Spam Classification

6830 Project Report -5
Aswani Yaramala | Yagnashree Vellanki

Introduction

Email and SMS spam classification is crucial for filtering out unwanted and potentially harmful messages, saving time, and preventing phishing attacks and malware infections. Our analysis of email and SMS data resulted in the development of a model that can accurately classify messages as spam or not. Furthermore, we identified the most commonly used words in both spam and non-spam messages. This model can be used by business and users to identify spam emails and reduce the risk of spams.

GitHub : https://github.com/aswani848/6830Project5

Presentation : Presentation link

Dataset

For this project, we utilized two datasets that were obtained from Kaggle. The first dataset contained text messages (SMS) with their respective categories indicating whether they were classified as spam or not. The second dataset contained email data that included the email text along with their respective categories, where 0 indicated non-spam and 1 indicated spam.

Analysis Technique

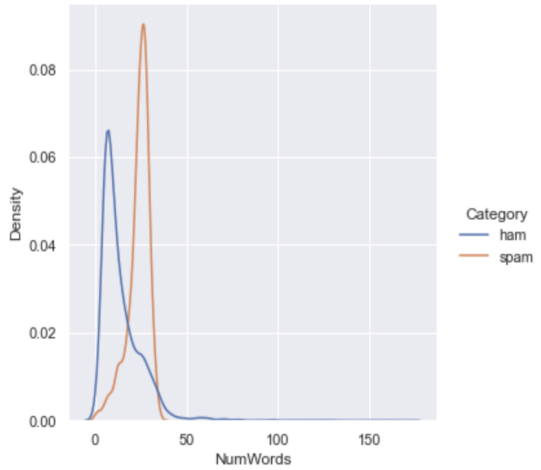
To build a spam classifier model, we chose to use the Multinomial Naive Bayes (MNB) algorithm, which is suitable for text classification due to its ability to handle categorical features. Gaussian Naive Bayes (GNB) works better with continuous data, so we decided against using it for this task. To prepare our data for modelling, we conducted data preprocessing to clean the text messages. Once we had cleaned the data, we converted text to vector and then we partitioned it into training and testing sets using an 80/20 split, where 80% of the data was used for training and 20% for testing. We then assessed the performance of our model using several evaluation metrics, including accuracy, precision, recall, and F1 score. We then used kernel density estimate (KDE) plots to visualise the distribution of the number of words and characters in both spam and non-spam messages and emails. Bar plots were also employed to illustrate the frequent words in both spam and non-spam texts.

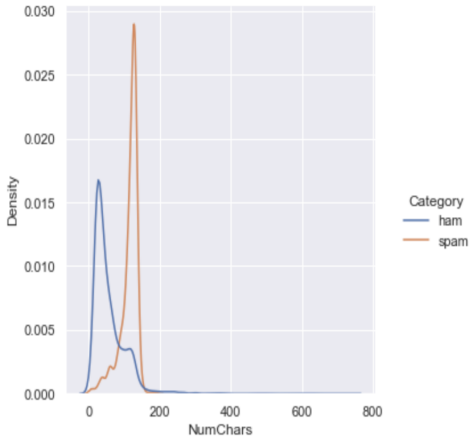
In addition, we used the imshow() function from the matplotlib library to display frequent words in image form. Overall, these visualisation techniques helped us better understand the characteristics of the spam and non-spam messages, which helped us to build an effective spam classifier model.

Results

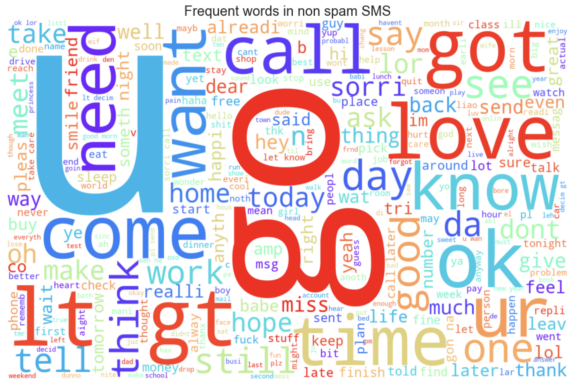
Dataset1: SMS data

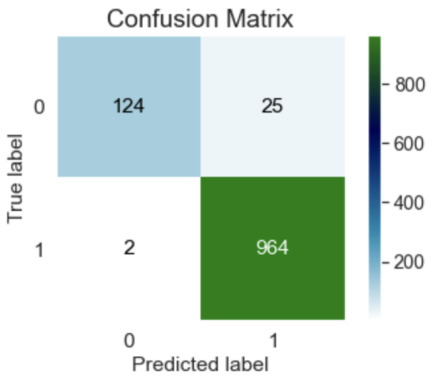
Our initial analysis of the SMS dataset revealed that, on average, spam messages contained more words and characters compared to non-spam messages. This observation makes sense, as typical text messages exchanged between individuals tend to be relatively short, whereas spam messages are often longer and more complex.





