the pseudocode for KNN Algorithm.

3.1.3 KNN Algorithm Pseudocode

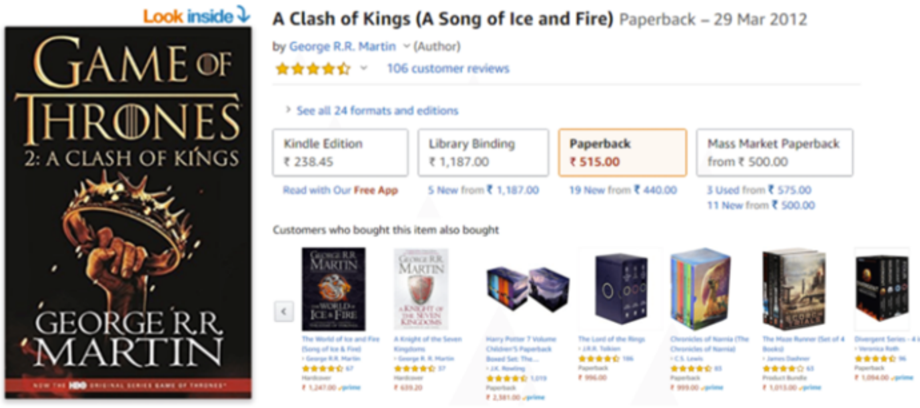
Consider the set, (X_i, C_i),

- Where X_i denotes feature variables and 'i' are data points ranging from $i = 1, 2, \ldots.., n$
- C_i denotes the output class for X_i for each i

The condition, $C_i \in 1, 2, 3, \ldots\ldots, c$ is acceptable for all values of 'i' by assuming that the total number of classes is denoted by 'c'.
Now let's pretend that there's a data point 'x' whose output class needs to be predicted. This can be done by using the K-Nearest Neighbour (KNN) Algorithm.

3.1.4 KNN Algorithm Use-Case

You probably have used Amazon to shop online! Have you ever noticed that when you buy something, Amazon shows you a list of suggestions based on what you bought? Also, Amazon has a section that says, 'Customers who bought this item also bought this...'
Machine learning is very important for Amazon's suggestion system. The idea behind a suggestion engine is to offer products to customers based on what other customers with similar shopping habits bought.


Look inside
GAME OF THRONES
2: A CLASH OF KINGS
GEORGE R.R. MARTIN
A Clash of Kings (A Song of Ice and Fire) Paperback – 29 Mar 2012
by George R.R. Martin (Author)
106 customer reviews
See all 24 formats and editions
Kindle Edition ₹ 238.45
Library Binding ₹ 1,187.00
Paperback ₹ 515.00
Mass Market Paperback from ₹ 500.00
Read with Our Free App
5 New from ₹ 1,187.00
19 New from ₹ 440.00
3 Used from ₹ 575.00
11 New from ₹ 500.00
Customers who bought this item also bought

For example, suppose that customer A who likes mystery novels bought the Game Of Thrones and Lord Of The Rings book series. A few weeks later, another customer B who reads the same kind of books buys Lord Of The Rings. He does not buy the Game of Thrones book series, but Amazon recommends it to him because his shopping habits and his choice of books are very similar to customer A.
So, Amazon suggests products to customers based on how similar their shopping habits are. This similarity can be calculated by using the KNN algorithm which mainly depends on feature similarity.

3.2 Naive Bayes

Naïve Bayes classification is a classifier that uses probability based on the Bayes Theorem. It calculates the probability of a class for each sample. It assumes that all attributes are not related to each other [3].

But why is Naive Bayes called 'Naive'?

The variables that we use to make predictions in real-world problems are not always unrelated to each other, they often have some connections between them. Naive Bayes assumes that each variable in the model does not depend on any other variable, so it is called 'Naive' [5]. However, this is a strong assumption, which is clearly violated in most practical applications and is therefore naive – hence the. That is changing the value of one feature, does not directly influence or change the value of any of the other features used in the algorithm name [1].

3.2.1 The Math Behind Naive Bayes

Naive Bayes is based on the Bayes theorem or the Bayes Rule, which is a way of finding the conditional probability, which means how likely something is to happen given some information about what happened before. The Bayes theorem can be written like this in math [5].
In the above equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- $P(A|B)$: Conditional probability of event A occurring, given the event B
- $P(A)$: Probability of event A occurring
- P(B): Probability of event B occurring
- $P(B|A)$: Conditional probability of event B occurring, given the event A

Formally, the terminologies of the Bayesian Theorem are as follows:

- A is known as the proposition and B is the evidence
- P(A) represents the prior probability of the proposition
- P(B) represents the prior probability of evidence
- $P(A|B)$ is called the posterior
- $P(B|A)$ is the likelihood

Therefore, the Bayes theorem can be summed up as:

$$\frac{\text{Likelihood} \times \text{Proposition prior probability}}{\text{Evidence prior probability}}$$

It can also be considered in the following manner:

Given a Hypothesis H and evidence E, Bayes Theorem states that the relationship between the probability of the Hypothesis before getting the evidence P(H) and the probability of the hypothesis after getting the evidence $P(H|E)$ is:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

Now that you know what the Bayes Theorem is, let's see how it can be derived.

3.2.2 Derivation Of The Bayes Theorem

The main aim of the Bayes Theorem is to calculate the conditional probability. The Bayes Rule can be derived from the following two equations:
The below equation represents the conditional probability of A, given B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

The below equation represents the conditional probability of B, given A:

$$P(B|A) = \frac{P(\mathrm{B} \cap A)}{P(A)}$$

Therefore, on combining the above two equations we get the Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

3.2.3 Bayes Theorem for Naive Bayes Algorithm

The above equation was for a single predictor variable, however, in real-world applications, there is more than one predictor variable and for a classification problem, there is more than one output class. The classes can be represented as, $C_1, C_2, \ldots, C_k$, and the predictor variables can be represented as a vector, $x_1, x_2, \ldots, x_n$.

