Analysis Report: Spam Message Detection using Naive Bayes Classification.

Report By: Tausif Utchhash

Dataset: sms\_spam.csv

Algorithm: Naive Bayesin

Package: naiveBayes() function in “e1071” package

Analysis Report: problem description, introduction of Naïve Bayesian algorithm, how to process plain text files using tm\_mpa() function in the “tm” package to remove punctions and stop words and perform stemming and changing cases, how to split messages into a document-to-term matrix, how to visualize word clouds, how model is trained and evaluated, and how model is improved

**Project Description:** In our modern world, where text messages are a fundamental part of communication, the constant intrusion of spam messages has become a significant issue. Dealing with unwanted messages not only disrupts personal communication but also poses potential risks like scams and fraudulent activities. Addressing this issue requires an effective SMS spam detection system, and the key to achieving this lies in the realm of machine learning.

Machine learning offers a transformative approach by allowing systems to learn and adapt based on patterns observed in large datasets. This capability is crucial in dealing with the ever-changing tactics used by spammers. Unlike rule-based methods that may struggle to keep up, machine learning models excel in noticing subtle details and understanding intricate patterns. within the content of SMS messages, enabling them to differentiate between genuine messages and spam with high accuracy.

Among the various algorithms in text classification, Naive Bayes stands out as a particularly suitable choice for SMS spam detection. Its simplicity and efficiency make it well-suited for handling the complexities of high-dimensional data, such as the numerous words present in text messages. Naive Bayes operates on the assumption that each word in a message is independent, a simplification that proves surprisingly effective in practice, especially when dealing with large datasets.

Moreover, the computational efficiency of Naive Bayes is noteworthy, making it adept at processing incoming SMS messages in real-time. This capability is crucial for promptly identifying and blocking spam messages before they reach users. The model's ability to continuously learn and adapt to new spam patterns ensures its effectiveness over time, which provides a dynamic defense against evolving spam tactics.

In summary, the fusion of machine learning, particularly through Naive Bayes Classification, provides a strong and flexible solution to the ongoing challenge of SMS spam. By tapping into the inherent patterns within the text of messages, this approach not only improves the efficiency of spam detection but also lays the groundwork for secure and simplified mobile communication channels in our interconnected world.

**Naive Bayesian Classification:** Naive Bayes is a probabilistic algorithm based on Bayes' theorem. Bayes’ theorem is a mathematical formula for calculating conditional probabilities. It is a simple and effective classification algorithm commonly used in machine learning for tasks such as spam filtering, text classification, and sentiment analysis. To understand Naive Bayes Classification we need to first understand the Bayes’ Theorem. Bayes' theorem is the foundation of Naive Bayes classification. It calculates the probability of a hypothesis (class) given the evidence (features). Mathematically it is expressed as : P(A|B) = {P(B|A)\*P(A)} / P(B)

In this mathematical expression,

P(A|B) is the probability of class A given the features B.

P(B∣A) is the probability of observing the features B given class A.

P(A) is the prior probability of class A.

P(B) is the probability of observing the features B.

**Naive Assumption:** The "naive" part of Naive Bayes comes from the assumption of independence between features. It assumes that when the class label is given the presence or absence of a particular feature is independent of the presence or absence of any other feature. Even though, this assumption simplifies the calculations significantly, this assumption is often unrealistic.   
  
**Working of Naive Bayes:** Naive Bayes works in two phases. First the training phase and then the testing phase.

**Training Phase:** To make things easier to understand let’s assume that we are working with a dataset that consists of SMS data. In this dataset each message is labeled as either spam or ham (not spam). After collecting the dataset, the data needs to be prepared for use and for that it is necessary to tokenize the messages into words or features. After tokenization of the messages, the prior probabilities P(spam) and P(ham) needs to be calculated. This involves counting the occurrences of spam and ham messages in the dataset and dividing them by the total number of messages. Next in the training phase, the conditional probabilities P( and P( are calculated. P( is the probability of observing given that the message is spam. P( is the probability of observing given that the message is ham(not spam).