During our analysis, we discovered that the most frequently used words in spam SMS messages were "free", "call", and "text". Conversely, the most commonly used words in non-spam SMS messages were "U", "call", "go", "get", "ok", and "love", which all make logical sense in the context of non-spam messages. We created a barplot to display the frequency of words in SMS messages, similar to what we did for emails. However, due to limited space in the report, we didn't include it. Nevertheless, the barplot is still available in our Git repository



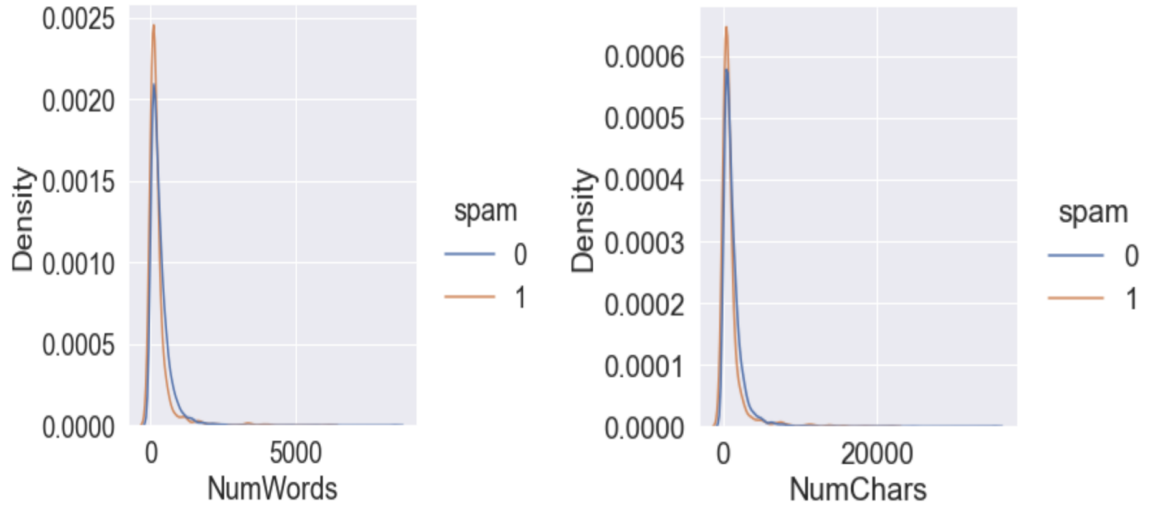


We got below results for our model:

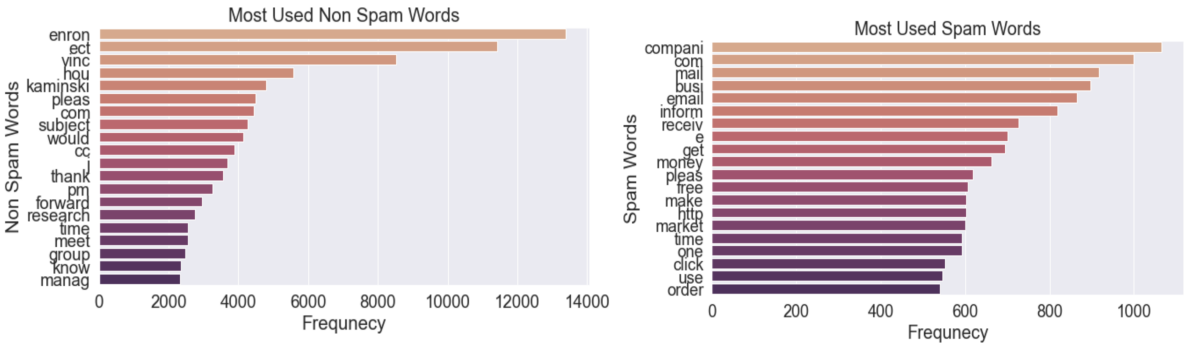
- Accuracy : 0.9757847533632287
- Precision : 0.974721941354904
- Recall : 0.9979296066252588
- f1 : 0.9861892583120205

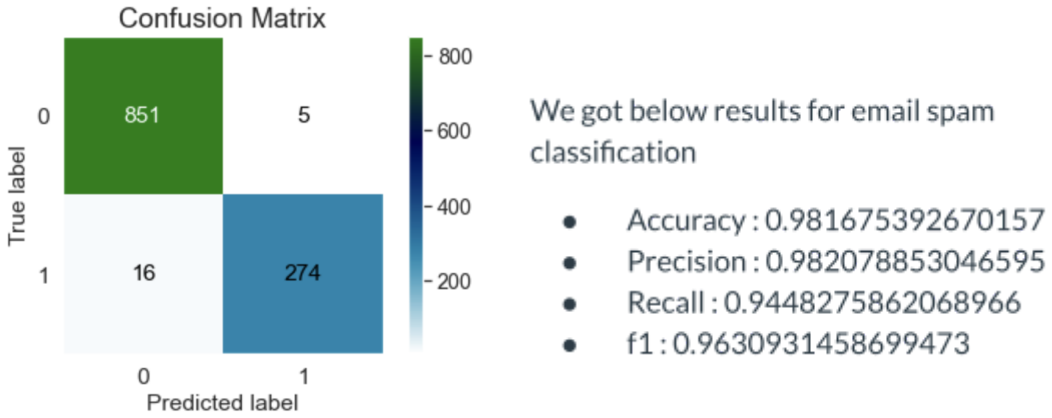
Dataset2: Email data

In the email dataset, we did not observe any significant difference in the number of words or characters between spam and non-spam emails. The distribution of these features appeared to be similar for both types of emails.



Additionally, we displayed the 20 most frequently used words in both spam and non-spam emails. We couldn't make much sense of frequent non spam words. This might be due to the dataset.





Technical

For both datasets, we followed a similar preprocessing procedure. We began by cleaning the raw data, which involved removing URLs, punctuation, and stopwords, as well as converting the text to lowercase and performing stemming. To accomplish these tasks, we used a variety of libraries, including nltk, re (regular expressions), and string.

Once the data was cleaned, we used the TfidfVectorizer() function to convert the text messages into vector form. This allowed us to extract numerical features from the text data, which we then used to train our machine learning model.

Throughout our analysis, we made use of several libraries for data visualisation, including seaborn and matplotlib. In addition, we utilised the WordCloud library to generate images that displayed the most frequently used words in the text data. These images provided a quick and intuitive way to gain insight into the content of the text messages and emails in our datasets.