The objective of a Naive Bayes algorithm is to measure the conditional probability of an event with a feature vector $x_1, x_2, \ldots, x_n$ belonging to a particular class C_i,

$$P(C_i|\, x_1, x_2 \ldots, x_n) = \frac{P(x_1, x_2 \ldots, x_n|C_i).P(C_i)}{P(x_1, x_2 \ldots, x_n)}\ for\ 1 < i < k$$

On computing the above equation, we get:

$$\begin{aligned}
&P(x_1, x_2 \ldots, x_n|C_i).P(C_i) = P(x_1, x_2 \ldots, x_n, C_i)\\
&P(x_1 x_2 \ldots x_n C_i) = P(x_1 \mid x_2, \ldots, x_n, C_i).P(x_2, \ldots, x_n, C_i)\\
&\qquad = P(x_1 \mid x_2, \ldots, x_n, C_i).P(x_2|x_3, \ldots, x_n, C_i)\, P(x_3, \ldots, x_n, C_i)\\
&\qquad = \ldots\\
&\qquad = P(x_1 \mid x_2, \ldots, x_n, C_i).P(x_2|x_3, \ldots, x_n, C_i) \ldots P(x_{n-1}|x_n, C_i).P(x_n|C_i).P(C_i)
\end{aligned}$$

However, the conditional probability, i.e., $P(x_j|x_j+1, \ldots, x_n, C_i)$ sums down to $P(x_j|C_i)$ since each predictor variable is independent in Naive Bayes.
The final equation comes down to:

$$P(C_i|\, x_1, x_2 \ldots, x_n) = \left(\prod_{j=1}^{j=n} P(x_j|\, C_i)\right) \cdot \frac{P(C_i)}{P(x_1, x_2 \ldots, x_n)}\ for\ 1 < i < k$$

Here, $P(x_1, x_2, \ldots, x_n)$ is constant for all the classes, therefore we get:

$$P(C_i|\, x_1, x_2 \ldots, x_n)\ \alpha\ \left(\prod_{j=1}^{j=n} P(x_j|\, C_i)\right) \cdot P(C_i)\ for\ 1 < i < k$$

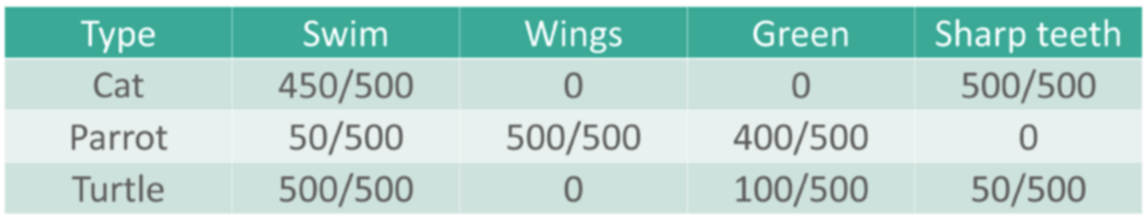
3.2.4 How Does Naive Bayes Work?

To get a better understanding of how Naive Bayes works, let's look at an example.
Consider a data set with 1500 observations and the following output classes:

- Cat
- Parrot
- Turtle

The Predictor variables are categorical in nature i.e., they store two values, either True or False:

- Swim
- Wings
- Green Color
- Sharp Teeth



Type	Swim	Wings	Green	Sharp teeth
Cat	450/500	0	0	500/500
Parrot	50/500	500/500	400/500	0
Turtle	500/500	0	100/500	50/500

From the above table, we can summarize that: The class of type cats shows that:

- Out of $500, 450(90\%)$ cats can swim
- 0 number of cats have wings
- 0 number of cats are of Green color
- All 500 cats have sharp teeth

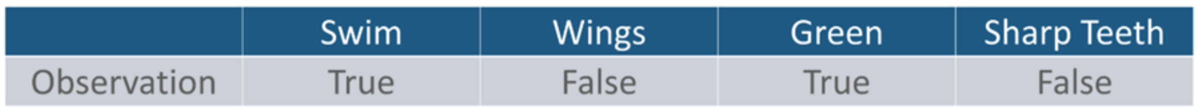
The class of type Parrot shows that:

- $50(10\%)$ parrots have a true value for swim
- All 500 parrots have wings
- Out of $500, 400(80\%)$ parrots are green in color
- No parrots have sharp teeth

The class of type Turtle shows:

- All 500 turtles can swim
- 0 number of turtles have wings
- Out of 500, 100 (20
- 50 out of 500 (10

Now, with the available data, let's classify the following observation into one of the output classes (Cats, Parrot or Turtle) by using the Naive Bayes Classifier.



	Swim	Wings	Green	Sharp Teeth
Observation	True	False	True	False

The goal here is to predict whether the animal is a Cat, Parrot or a Turtle based on the defined predictor variables (swim, wings, green, sharp teeth).
To solve this, we will use the Naive Bayes approach,

$$P(H|\text{Multiple Evidences}) = \frac{P(C_1|H) \times P(C_2|H) \ldots\ldots \times P(C_n|H) \times P(H)}{\text{Multiple Evidences}}$$

In the observation, the variables Swim and Green are true and the outcome can be any one of the animals (Cat, Parrot, Turtle). To check if the animal is a cat:

$$
\begin{aligned}
P(\text{Cat}|\text{Swim, Green}) &= \frac{P(\text{Swim}|\text{Cat}) \times P(\text{Green}|\text{Cat}) \times P(\text{Cat})}{P(\text{Swim, Green})} \\
&= \frac{0.9 \times 0 \times 0.333}{P(\text{Swim, Green})} = 0
\end{aligned}
$$

To check if the animal is a Parrot:

$$
\begin{aligned}
P(\text{Parrot}|\text{Swim, Green}) &= \frac{P(\text{Swim}|\text{Parrot}) \times P(\text{Green}|\text{Parrot}) \times P(\text{Parrot})}{P(\text{Swim, Green})} \\
&= \frac{0.1 \times 0.80 \times 0.333}{P(\text{Swim, Green})} \\
&= \frac{0.0264}{P(Swim, Green)}
\end{aligned}
$$

To check if the animal is a Turtle:

$$
\begin{aligned}
P(\text{Turtle}|\text{Swim, Green}) &= \frac{P(\text{Swim}|\text{Turtle}) \times P(\text{Green}|\text{Turtle}) \times P(\text{Turtle})}{P(\text{Swim, Green})} \\
&= \frac{0.1 \times 0.02 \times 0.333}{P(\text{Swim, Green})} \\
&= \frac{0.0666}{P(Swim, Green)}
\end{aligned}
$$

For all the above calculations the denominator is the same i.e, P(Swim, Green). The value of $P(\text{Turtle}|\text{Swim, Green})$ is greater than $P(\text{Parrot}|\text{Swim, Green})$, therefore we can correctly predict the class of the animal as Turtle.

3.3 What are the types of Naive Bayes classifier?

The main types of Naive Bayes classifier are mentioned below:

- Multinomial Naive Bayes — These types of classifiers are usually used for the problems of document classification. It checks whether the document belongs to a particular category like sports or technology or political etc and then classifies them accordingly. The predictors used for classification in this technique are the frequency of words present in the document.
- Complement Naive Bayes — This is basically an adaptation of the multinomial naive bayes that is particularly suited for imbalanced datasets.
- Bernoulli Naive Bayes — This classifier is also analogous to multinomial naive bayes but instead of words, the predictors are Boolean values. The parameters used to predict the class variable accepts only yes or no values, for example, if a word occurs in the text or not.
- Out-of-Core Naive Bayes — This classifier is used to handle cases of large scale classification problems for which the complete training dataset might not fit in the memory.
- Gaussian Naive Bayes — In a Gaussian Naive Bayes, the predictors take a continuous value assuming that it has been sampled from a Gaussian Distribution. It is also called a Normal Distribution.