**Testing Phase:** After receiving a new message the objective is to determine whether the incoming message is spam or ham. The message once again is tokenized into words or features. Then the posterior probability is calculated. The posterior probability is the probability of a specific class given observed evidence. For example, the posterior probability equation for spam and ham class will be:

* P(spam|, , …, ) | P(spam) \* P( | spam)
* P(ham|, , …, ) | P(ham) \* P( | ham)

For each type of message (spam or not spam), the computer has already learned from many other messages. Now, it calculates how likely it is for this new message to be spam or not spam based on the words it sees. It calculates the probability of this message being spam and the chance of it not being spam. Whichever type (spam or not spam) has the higher chance, the computer flags the income message as spam or ham. Spam if P(spam|, , …, ) > P(ham|, , …, ) otherwise, ham.

**The Laplace Estimator:** The Laplace estimator is a technique used to handle the issue of zero probabilities in Naive Bayes classification. The problem arises when a particular feature or word in the data has not been observed in a specific class during training, leading to a conditional probability of zero. This situation can be problematic when applying Bayes' theorem during the classification phase. In Naive Bayes, when calculating conditional probabilities for features given a class, having a zero probability can cause the entire posterior probability to be zero. Laplace smoothing helps to avoid this issue by assigning a small, non-zero probability to unseen features in a class. Text data often exhibits sparsity. Which means many words occur infrequently. Laplace smoothing addresses the sparsity problem as well by providing a minimal probability for words not observed in a specific class. Laplace estimator makes Naive Bayes Classification more robust. It ensures that probabilities are distributed evenly. Laplace estimator also prevents overfitting. Let’s consider a scenario where the word "promotion" did not appear in any spam messages during training. Without Laplace smoothing, P(promotion|spam) would be zero. With Laplace smoothing, it becomes a small positive value, preventing the multiplication by zero and allowing for a stronger probability calculation.

Laplace estimator adjusts the conditional probability calculation using the following equation:

P(|) = (Count of in +1) / (Total count of words in +Vocabulary size.

Here,

* is a specific feature or word.
* is a particular class.
* The "+1" in the numerator is the Laplace smoothing term.
* The denominator includes the total count of words in the class and the size of the vocabulary.

**Understanding and preprocessing the data:** For this project we used the sms\_spam.csv dataset and upon reviewing the dataset we can see that the spam messages tend to have the words “urgent” or “free” in them.  
 A close-up of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

Yes, It is possible to have ham messages with the words “urgent” or “free” but the ham messages would provide further context like “urgent need of assistance” or “are you free for coffee” but in spam the phrases are like “free game” or “urgent booking”. The classifier will detect a message to be spam or ham based on evidence provided by all the words present in the message. The dataset is then saved to a data frame and we can see that there are 5559 objects of 2 variables.

A close-up of a text

Description automatically generated

A white background with blue text

Description automatically generated

The class distribution suggests that there are 4812 ham text messages and 747 spam text messages.

A black and white text

Description automatically generated

A screenshot of a computer code

Description automatically generated

After forming a volatile corpus and inspecting it, it is visible that the documents in the dataset vary in size. Corpus is a collection of text documents. It is getting called volatile as it is stored in memory rather than being stored on disk.



A screenshot of a computer

Description automatically generated

Using lapply() function, we can see the actual text messages. Here, from 1 to 10 is displaced as we gave the command [1:10].

Before going any further we need to clean the data first. It is necessary to remove punctuation and other characters that clutter the result to standardize the words for better analysis.

