SMS/Email Spam Detection Using Machine Learning

Shashank Kurakula

Introduction

The proliferation of mobile phones has led to an increase in unwanted text messages, commonly known as SMS spam. These unsolicited messages can be annoying and potentially harmful. In this report, we explore the use of machine learning techniques to detect and classify SMS spam.

Dataset Description

Our dataset comprises a collection of labeled SMS messages, where each message is categorized as either “spam” or “ham” (non-spam). The dataset was sourced from Kaggle, a popular platform for data science competitions and datasets. The dataset was downloaded from the Kaggle website https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset?resource=download.

The dataset contains the following features:

1. **Text Content**: The actual content of the SMS messages.
2. **Label**: The binary label indicating whether the message is spam or not.

Exploratory Data Analysis

Before diving into model building, we perform exploratory data analysis (EDA) to gain insights into the dataset. Some key EDA steps include:

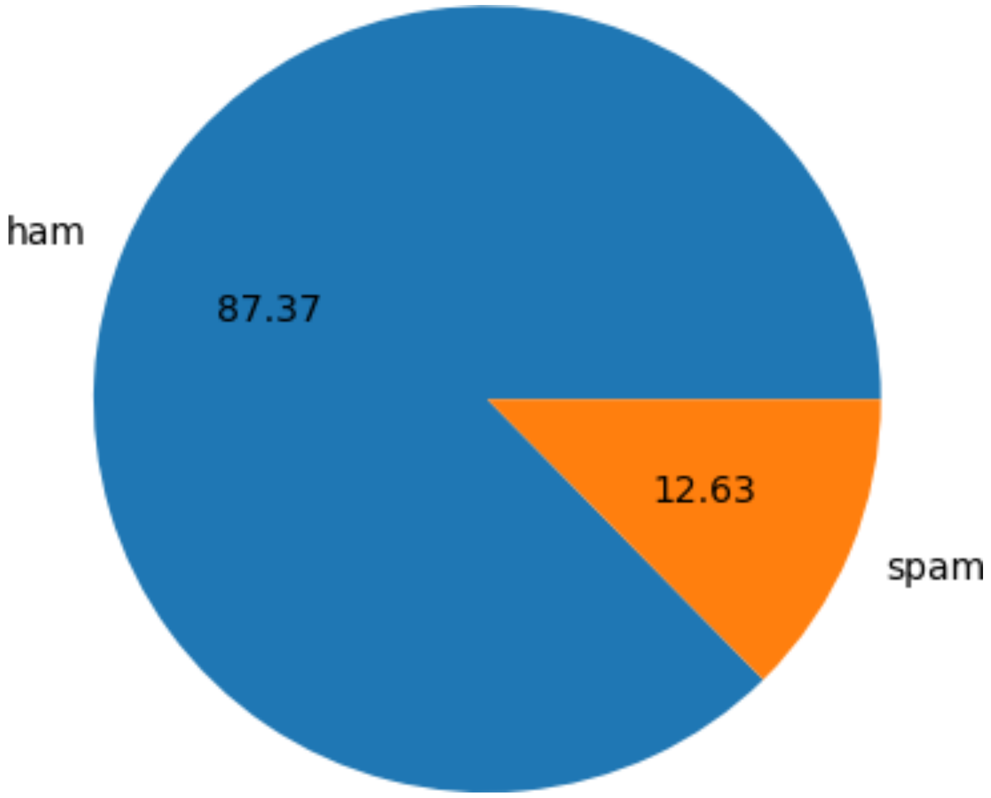
1. **Distribution of Labels**: We visualize the distribution of spam and ham messages to understand the class balance. Imbalanced classes may impact model performance.
2. **Word Clouds**: Creating word clouds for both spam and ham messages helps identify common terms associated with each category.

3. **Message Length Analysis**: We explore the relationship between message length and spam/ham classification.

