Building a Spam Email Detection Model: A Step-by-Step Guide

Sudipakoner Follow 6 min read · Jul 31, 2023

Introduction:

———————-

The ever-increasing prevalence of spam emails has become a major nuisance for email users worldwide. Not only do they clutter our inboxes, but they also pose security risks and potentially harmful consequences. In this blog, we will explore the process of building a powerful spam email detection model using machine learning techniques. We will delve into data cleaning, exploratory data analysis, text preprocessing, model building, evaluation, and even model improvement. So, let's embark on our journey towards creating an effective spam email filter.

Spam word WordCloud

Dataset used:

— — — — — — — — — -

Email Spam Classification Dataset CSV

CSV file containing spam/not spam information about 5172 emails.

www.kaggle.com

Complete Project Github link :

https://github.com/SUDIPA9002/Spam-Email-detection

1. **Data Cleaning:**

   — — — — — — — — — — -

   To begin, we load our dataset from a CSV file and perform data cleaning

to ensure the dataset is suitable for analysis. The unnecessary columns are dropped, and the target variable is encoded to facilitate further processing. Duplicate and missing values, if any, are also handled, ensuring that we have a clean dataset to work with.

```
import numpy as np
import pandas as pd
data = pd.read_csv('Spam Email Detection - spam.csv')
```

```
### 1. Data Cleaning
data.info()
data.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace = True) #drop
data.shape
data.sample(2)
data.rename(columns={'v1':'Target','v2':'Mail-Text'},inplace=True)
data.sample(2)
```

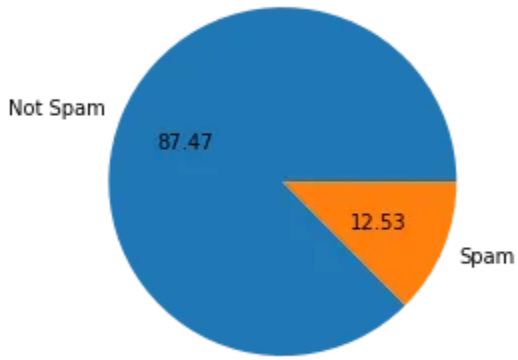
```
# Encoding data
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data['Target'] = encoder.fit_transform(data['Target'])
data.tail(2)

# check for missing values
data.isna().sum()

# check for duplicate values
data.duplicated().sum()

# drop duplicate values
data = data.drop_duplicates(keep = 'first')
data.duplicated().sum()
data.shape
data.sample(4)
```

2. Exploratory Data Analysis (EDA):

EDA is essential for understanding the dataset's characteristics and gaining insights into the distribution of spam and non-spam emails. Visualizations like pie charts, histograms, and pair plots are created to analyze the character count, word count, and sentence count of both spam and non-spam emails. This analysis helps us understand the data and potentially identify patterns that might assist in model training.

```
## 2. Data Visualization
data['Target'].value_counts()
data.rename(columns={'Target':'Type'},inplace=True)
data['Type'].value_counts()
```

```
# pie chart
import matplotlib.pyplot as plt
plt.pie(data['Type'].value_counts(),labels=['Not Spam','Spam'],autopct="%0.2f")
plt.show()
```