3.4 What is Laplace Correction?

When you have a model with a lot of attributes, it is possible that the entire probability might become zero because one of the feature's values is zero. To overcome this situation, you can increase the count of the variable with zero to a small value like in the numerator so that the overall probability doesn't come as zero [3].

This type of correction is called the Laplace Correction. Usually, all naive Bayes models use this implementation as a parameter.

3.5 Advantages and Disadvantages of Naive Bayes and KNN

3.5.1 The naive Bayes algorithm has both its pros and its cons

Pros of Naive Bayes —

1. It is easy and fast to predict the class of the training data set.
2. It performs well in multiclass prediction.
3. It performs better as compared to other models like logistic regression while assuming the independent variables.
4. It requires less training data.
5. It performs better in the case of categorical input variables as compared to numerical variables.

Cons of Naive Bayes —

1. The model is not able to make a prediction in situations where the categorical variable has a category that was not observed in the training data set and assigns a 0 (zero) probability to it. This is known as the 'Zero Frequency'. You can solve this using the Laplace estimation.
2. Since Naive Bayes is considered to be a bad estimator, the probability outputs are not taken seriously.
3. Naive Bayes works on the principle of assumption of independent predictors, but it is practically impossible to get a set of predictors that are completely independent.

3.5.2 The KNN algorithm has both its pros and its cons

Pros of KNN —

1. No Training Period: KNN is a lazy learner, meaning it does not need to learn from the training data beforehand. It saves the training data and only uses it when making predictions, which makes it faster than algorithms that need training. It does not use any function to separate the classes, which simplifies the implementation.
2. Seamless Data Addition: KNN does not need a training period, so adding new data does not affect the algorithm's accuracy. It can easily adapt to changes in the data.

3. Ease of Implementation: KNN is easy to implement, requiring only two parameters: the number of neighbors (K) and the distance function (e.g., Euclidean or Manhattan).

Cons of KNN —

1. Inefficient for Large Datasets: KNN is not suitable for large datasets because it has to calculate the distance between the new point and every existing point, which is costly and time-consuming.
2. Challenges with High Dimensions: KNN has problems with high-dimensional data because it becomes hard to measure the distance in each dimension accurately. The distance function may not capture the similarity well, and some dimensions may be irrelevant or redundant.
3. Feature Scaling Required: Before applying KNN, feature scaling, such as standardization and normalization, is necessary. Otherwise, some features may dominate others and lead to wrong predictions.
4. Sensitive to Noise: KNN is sensitive to noise in the dataset, requiring manual cleaning of missing values and outliers to produce accurate results.

3.6 Compare and contrast K-NN and Naïve Bayes in terms of their assumptions, complexity, performance, and applications

3.6.1 Assumptions:

K-Nearest Neighbors (K-NN):

- It assumes that data points that are similar to each other are also close to each other in the feature space.
- It assumes that all features have equal importance or influence on the similarity calculation.
- It does not make any explicit probabilistic assumption about the data distribution.

Naïve Bayes:

- It assumes that the features are independent of each other. It assumes that the presence (or absence) of one feature does not affect the presence (or absence) of any other feature.
- It uses probabilistic assumptions based on Bayes' theorem to calculate the posterior probabilities.

3.6.2 Complexity

K-NN:

- Complexity increases with the size of the dataset because it requires calculating distances to all data points for each prediction.
- The complexity for a new prediction is $O(N)$, where N is the number of data points in the training dataset.

Naïve Bayes:

- Computationally efficient, especially for high-dimensional data, as it involves simple counting and multiplication.
- The complexity is $O(N \times M)$, where N is the number of data points and M is the number of features.

3.6.3 Performance:

K-NN:

- Can be sensitive to outliers and noise in the data.
- Performs well when the decision boundary is complex and non-linear.
- Works better with small to medium-sized datasets.

Naïve Bayes:

- Tends to be robust to outliers and noise due to the probabilistic nature.
- Performs well with high-dimensional data and is effective in text classification tasks.
- Suitable for both small and large datasets.

3.6.4 Applications:

K-NN:

- Used in recommendation systems, such as collaborative filtering.
- Applied in pattern recognition and image classification.
- Used in anomaly detection and clustering.
- Can be used in both regression and classification tasks.

Naïve Bayes:

- Widely used in text classification tasks like spam detection and sentiment analysis.
- Used in document categorization and email filtering.
- Applicable in medical diagnosis and credit risk assessment.
- Often used for binary and multiclass classification problems

4 Methodology for Text Classification using Naïve Bayes

4.1 Step 1: Data Collection

- Collect a labeled dataset with text documents and corresponding class labels, such as positive or negative sentiment, spam or ham, etc.
- Ensure that the dataset is balanced and representative of the target domain and task.

4.2 Step 2: Data Preprocessing

- Clean the text documents by removing HTML tags, punctuation, and special characters that are irrelevant to the classification task.
- Tokenize the text documents by splitting them into individual words or tokens, such as “she,” “is,” “funny,” etc.
- Convert all text to lowercase to ensure case insensitivity and reduce the vocabulary size.
- Remove stopwords, which are common words (e.g., “the,” “and”) that don’t carry much meaning or discriminative power for the classification task.
- Apply stemming or lemmatization, which are techniques to reduce words to their root form (e.g., “running” to “run”), to further reduce the vocabulary size and normalize the text.

4.3 Step 3: Feature Extraction

Transform the text documents into a numerical representation that can be used by the classifier. There are different methods for this step, such as:

- Bag of Words (BoW): Convert text documents into a matrix where rows represent documents, columns represent terms, and cell values represent the frequency of terms in documents. This method ignores the word order and grammar but captures the word occurrence information.
- Term Frequency-Inverse Document Frequency (TF-IDF): Weight terms based on their frequency in a document and their rarity in the entire corpus. This method assigns higher weights to more important and distinctive terms, and lower weights to more common and irrelevant terms.
- N-grams: Use sequences of n consecutive words as features, instead of single words. This method captures some word order and context information but increases the vocabulary size exponentially.

4.4 Step 4: Model Training

- Split the dataset into a training set and a testing set, using a suitable ratio such as 80:20 or 70:30.
- Train a Naïve Bayes classifier on the training data, using one of its variants such as Multinomial Naïve Bayes or Bernoulli Naïve Bayes, depending on the feature representation method.
- Calculate probabilities for each class and each feature using the training data, applying Bayes’ theorem and making the assumption that features are conditionally independent given the class.

4.5 Step 5: Model Evaluation

- Use the trained model to classify text documents in the testing set, by selecting the class with the highest posterior probability for each document.

- Evaluate the model's performance using various metrics such as confusion matrix, accuracy, precision, recall F1-score, etc by comparing the predicted labels with true labels.
- Implement Laplace smoothing (or add-one smoothing) to handle zero probabilities for unseen terms in the testing data, by adding a small constant to the feature frequencies.