First, the message needs to be standardized in a format where it uses only lowercase characters. Then the numbers must be removed. All the stop words are removed next as well as the punctuation. But as removing punctuation blindly can sometimes lead to unintended consequences like two words getting joint together when the ideal scenario is to have a blank space between them. This is why it is better to create a custom function that replaces the punctuation rather than removing them blindly. Moreover, stemming is done. This process strips all the suffixes and transforms the word to its base form. This means “jump”, “jumping”, “jumped”, “jumps” all are transformed to “jump”. Finally, the additional white spaces are removed as after removing numbers, stop words, punctuation and after stemming, the text messages are left with unwanted blank spaces.

The sms\_data\_dtm object is created that contains the tokenized corpus.

A close-up of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

The printed texts above show us what we need for better analysis.

A computer screen shot of a computer code

Description automatically generated

In this code block the same preprocessing steps are applied in the same order but if sms\_data\_dtm and sms\_data\_dtm1 are compared, we would see discrepancy.

A screenshot of a computer code

Description automatically generated

The reason for this discrepancy is the minor difference in the ordering of the preprocessing steps and the fact that some words split differently than when they are cleaned before tokenization.

To visualize which word features occur frequently, we need to see a word cloud.

A screenshot of a computer code

Description automatically generated



A circle of words

Description automatically generated

In a word cloud most frequently used word features in spam messages are shown as red.

A black and white text

Description automatically generated

A close up of words

Description automatically generated

In the next word cloud most frequently used word features in ham messages are shown as green.

A black text with blue and green text

Description automatically generated with medium confidence

A close up of words

Description automatically generated

**Training the model:** The sms\_data\_dtm is split into training and testing randomly. The split is random but as set.seed(2) function is used, the results will always be replicable.

A computer code with text

Description automatically generated

A white background with blue text

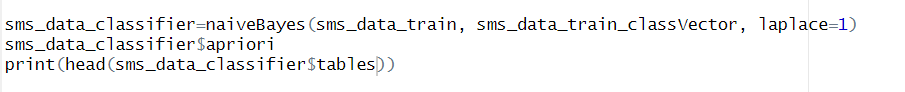
Description automatically generated

Then adjustments are made on the clean and split training and testing data because the minimum frequency or count in the dataset needs to be reasonable. The Naive Bayes classifier usually works with categorical data, but when dealing with word frequency data, the numeric values in the matrix can be a challenge. To resolve this, we should convert these numeric values to a simpler categorical form that just indicates whether a word is present in a message or not.

A computer code with text

Description automatically generated

Finally, the model is created using the Naive Bayes Algorithm.



A screenshot of a computer screen

Description automatically generated

**Naive Bayes Classification Result Analysis:** The confusion matrix below shows that among 1240 ham text messages 1220 text messages were correctly pinned as ham and 20 were incorrectly pinned as spam. Among 191 spam text messages 183 were correctly flagged as spam but 8 ham text messages were flagged as spam.   
A close-up of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

From the Confusion Matrix and Statistics results it can be said that we are 95% confident that the true accuracy lies between 97.18% and 98.7%. Kappa of 91.76% means there is strong agreement between the model's predictions and the actual classes. The model correctly identifies 98.39% of actual "ham" messages and it correctly identifies 95.81% of actual "spam" messages. PPV of 99.35%, it means there's a 99.35% chance that a predicted "ham" message is genuinely "ham and NPV of 90.15%, it means there's a 90.15% chance that a predicted "spam" message is genuinely "spam. The Detection rate of 85.26% means the model correctly predicts 85.26% of all messages.

**Optimizing Naive Bayes Classification:** The model can be optimized even more with more training data. Hyper parameter tuning would also help optimize the model. If the training data consists of different text representations, it’d make detecting spam messages even more efficient. Parallelization could be used to expedite the training process.

**Conclusion:** Naive Bayes classification is a great method for spotting spam in text messages. It works well because it's simple, handles big data sets efficiently, and figures out if a message is spam or not by looking at the likelihood of certain characteristics. Even though it assumes that different features don't really affect each other, which sometimes gives obscure results, the majority of time it turns out to be quite effective, especially with text data.