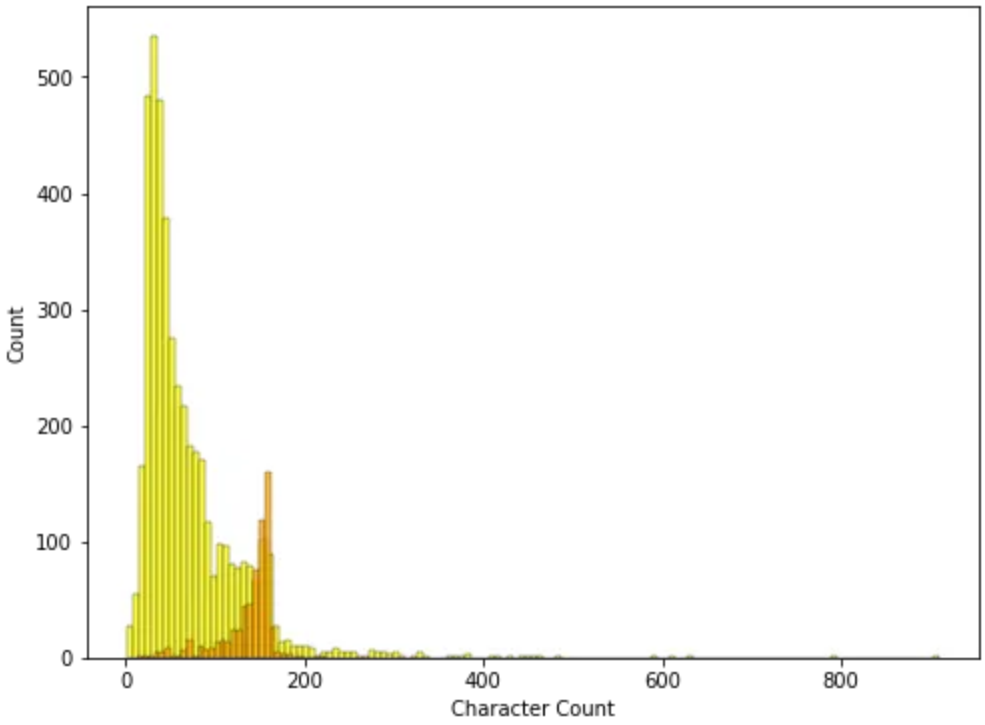


Pie chart

Data is not in balanced format.Now we have to convert data into a balanced format...

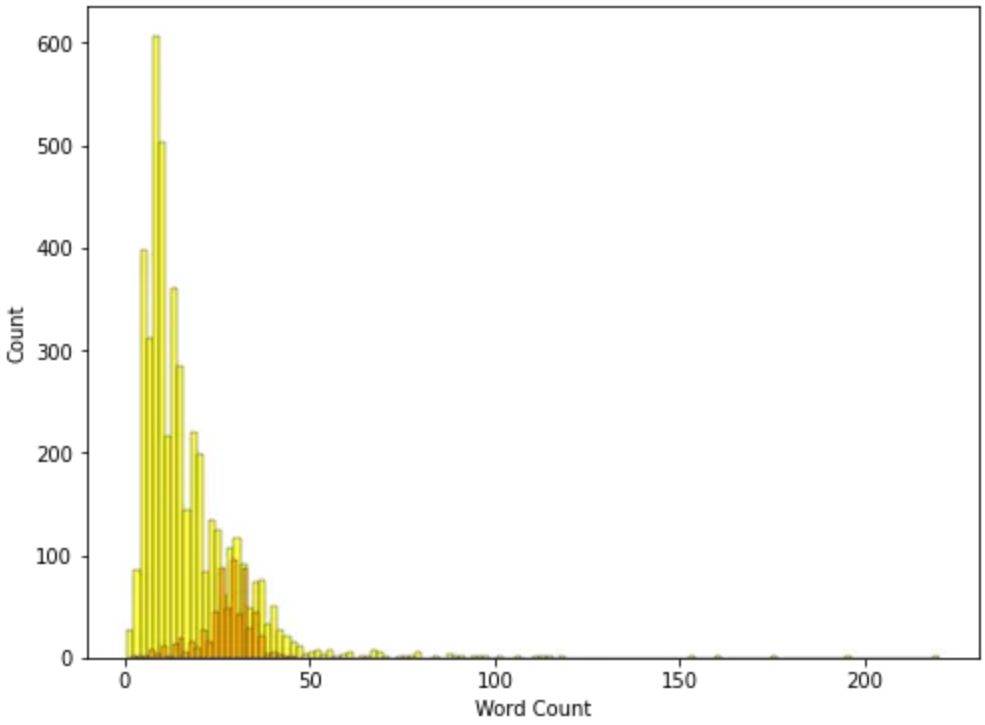
```
import nltk
## if nltk is not present then run this command - pip install nltk
nltk.download('punkt')
data['num_characters']=data['Mail-Text'].apply(len) # number of characters in a
data.rename(columns={'num_characters':'Character Count'},inplace=True)
data.head(3)
data['Mail-Text'].apply(lambda x:nltk.word_tokenize(x)) ## Breaking sentence int
data['Mail-Text'].apply(lambda x:len(nltk.word_tokenize(x))) ## Counting number
data['Word Count'] = data['Mail-Text'].apply(lambda x:len(nltk.word_tokenize(x))
data.head(4)
data['Sentence Count']=data['Mail-Text'].apply(lambda x:len(nltk.sent_tokenize(x
data.tail(4)
data.describe()
# Describe function for Not Spam Messages
data[data['Type'] == 0][['Character Count','Word Count','Sentence Count']].descr

# Describe function for Spam Messages
data[data['Type'] == 1][['Character Count','Word Count','Sentence Count']].descr
```

```
## if seaborn is not present then run this command-pip install seaborn
import seaborn as sns

## Histogram plot
plt.figure(figsize=(8,6))
sns.histplot(data[data['Type']==0]['Character Count'],color = 'yellow')
sns.histplot(data[data['Type']==1]['Character Count'],color = 'orange')
```