4.6 Concepts:

- **Bag of Words (BoW):** A simple representation of text data that counts the frequency of each word in a document without considering the word's order or grammar.
- **Term Frequency-Inverse Document Frequency (TF-IDF):** A technique that assigns weights to words based on their importance in a document relative to their frequency across all documents.
- **Laplace Smoothing (Add-One Smoothing):** A technique to avoid zero probabilities for unseen words by adding a small constant to the word frequencies.
- **Confusion Matrix:** A table used to evaluate the performance of a classification model, showing the true positives, true negatives, false positives, and false negatives.

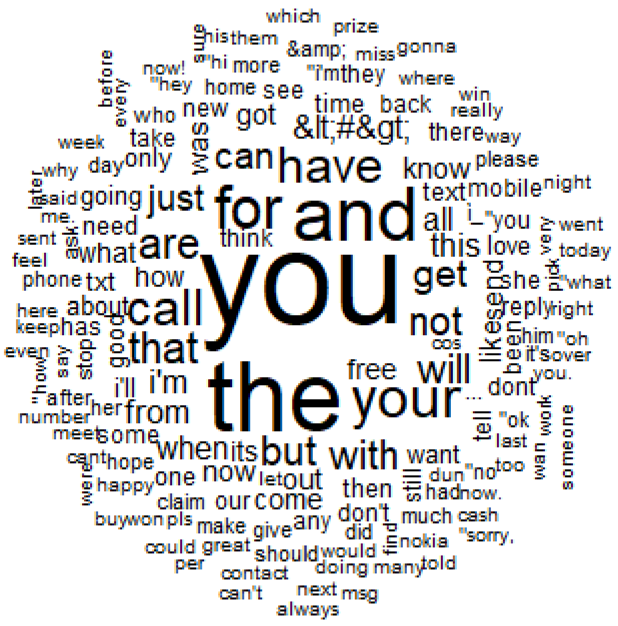
5 Naïve Bayes Classification Model on the SMS DATA

5.1 Objective:

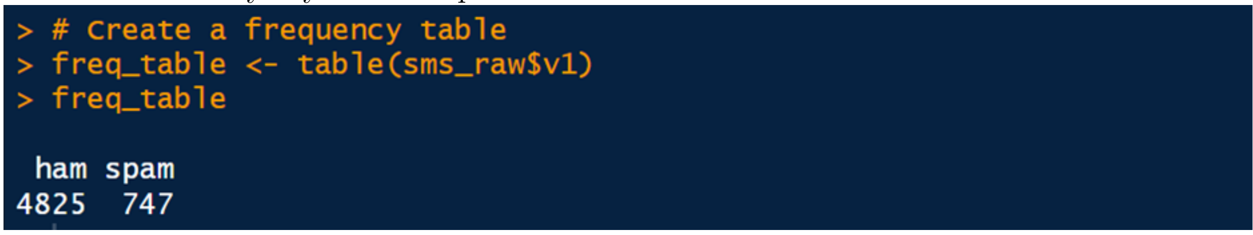
The objective of this analysis is to build and evaluate a text classification model to distinguish between spam and ham (non-spam) messages in an SMS dataset.

5.2 Data Exploration:

- **Loading the Data:** The analysis starts by loading the SMS dataset from a CSV file. The dataset contains two columns: v1 (message labels - spam or ham) and v2 (message text).
- **Data Summary:** The structure of the dataset is examined to understand its format and characteristics. The message labels (v1) are converted into a factor to facilitate analysis.
- **Descriptive Statistics for the Dataset**
  - **Word Cloud:** A word cloud was generated for the dataset to visualize the most frequently occurring words in "spam" and "ham" messages. The word cloud visually represents the importance of words based on their frequency. The larger the word in the cloud, the more frequent it is in spam/ham messages.



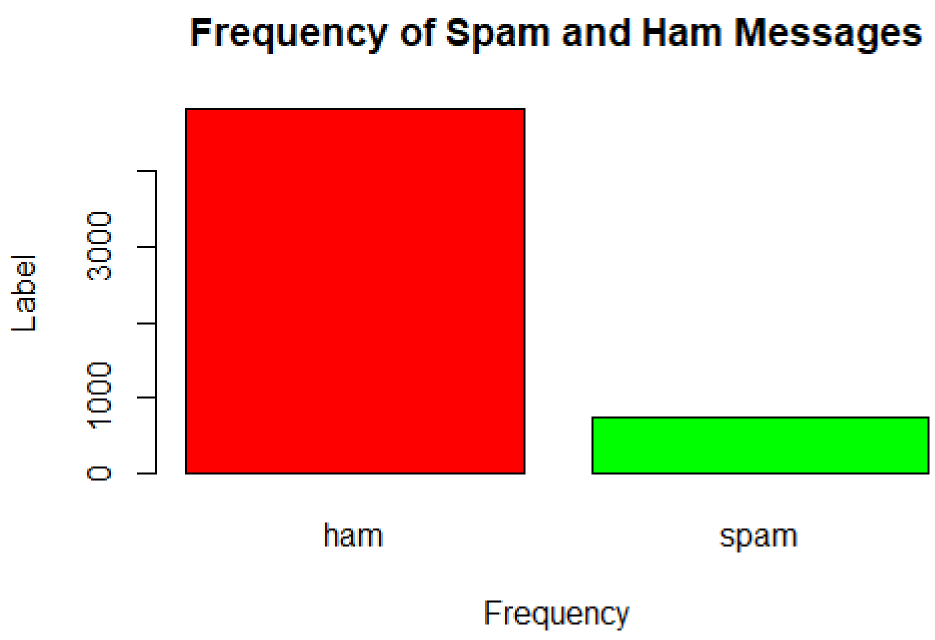
– **Frequency Table:**
  Frequency tables offer detailed insights into the most frequent terms, making it easier to identify key words or phrases in the text.



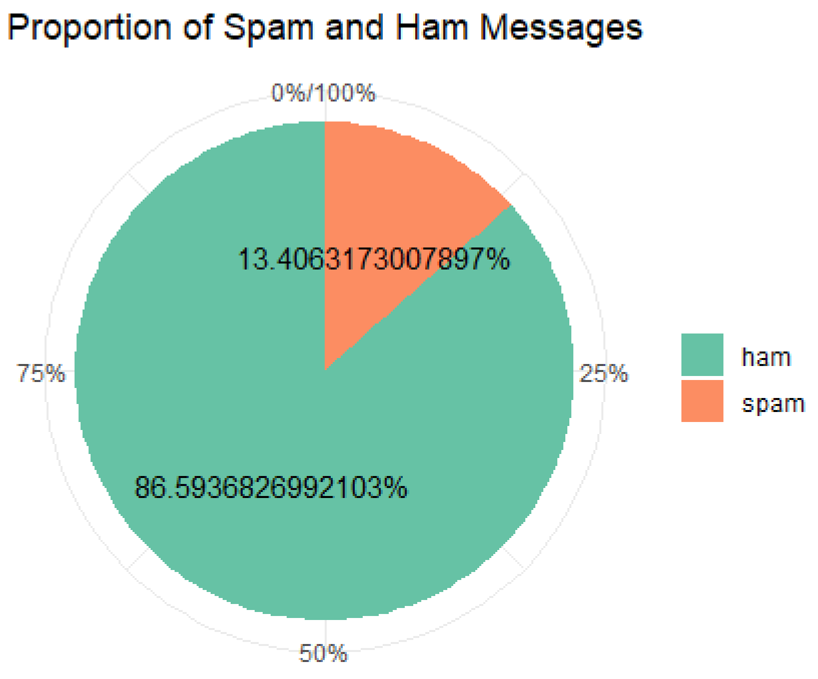
```
> # Create a frequency table
> freq_table <- table(sms_raw$v1)
> freq_table

 ham spam
4825  747
```