Data Cleaning

In this initial step, we focus on preparing our dataset for further analysis and modeling. Here are the specific actions we take:

- **Column Selection and Renaming**: We drop columns 2, 3, and 4 due to missing values or low relevance. Since the column names are unspecified, we rename them using the `df.rename()` function.
- **Label Encoding**: The target column contains non-numerical values (spam and ham). To make it compatible with machine learning algorithms, we convert these labels into numeric values. We achieve this using the `LabelEncoder` from `sklearn.preprocessing`. Specifically, we apply `label_encoder.fit_transform(target)` and update the target column in our dataset.



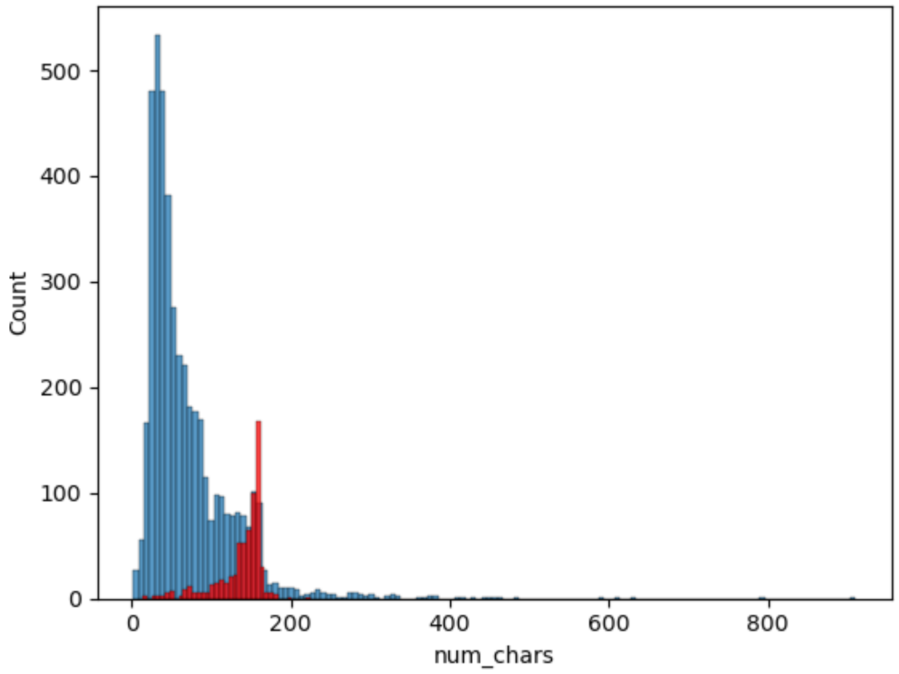
We utilize the Natural Language Toolkit (nltk) library for text processing, specifically focusing on the analysis of messages. As part of our preprocessing steps, we download the Punkt tokenizer, which is essential for sentence and word tokenization.

1. Character, Word, and Sentence Counts:

For each message in our dataset, we calculate the following metrics:

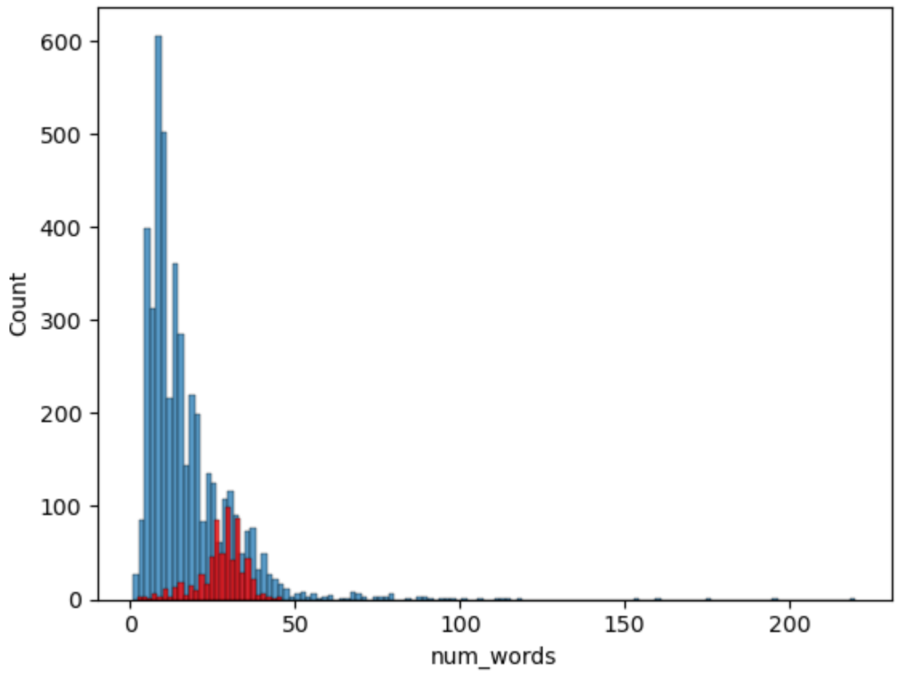
- Number of characters (using `apply(len)`).
- Number of words (using `nltk.word_tokenize()`).
- Number of sentences (using `nltk.sent_tokenize()`).

These metrics provide valuable insights into the length and complexity of the messages.



2. Exploratory Data Analysis (EDA):

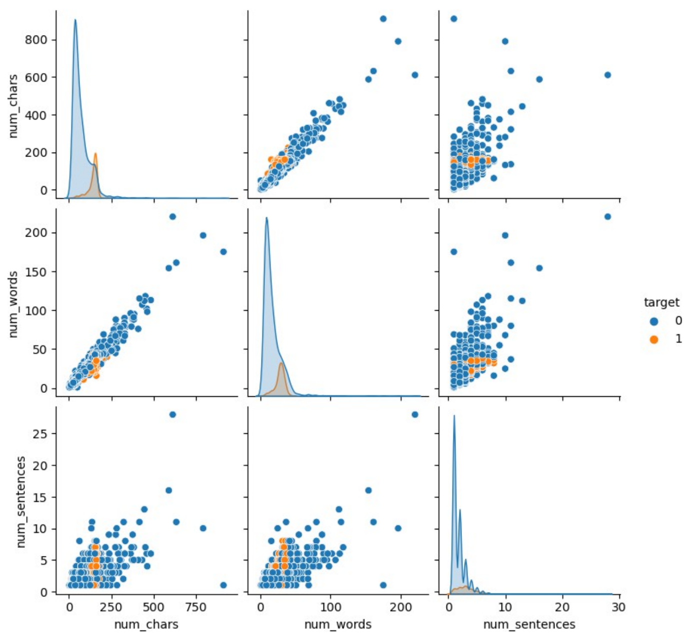
- We examine summary statistics separately for ham (legitimate) and spam messages.
- Notably, ham messages tend to have more characters and words compared to spam messages.



- We visualize these differences using seaborn (seaborn) plots.