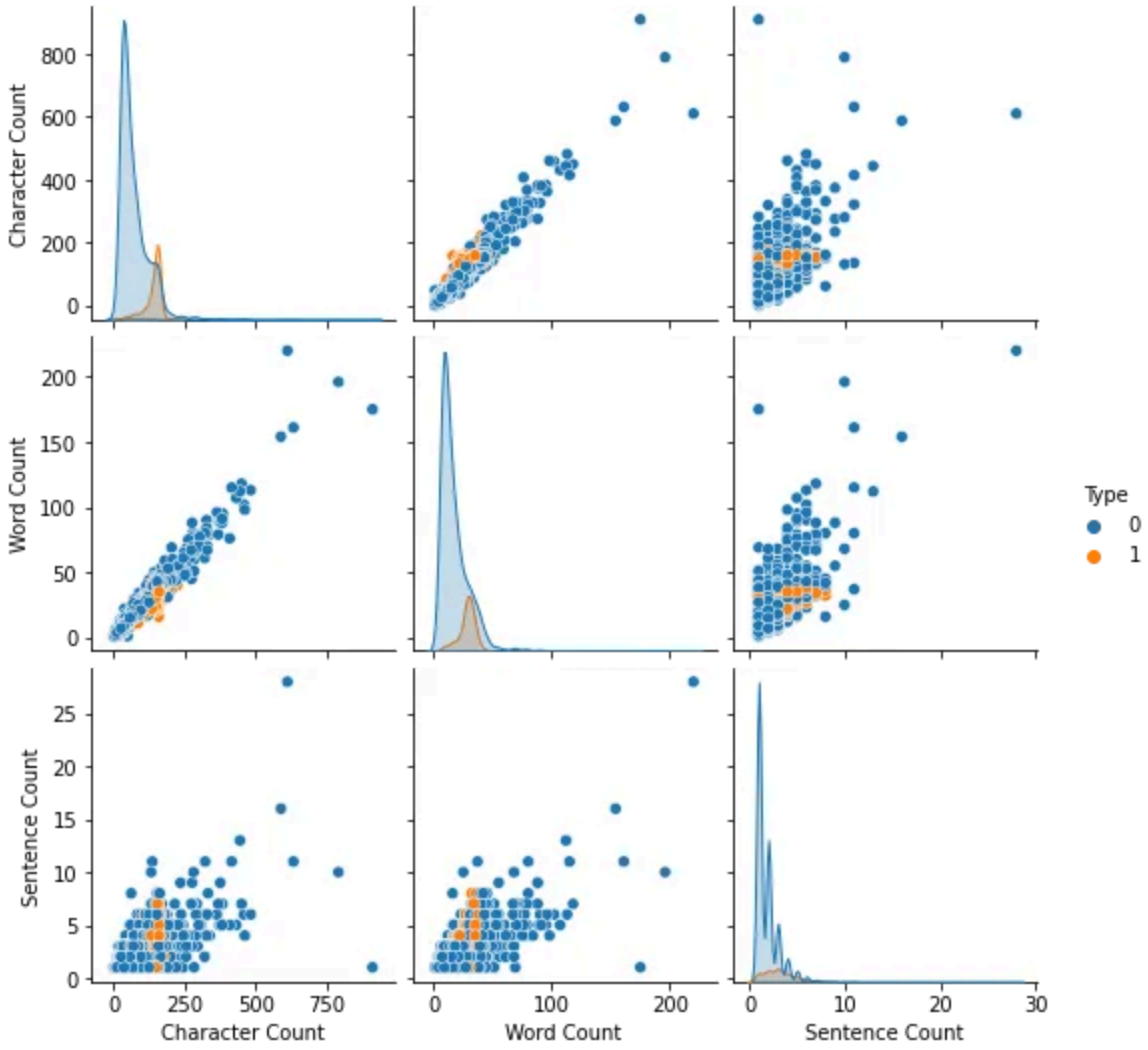
Histogram Plot 1

```
plt.figure(figsize=(8,6))
sns.histplot(data[data['Type']==0]['Word Count'],color = 'yellow')
sns.histplot(data[data['Type']==1]['Word Count'],color = 'orange')
```