– **Bar Graph:**
  A bar graph was created to visualize the difference between spam and ham messages. The graph clearly shows that we have more ham messages than spam massages



- **Pie Chat** A pie chart illustrates the distribution of message types in the "spam" and "ham" datasets. It shows the proportion of spam messages compared to non-spam (ham) messages in the while dataset.



- **Corpus Creation:** The text data is converted into a text corpus using the tm package. A corpus is a collection of text documents and serves as the basis for text analysis.

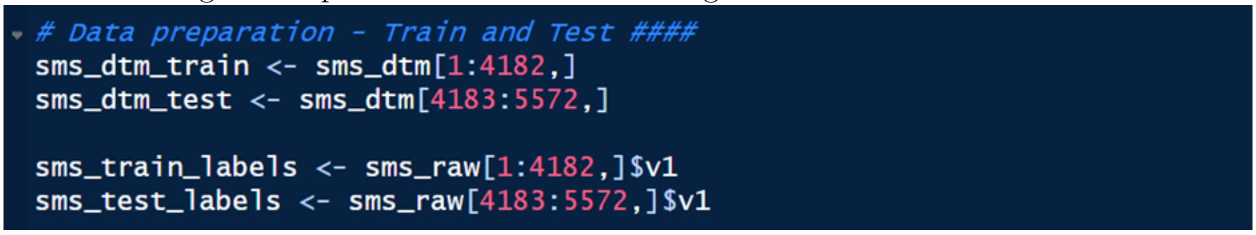
5.3 Data Preprocessing:

Text Cleaning: Text preprocessing is performed to prepare the data for analysis. The following steps are executed:

- Text is converted to lowercase to ensure consistent case handling.
- Numbers are removed to reduce noise in the data.
- Stop words (common words like "and," "the," etc.) are removed.
- Punctuation is removed.
- Stemming is applied to reduce words to their root forms.
- Extra whitespace is removed.
- Tokenization is performed to convert the text into a matrix representation.

5.4 Data Splitting:

Train-Test Split: The dataset is split into training and testing sets. In the original code, the first 4182 rows are used for training, and the remaining rows (4183-5572) are used for testing. This split allows for model training and evaluation.



```
# Data preparation - Train and Test ####
sms_dtm_train <- sms_dtm[1:4182,]
sms_dtm_test <- sms_dtm[4183:5572,]

sms_train_labels <- sms_raw[1:4182,]$v1
sms_test_labels <- sms_raw[4183:5572,]$v1
```

5.5 Feature Selection:

Frequent Word Filtering: The code identifies and selects frequent words (terms appearing at least 5 times) to create indicator features.

5.6 Categorical Features:

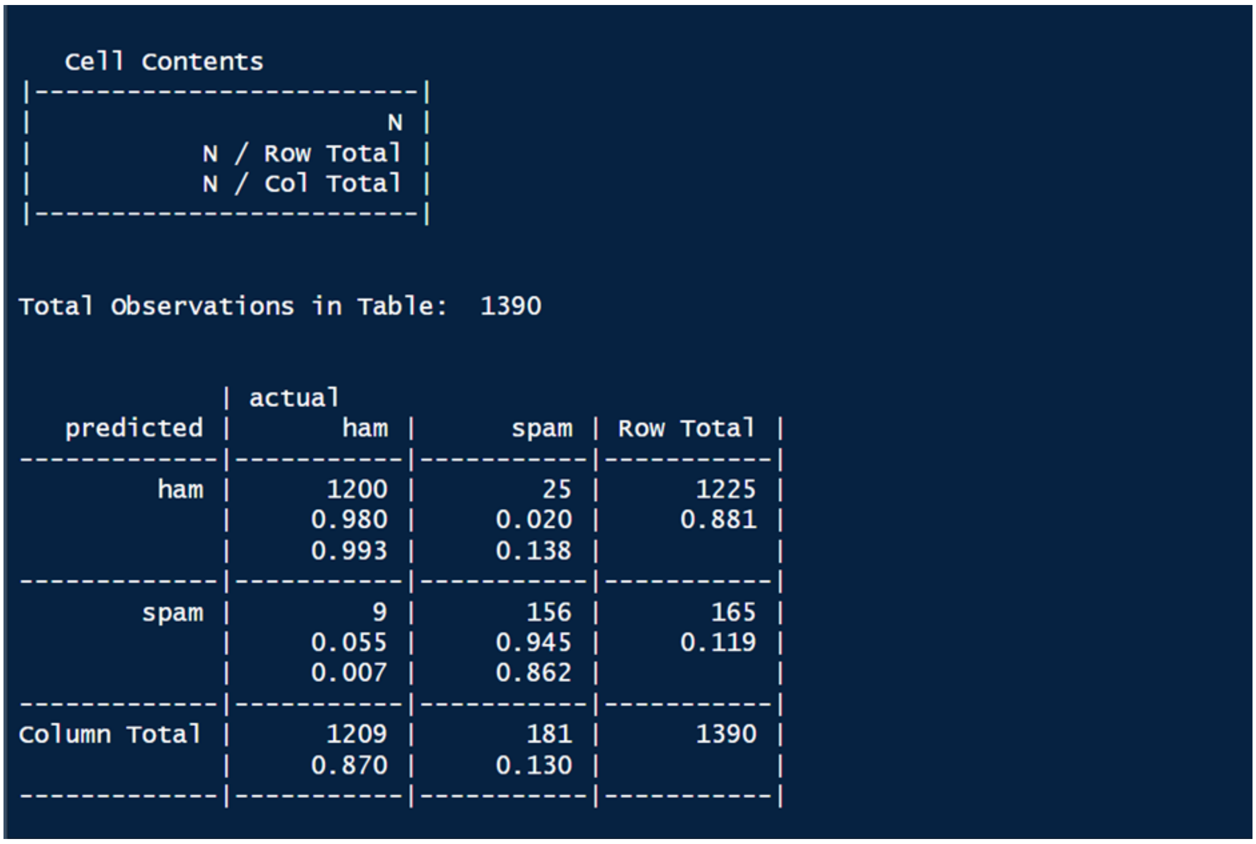
Changing cells in sparse matrix to indicate yes/no since Naive Bayes typically works with Categorical features.