3. Pairplot Visualization:

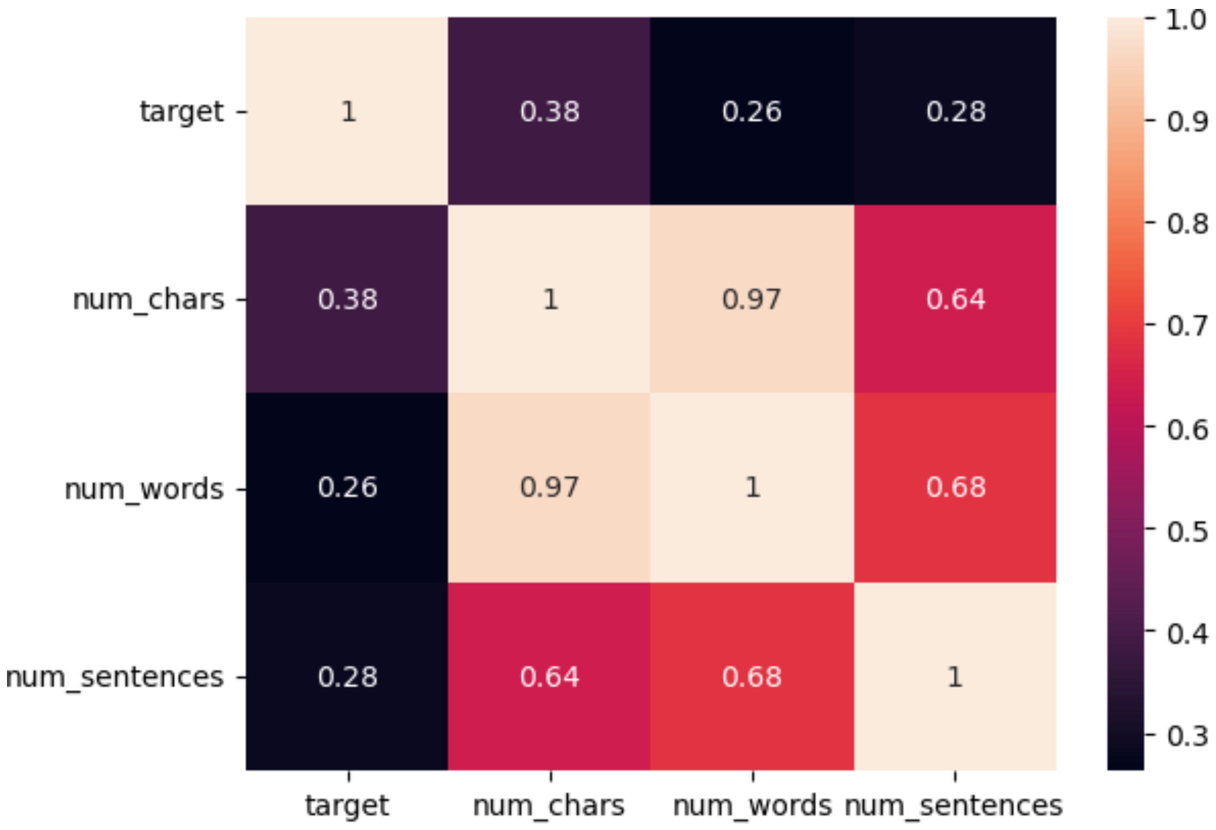
- To gain a deeper understanding, we create a pairplot using `seaborn.pairplot(df, hue='target')`.
- This plot allows us to explore relationships between different features and the target variable (ham or spam).



4. Correlation Analysis:

   1. We generate a heatmap to visualize the correlation between features.

   - Interestingly, the number of characters exhibits a positive correlation (0.38) with the target variable.
   - Based on this correlation, we select the number of characters as a feature for our predictive model.