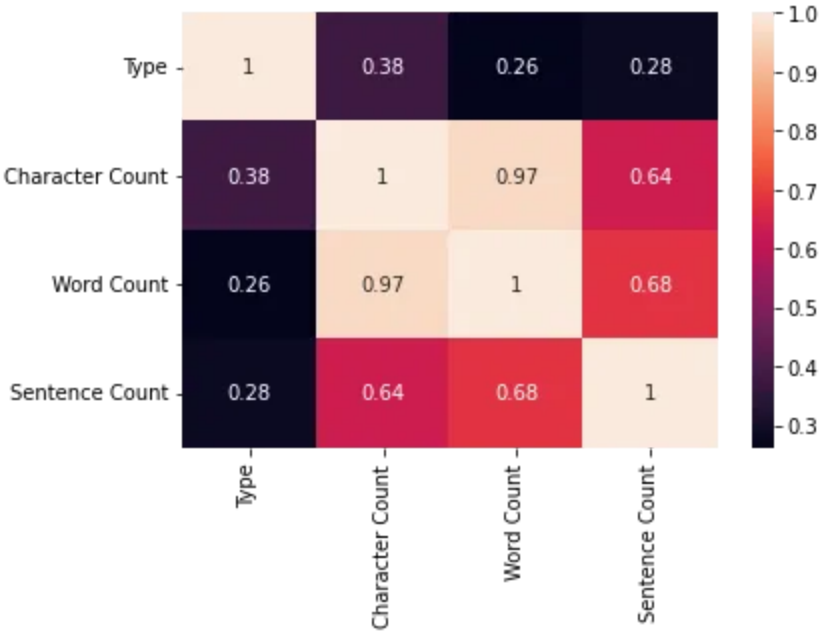
Histogram Plot 2

```
sns.pairplot(data,hue='Type')
```



Pairplot

```
columns_to_select = ['Type', 'Character Count', 'Word Count', 'Sentence Count']
numeric_data = data[columns_to_select]

## Heat map plot
sns.heatmap(numeric_data.corr(),annot=True)
```



Heatmap

3. Text Preprocessing:

— — — — — — — — — — — — — — — -

Text preprocessing is a critical step before feeding the data into our machine learning models. We convert the text to lowercase, tokenize the sentences, remove special characters, stopwords, and punctuation. Additionally, we apply stemming to reduce words to their root form. These steps ensure that the text data is in a consistent and standardized format for accurate model training.