5.7 Model Training and Evaluation:

Naive Bayes Classifier: A Naive Bayes text classification model is trained using the training data. The classifier uses the selected features to distinguish between spam and ham messages.

5.8 Model Evaluation:

The trained model is evaluated using the testing dataset, and its performance is assessed using a confusion matrix. This matrix provides details on true positives, true negatives, false positives, and false negatives.



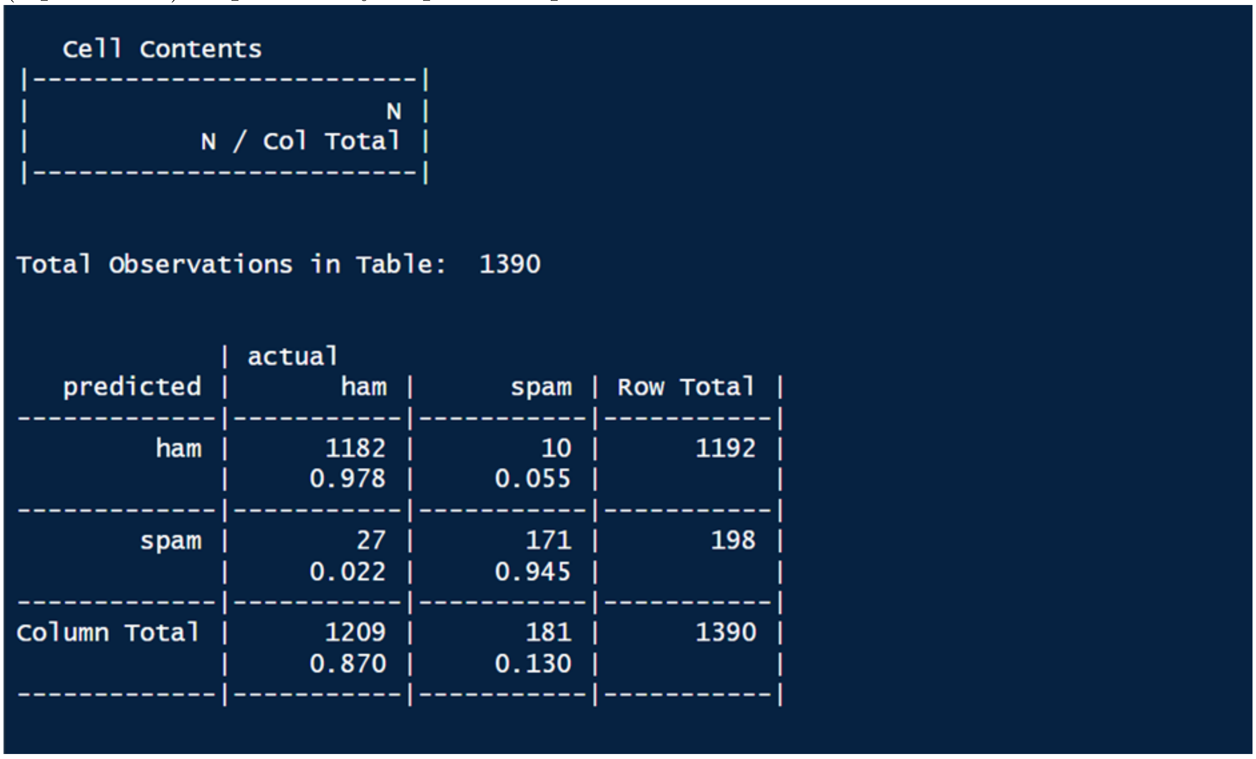
```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  1390


             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1200 |        25 |      1225 |
             |     0.980 |     0.020 |     0.881 |
             |     0.993 |     0.138 |           |
-------------|-----------|-----------|-----------|
        spam |         9 |       156 |       165 |
             |     0.055 |     0.945 |     0.119 |
             |     0.007 |     0.862 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

5.9 Model Improvement:

Laplace Smoothing: The code rebuilds the Naive Bayes model with Laplace smoothing (laplace = 1) to potentially improve its performance.



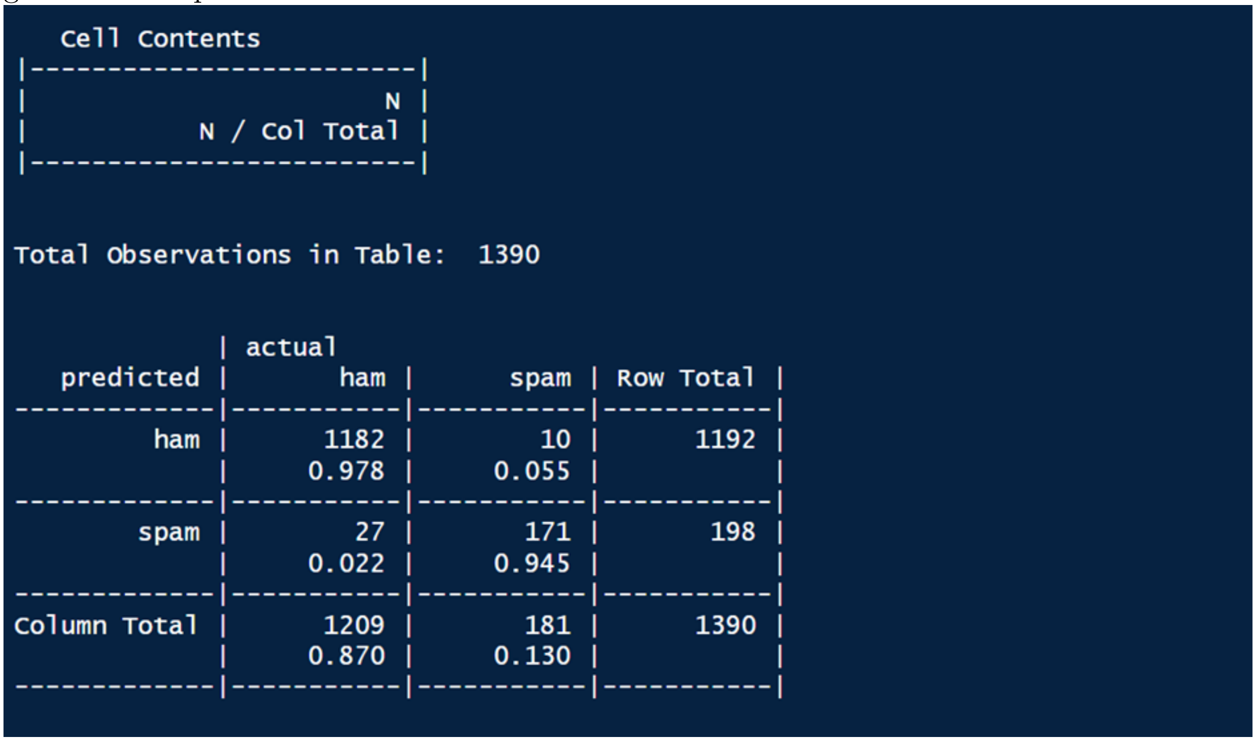
```
   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  1390


             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1182 |        10 |      1192 |
             |     0.978 |     0.055 |           |
-------------|-----------|-----------|-----------|
        spam |        27 |       171 |       198 |
             |     0.022 |     0.945 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

5.10 Evaluation of Improved Model:

The improved model is evaluated using the testing dataset and the confusion matrix is generated for performance assessment.