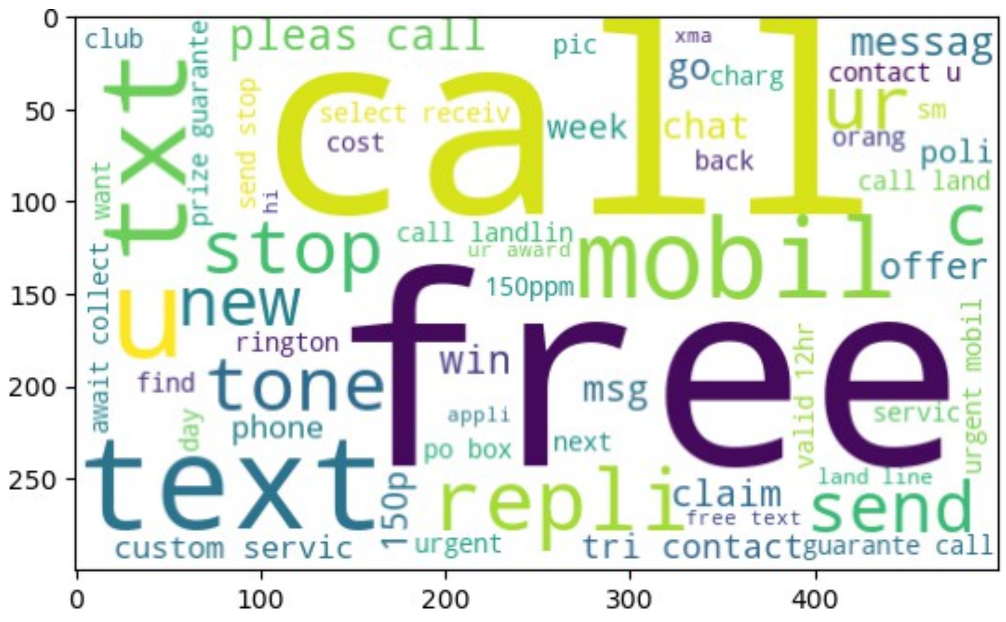


Number of chars for my model since it has a 0.38 correlation with the tar- get variable.

Feature Engineering

Feature engineering plays a crucial role in improving model performance. Some feature engineering techniques we consider include:

1. **Text Preprocessing**: Cleaning and tokenizing the text data by removing stop words, punctuation, and special characters.
2. **TF-IDF Vectorization**: Transforming the text into numerical features using the Term Frequency-Inverse Document Frequency (TF-IDF) technique.
3. **N-grams**: Extracting n-grams (word sequences) to capture contextual information.



Model Selection

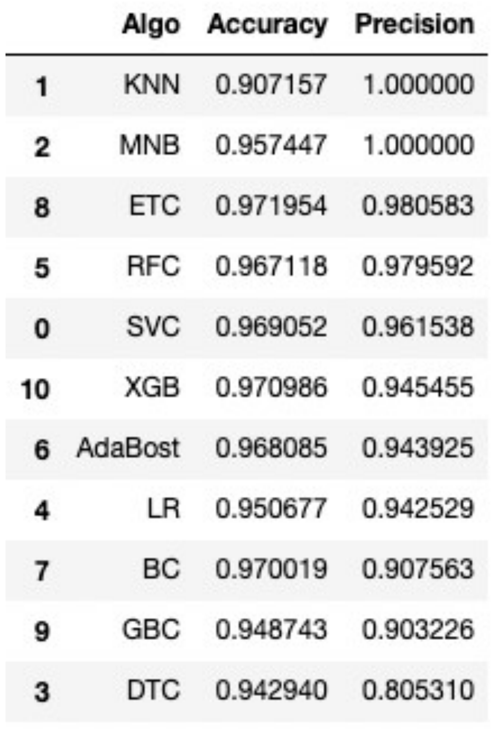
We evaluate various machine learning models for SMS spam detection, including:

1. **Naive Bayes**: A simple yet effective probabilistic model.
2. **Random Forest**: An ensemble method that combines decision trees.
3. **Support Vector Machines (SVM)**: A powerful classifier for high-dimensional data.

Model Evaluation

To assess model performance, we use metrics such as accuracy, precision, recall, and F1-score. Additionally, we perform cross-validation to ensure robustness.

The results were:



	Algo	Accuracy	Precision
1	KNN	0.907157	1.000000
2	MNB	0.957447	1.000000
8	ETC	0.971954	0.980583
5	RFC	0.967118	0.979592
0	SVC	0.969052	0.961538
10	XGB	0.970986	0.945455
6	AdaBost	0.968085	0.943925
4	LR	0.950677	0.942529
7	BC	0.970019	0.907563
9	GBC	0.948743	0.903226
3	DTC	0.942940	0.805310

Pickle files:

```
pickle.dump(tfidf,open('vectorizer.pkl','wb'))

pickle.dump(mnb,open(‘mnbmodel.pkl','wb'))
```

Conclusion

In this report, I present an in-depth analysis of SMS spam detection using machine learning. By leveraging appropriate techniques and models, we aim to build an accurate and efficient spam classifier.