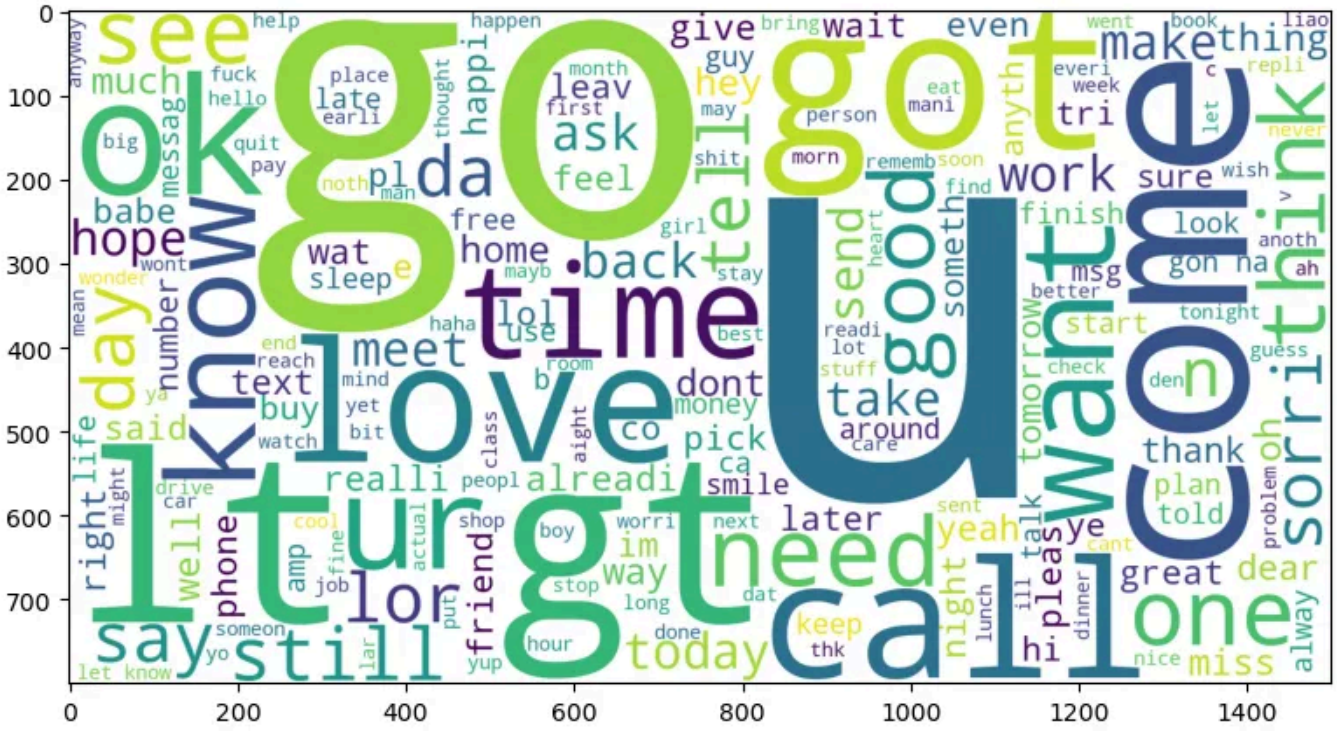
```
nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords.words('english')
import string
string.punctuation
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
ps.stem('Roaming')
```

```
def transform_text(text):
    text = text.lower() # 1. Lower Case Convertion
    text = nltk.word_tokenize(text) # 2. Tokenization

    y=[] # 3. Removing special Characters
    for i in text:
        if i.isalnum():
            y.append(i)

    text = y[:]
    y.clear()

    for i in text: # 4. Removing stop words and punctuation
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:  # 5. Stemmimg
        y.append(ps.stem(i))

    return " ".join(y)
```

```
transform_text('Okay name ur price as long as its legal! Wen can I pick them up?
data['Mail-Text'][100]
data['Transformed-Text']=data['Mail-Text'].apply(transform_text)
```

```
#### Word Cloud Formation:
pip install wordcloud
from wordcloud import WordCloud
wc = WordCloud(width=1500,height=800,min_font_size=10,background_color='white')
spam_wc = wc.generate(data[data['Type']==1]['Transformed-Text'].str.cat(sep=" ")
plt.figure(figsize=(20,10))
plt.imshow(spam_wc)
```