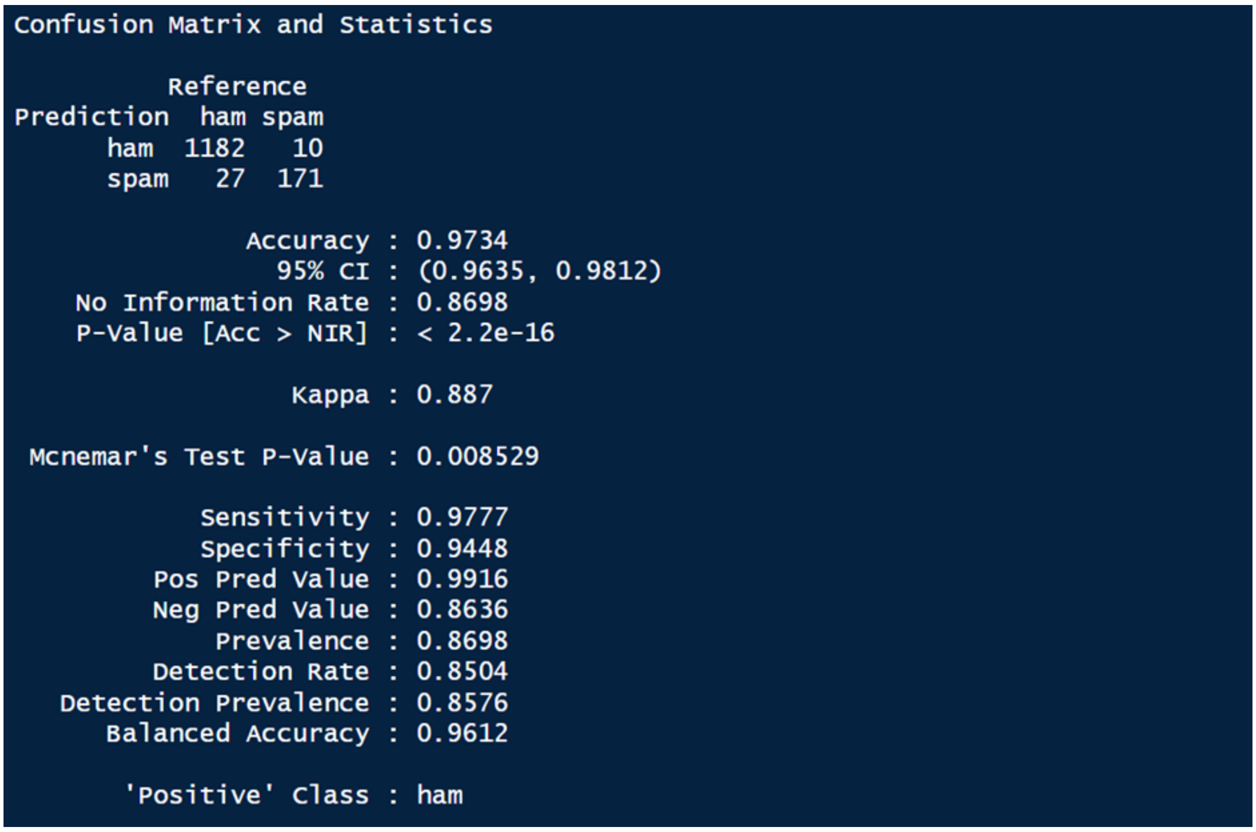
```
   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  1390


             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1182 |        10 |      1192 |
             |     0.978 |     0.055 |           |
-------------|-----------|-----------|-----------|
        spam |        27 |       171 |       198 |
             |     0.022 |     0.945 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

5.11 Result Analysis:

Confusion Matrix: A confusion matrix is generated to assess the model's performance. It provides details on the accuracy of the model in classifying spam and ham messages.



```
Confusion Matrix and Statistics

          Reference
Prediction  ham spam
      ham  1182   10
      spam   27  171

               Accuracy : 0.9734
                 95% CI : (0.9635, 0.9812)
    No Information Rate : 0.8698
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.887

 Mcnemar's Test P-Value : 0.008529

            Sensitivity : 0.9777
            Specificity : 0.9448
         Pos Pred Value : 0.9916
         Neg Pred Value : 0.8636
             Prevalence : 0.8698
         Detection Rate : 0.8504
   Detection Prevalence : 0.8576
      Balanced Accuracy : 0.9612

       'Positive' Class : ham
```

- True Positives (TP): 1,182 messages were correctly classified as "ham."
- True Negatives (TN): 171 messages were correctly classified as "spam."
- False Positives (FP): 10 messages were incorrectly classified as "ham" when they were "spam."
- False Negatives (FN): 27 messages were incorrectly classified as "spam" when they were "ham."
  Model Accuracy: The overall model accuracy is 97.34%, indicating that the model correctly classifies 97.34% of the messages in the test set.

  95% Confidence Interval: The 95% confidence interval for accuracy is (96.35%, 98.12%). This interval provides a range within which the true accuracy of the model is likely to fall.

  No Information Rate (NIR): The no information rate is 86.98%, representing the accuracy if the model always predicted the majority class (in this case, "ham").

  Kappa Statistic: The Kappa statistic measures the level of agreement between the model's predictions and the actual data. A Kappa value of 0.887 indicates a substantial level of agreement.

  Mcnemar's Test P-Value: The Mcnemar's test p-value is 0.008529, suggesting a significant difference in the classification results between different classes.

Sensitivity (True Positive Rate): The sensitivity, or true positive rate, is 97.77%. It indicates the model's ability to correctly identify "ham" messages when they are indeed "ham."

Specificity (True Negative Rate): The specificity, or true negative rate, is 94.48%. It reflects the model's ability to correctly identify "spam" messages when they are indeed "spam."

Positive Predictive Value (Precision): The positive predictive value, or precision, is 99.16%. It represents the proportion of messages classified as "ham" that were correctly identified.

Negative Predictive Value: The negative predictive value is 86.36%. It indicates the proportion of messages classified as "spam" that were correctly identified.

Prevalence: The prevalence of "ham" messages in the dataset is 86.98%.

Detection Rate: The detection rate is 85.04%, representing the rate at which the model correctly detects "ham" messages.

Detection Prevalence: The detection prevalence is 85.76%, indicating the prevalence of messages correctly classified by the model.

Balanced Accuracy: The balanced accuracy is 96.12%, which considers both sensitivity and specificity. It indicates the overall performance of the model.

5.12 Conclusion:

The Naive Bayes text classification model demonstrates excellent performance in distinguishing between "ham" and "spam" messages. With a high accuracy of 97.34%, the model outperforms the no-information rate. It also exhibits substantial agreement with a Kappa statistic of 0.887. Sensitivity and specificity values indicate strong performance in correctly classifying both "ham" and "spam" messages.

The model's positive predictive value (precision) is notably high at 99.16%, signifying accurate identification of "ham" messages. The negative predictive value is also solid at 86.36%.

Overall, the results suggest that the model effectively identifies "ham" and "spam" messages, making it a reliable tool for SMS spam detection.

5.13 Appendix

