SMS Spam Classifier: A Detailed Report

Table of Contents

1. Introduction

1.1 The Problem of SMS Spam

In today's digital age, Short Message Service (SMS) remains a ubiquitous form of communication. However, its widespread use has also made it a prime target for unsolicited and often malicious messages, commonly known as spam. SMS spam can range from annoying advertisements to phishing attempts and malware distribution, posing a significant threat to user privacy and security. The sheer volume of SMS traffic makes manual filtering impractical, necessitating the development of automated and intelligent spam detection systems.

1.2 Objective of the Project

The primary objective of this project is to develop a robust and accurate SMS spam classifier. By leveraging natural language processing (NLP) techniques and machine learning algorithms, we aim to build a model that can effectively distinguish between legitimate messages (ham) and spam. This report details the entire process, from data exploration and preprocessing to model building and evaluation, providing a comprehensive overview of the methodologies employed and the results achieved.

2. Data Loading and Initial Exploration

2.1 Loading the Dataset

The first step in our project is to load the dataset, which is contained in a CSV file named spam.csv. We use the pandas library in Python for this purpose, specifying the encoding as "latin-1" to handle any special characters that may be present in the text messages.

2.2 Initial Data Inspection

Upon loading the data, we perform an initial inspection to understand its structure and content. The dataset contains 5,572 messages, each with five columns. The first two columns, v1 and v2, contain the label (ham or spam) and the message text, respectively. The remaining three columns, Unnamed: 2, Unnamed: 3, and Unnamed: 4, appear to be largely empty and do not provide any useful information.

3. Data Cleaning

3.1 Handling Missing Values

A crucial step in any data analysis project is to handle missing values. In our dataset, the last three columns have a significant number of missing values and are therefore dropped from the dataframe. The v1 and v2 columns, which are essential for our analysis, have no missing values.

3.2 Removing Duplicates

To ensure the quality of our training data, we check for and remove any duplicate messages. The dataset contains 403 duplicate entries, which are removed to prevent any bias in our model.

3.3 Renaming Columns

For better readability and understanding, we rename the v1 and v2 columns to Target and Message, respectively. This makes the code and

subsequent analysis more intuitive.

4. Exploratory Data Analysis (EDA)

4.1 Class Distribution

We begin our EDA by examining the distribution of ham and spam messages in the dataset. The analysis reveals a significant class imbalance, with approximately 87.3% of the messages being ham and only 12.7% being spam. This imbalance is an important consideration for model training and evaluation.

4.2 Analysis of Message Length

We analyze the length of the messages for both ham and spam categories. The average length of a spam message is significantly higher than that of a ham message. This suggests that message length could be a useful feature for our classifier.

4.3 Analysis of Word Count

Similar to message length, we also analyze the number of words in each message. The findings are consistent with our analysis of message length, with spam messages generally containing more words than ham messages.

4.4 Visualizing Key Features

To further understand the characteristics of ham and spam messages, we create visualizations such as histograms and word clouds. The histograms of message length and word count clearly show the difference in distribution between the two classes. The word clouds provide a visual representation of the most frequent words in both ham and spam messages, highlighting the distinct vocabulary used in each category.

5. Data Preprocessing

5.1 Text Transformation

Before feeding the text data into our machine learning models, it needs to be preprocessed. This involves several steps:

- **Lowercasing:** Converting all text to lowercase to ensure consistency.
- **Removing Digits and Punctuation:** Eliminating numbers and punctuation marks that do not contribute to the meaning of the message.
- **Tokenization:** Splitting the text into individual words or tokens.

5.2 Stemming and Stopword Removal

To further refine our text data, we perform stemming and stopword removal.

- **Stopword Removal:** Removing common words (e.g., "the," "is," "in") that do not carry significant meaning.
- **Stemming:** Reducing words to their root form (e.g., "running" to "run") to group together different inflections of the same word.

6. Model Building

6.1 Vectorization Techniques

Since machine learning models cannot work directly with text data, we need to convert the preprocessed messages into numerical vectors. We explore two popular vectorization techniques:

6.1.1 CountVectorizer

CountVectorizer converts a collection of text documents to a matrix of token counts. Each row represents a document, and each column represents a unique word in the vocabulary. The value in each cell is the frequency of that word in the corresponding document.

6.1.2 TF-IDF Vectorizer

Term Frequency-Inverse Document Frequency (TF-IDF) is another

vectorization technique that reflects the importance of a word in a document relative to a collection of documents. It gives more weight to words that are frequent in a document but rare in the overall corpus.

6.2 Naive Bayes Classifiers

For our classification task, we use three variants of the Naive Bayes algorithm, which is a popular choice for text classification due to its simplicity and efficiency.

6.2.1 Gaussian Naive Bayes

Gaussian Naive Bayes is suitable for continuous data and assumes that the features follow a Gaussian distribution.

6.2.2 Multinomial Naive Bayes

Multinomial Naive Bayes is well-suited for discrete data, such as word counts, and is commonly used in text classification tasks.

6.2.3 Bernoulli Naive Bayes

Bernoulli Naive Bayes is another variant that is effective for binary or boolean features.

7. Results and Evaluation

7.1 Performance Metrics

To evaluate the performance of our models, we use several metrics:

- **Accuracy:** The proportion of correctly classified instances.
- **Precision:** The proportion of true positives among all instances classified as positive.
- **Recall:** The proportion of true positives that were correctly identified.
- **F1-Score:** The harmonic mean of precision and recall.

7.2 Model Performance with CountVectorizer

When using CountVectorizer, the Bernoulli Naive Bayes model achieves the highest accuracy and precision, indicating its effectiveness in this scenario.

7.3 Model Performance with TF-IDF Vectorizer

With the TF-IDF vectorizer, the Multinomial Naive Bayes model demonstrates the best performance, achieving a high accuracy and a perfect precision score of 1.0.

7.4 Confusion Matrices

We also generate confusion matrices for each model to visualize their performance. The confusion matrices provide a detailed breakdown of the number of true positives, true negatives, false positives, and false negatives, offering further insights into the models' classification capabilities.

8. Conclusion

8.1 Summary of Findings

This project successfully demonstrates the application of NLP and machine learning techniques for SMS spam classification. Our analysis reveals that spam and ham messages have distinct characteristics in terms of length, word count, and vocabulary. By leveraging these features, we were able to build effective classifiers.

8.2 Best Performing Model

The combination of the TF-IDF vectorizer and the Multinomial Naive Bayes classifier emerged as the best-performing model, achieving an impressive accuracy of 97.29% and a precision of 1.0. This indicates that the model is highly effective at identifying spam messages with a very low false positive rate.

8.3 Future Work

While the current model performs well, there is always room for improvement. Future work could involve exploring more advanced NLP techniques, such as word embeddings and deep learning models, to further enhance the accuracy and robustness of the classifier. Additionally, incorporating real-time learning capabilities would allow the model to adapt to new and evolving spamming techniques.