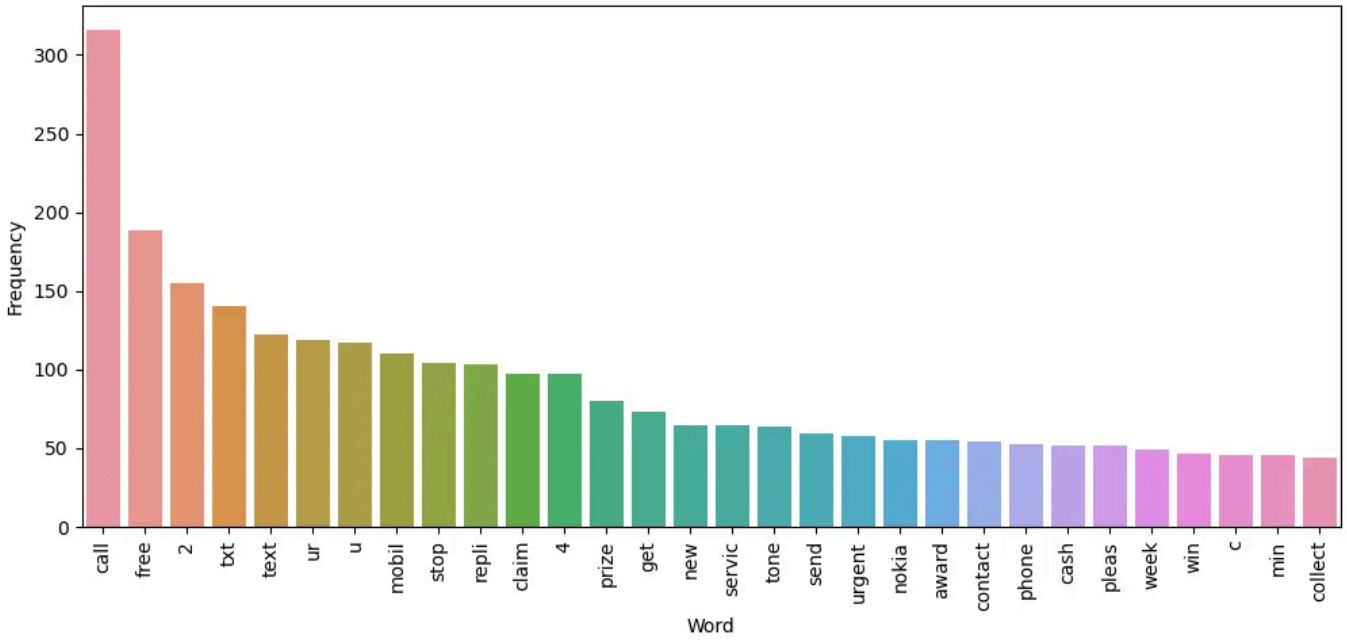

```
non_spam_wc = wc.generate(data[data['Type']==0]['Transformed-Text'].str.cat(sep=
plt.figure(figsize=(12,5))
plt.imshow(non_spam_wc)
```