```r
#install.packages("quanteda") #popular package for
   statistical analysis of text data
require(quanteda)
require(RColorBrewer)
library(ggplot2)
library(tm)
library(class)
library(caret)
library(wordcloud)
library(e1071)
library(gmodels)
library(dplyr)
library(scales)
library(caret)
sms_raw1<-read.csv("C:\\Users\\Haruna\\Desktop\\Data Science
   Materials\\Second_Semester 2023\\Machine Learning and Data
    Mining\\Assignment_Docs\\SMS DATA.csv", encoding = "
   latin1", stringsAsFactors=FALSE)

#attach(spam)
View(sms_raw1)
#View(sms_raw2)

# Reduce the dataset to only two columns (e.g., v1 and v2)
sms_raw <- sms_raw1[, c("v1", "v2")]
View(sms_raw)

# Initial Dataset Exploration
str(sms_raw)

# Convert categorical variable into factor
sms_raw$v1 <- as.factor(sms_raw$v1)

# EDA
# Descriptive Statistics
str(sms_raw)

# Create a frequency table
freq_table <- table(sms_raw$v1)
freq_table

# Create a bar graph
barplot(freq_table, col = c("red", "green"), main = "
   Frequency of Spam and Ham Messages", xlab = "Frequency",
   ylab = "Label")

# Calculate the proportions of spam and ham messages
proportions <- prop.table(table(sms_raw$v1))
```

```r
# Convert the table object to a data frame
proportions_df <- as.data.frame(proportions)

# Use the fortify function on the data frame
proportions_df <- fortify(proportions_df, region = "Var1")

# Create a pie chart
ggplot(proportions_df, aes(x = "", y = Freq, fill = Var1)) +
  geom_col(width = 1) +
  coord_polar(theta = "y") +
  scale_y_continuous(labels = percent_format()) +
  scale_fill_brewer(palette = "Set2") +
  labs(title = "Proportion of Spam and Ham Messages", x =
     NULL, y = NULL, fill = NULL) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_text(aes(label = paste0(Freq * 100, "%")), position =
     position_stack(vjust = 0.5))

# Creating a word cloud to visualize the text data
wordcloud(sms_raw, min.freq = 50, random.order = F)

# Creating a corpus
sms_corpus <- VCorpus(VectorSource(sms_raw$v2))

# For potential additional options offered by the "tm"
   package:
#vignette("tm")

inspect(sms_corpus[1:5])

# to check for a single message we can use the as.character()
   function as well as double-brackets
as.character(sms_corpus[[400]])

# Using lapply() to print several messages
lapply(sms_corpus[1:10], as.character)

# converting the corpus to lower-case to start cleaning up
   the corpus
sms_corpus_clean <- tm_map(sms_corpus,
                           content_transformer(tolower))

# Comparing first sms to check result
as.character(sms_corpus[[1]])
as.character(sms_corpus_clean[[1]])

# Removing numbers to reduce noise (numbers will be unique
    and will not provide useful patters across all messages)
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)

# Check results
lapply(sms_corpus_clean[1:10], as.character)

# Removing "stop words" (to, and, but, or) and punctuation
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords,
    stopwords())

sms_corpus_clean <- tm_map(sms_corpus_clean,
    removePunctuation)

# Stemming
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)

# Remove additional whitespace
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)

# Tokenization
sms_dtm <- DocumentTermMatrix(sms_corpus_clean) # Thanks to
    the previous preprocessing the object is ready

# Data preparation - Train and Test ####
sms_dtm_train <- sms_dtm[1:4182,]
sms_dtm_test <- sms_dtm[4183:5572,]

sms_train_labels <- sms_raw[1:4182,]$v1
sms_test_labels <- sms_raw[4183:5572,]$v1

# Comparing proportion of SPAM
prop.table(table(sms_train_labels))
prop.table(table(sms_test_labels))

# Creating a word cloud to visualize the text data
wordcloud(sms_corpus_clean, min.freq = 50, random.order = F)

# Creating subset of SPAM sms to visualize later
spam <- subset(sms_raw, v1 == "spam")
ham <- subset(sms_raw, v1 == "ham")

# Visualizing both types separately
wordcloud(spam$v2, max.words = 40, scale = c(3, 0.5), random.
    order = F)
wordcloud(ham$v2, max.words = 40, scale = c(3, 0.5), random.
    order = F)

# Data preparation - creating indicator features for frequent
```

```
    words####
# Filtering out unfrequent words
sms_freq_words <- findFreqTerms(sms_dtm_train, 5) # function
   to find all terms appearing at least 5 times
sms_dtm_freq_train <- sms_dtm_train[, sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[,sms_freq_words]

# Changing cells in sparse matrix to indicate yes/no since
   Naive Bayes typically works with Categorical features
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)

# Training model on the data ####
sms_classifier <- naiveBayes(sms_train, sms_train_labels)

# Evaluating model performance
sms_test_pred <- predict(sms_classifier, sms_test)

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

# Improving Model Performance

# Rebuilding Naive Bayes with laplace = 1
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels,
                              laplace = 1)

# Evaluating 2nd model's performance
sms_test_pred2 <- predict(sms_classifier2, sms_test)

CrossTable(sms_test_pred2, sms_test_labels,
           prop.chisq = F, prop.t = F, prop.r = F,
           dnn = c('predicted', 'actual'))

# Confusion matrix
conf_matrix <- confusionMatrix(data = sms_test_pred2,
   reference = sms_test_labels)
conf_matrix
```

References

[1] Daniel Berrar. Bayes' theorem and naive bayes classifier. *Encyclopedia of bioinformatics and computational biology: ABC of bioinformatics*, 403:412, 2018.

[2] Zulaikha Lateef. Knn algorithm: A practical implementation of knn algorithm in r. *Edureka*, March 2022.

[3] Zulfany Erlisa Rasjid and Reina Setiawan. Performance comparison and optimization of text document classification using k-nn and naïve bayes classification techniques. *Procedia computer science*, 116:107–112, 2017.

[4] Shichao Zhang, Xuelong Li, Ming Zong, Xiaofeng Zhu, and Debo Cheng. Learning k for knn classification. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(3):1–19, 2017.

[5] Lateef Zulaikha. A comprehensive guide to naive bayes in r. *Edureka*, May 2020.