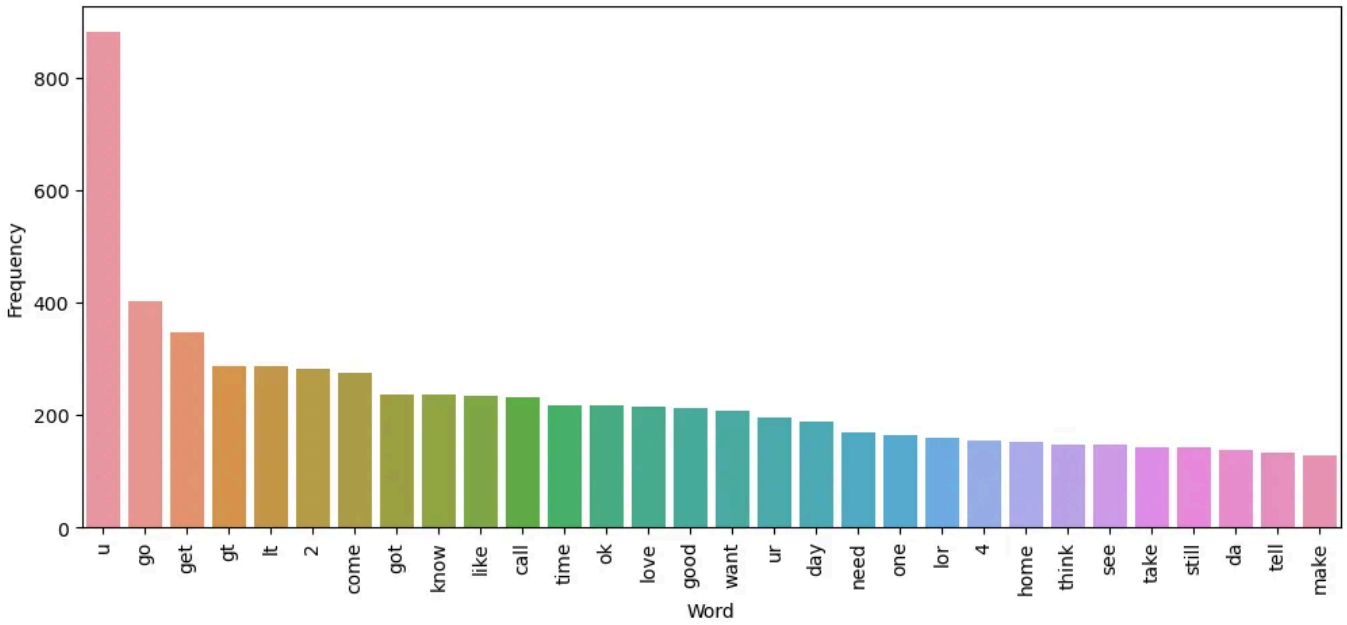
Non-spam word cloud

```
spam_word = []
for msg in data[data['Type']==1]['Transformed-Text'].tolist():
    for word in msg.split():
        spam_word.append(word)
len(spam_word)
from collections import Counter
Counter(spam_word)
pd.DataFrame(Counter(spam_word).most_common(30))
```

```
data_frame = pd.DataFrame(Counter(spam_word).most_common(30), columns=['Word', '
plt.figure(figsize=(12, 5))
sns.barplot(x='Word', y='Frequency', data=data_frame)
plt.xticks(rotation='vertical')
plt.show()
```




```
non_spam_word = []
for msg in data[data['Type']==0]['Transformed-Text'].tolist():
    for word in msg.split():
        non_spam_word.append(word)
len(non_spam_word)
data_frame = pd.DataFrame(Counter(non_spam_word).most_common(30), columns=['Word
plt.figure(figsize=(12, 5))
sns.barplot(x='Word', y='Frequency', data=data_frame)
plt.xticks(rotation='vertical')
plt.show()
```



4. Model Building:

— — — — — — — — — — — — — — -

Next, we extract features using two different approaches: CountVectorizer and TfidfVectorizer. Subsequently, we train various machine learning models, such as Gaussian Naive Bayes, Multinomial Naive Bayes, Logistic Regression, Support Vector Machine (SVM), Decision Tree, K-Nearest Neighbors (KNN), Random Forest, and others. We evaluate the models using accuracy and precision scores to select the best-performing model.

5. Model Improvement:

— — — — — — — — — — — — — — — — — — —

To further enhance the model's performance, we explore model improvement techniques. We experiment with changing the max_feature parameter of TfidfVectorizer and include additional features like character count in the dataset. By doing so, we aim to optimize the model and achieve better results.

Algorithm	Accuracy	Precision	Accuracy_on_Scaling	Precision_on_Scaling	Accuracy_max_ft_3000	Precision_max_ft_3000
MultinomialNB	0.959342	1.000000	0.940949	0.983871	0.940949	0.983871
KNN	0.908035	1.000000	0.934172	0.757282	0.934172	0.757282
Extra Tree	0.973863	1.000000	0.982575	0.981308	0.982575	0.981308
RF	0.969022	0.978495	0.978703	0.971429	0.978703	0.971429
SVC	0.974831	0.970297	0.882865	0.000000	0.882865	0.000000
Logistic Regression	0.952565	0.939024	0.955470	0.886598	0.955470	0.886598
GradientBoosting	0.950629	0.926829	0.956438	0.895833	0.956438	0.895833
XGBoost	0.965150	0.912621	0.969990	0.916667	0.969990	0.916667
AdaBoost	0.954501	0.885417	0.967086	0.914286	0.967086	0.914286
Bagging	0.958374	0.842105	0.964182	0.850000	0.964182	0.850000
ecisionTreeClassifier	0.936108	0.823529	0.948693	0.840000	0.948693	0.840000

6. Model Evaluation:

— — — — — — — — — — — — — — -

Once the models are trained, we evaluate their performance based on accuracy and precision scores. These metrics help us determine how well the model is classifying emails into spam and non-spam categories. The model with the highest accuracy and precision is chosen as the final selected model for deployment.

MultinomialNB is selected , as precision score is the highest.

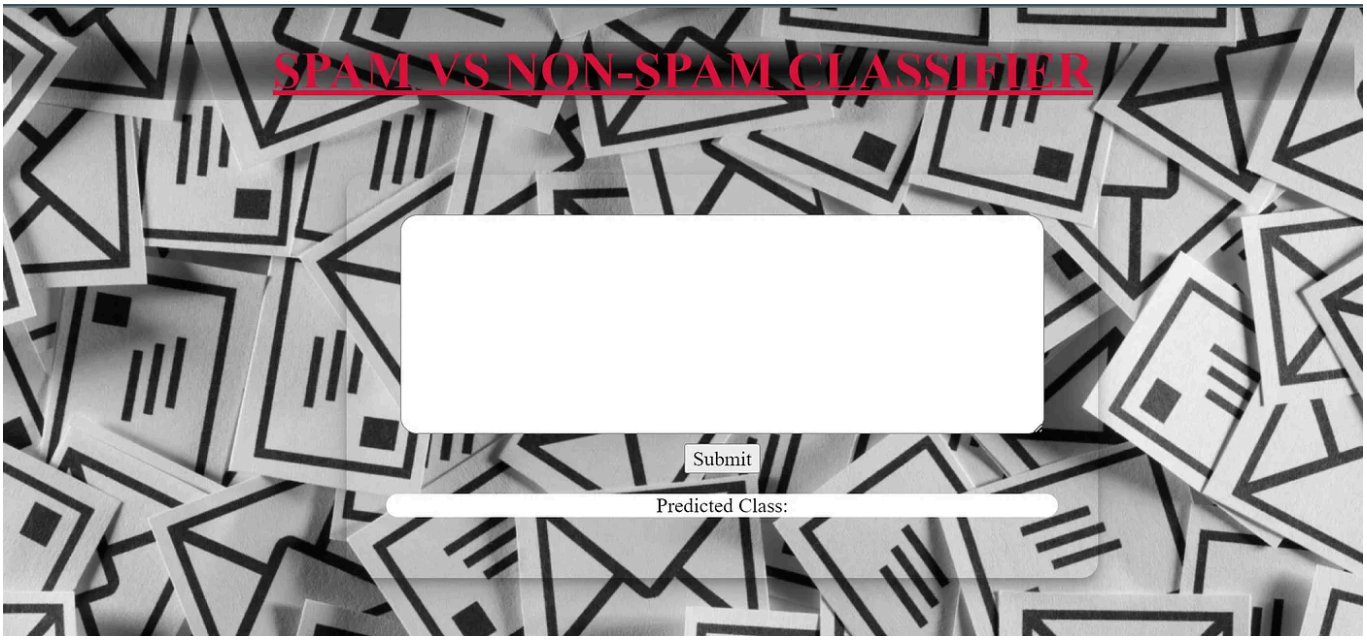
```
mnb.fit(X_train,y_train)
y_pred2 = mnb.predict(X_test)
print(accuracy_score(y_test,y_pred2))
print(confusion_matrix(y_test,y_pred2))
print(precision_score(y_test,y_pred2))
```

```
0.9593417231364957
[[912   0]
 [ 42  79]]
1.0
```

Deployment:

— — — — — — — -

Finally, the selected model is saved using the `pickle` library, and we create a user-friendly Flask web application for spam email detection. Users can input an email text, and the web application will predict whether the email is spam or not.



Conclusion:

— — — — — — — —

Building a spam email detection model involves various essential steps, from data cleaning and exploratory data analysis to text preprocessing, model building, evaluation, and improvement. By following this step-by-step guide, you can create an effective spam email filter that can significantly enhance your email experience, keeping your inbox clutter-free and secure from potential threats.

As technology continues to advance, we can expect even more innovative solutions to make our digital experiences safer and more enjoyable. So, let's embrace the possibilities and continue our journey into the exciting world of machine learning!