SMS SPAM DETECTION

A MINI PROJECT
REPORT

Submitted by

SANTHA LAKSHMI S (211521106138)

PRETHI A (211521106125)

ELECTRONICS AND COMMUNICATION ENGINEERING

PANIMALAR INSTITUTE OF TECHNOLOGY, CHENNAI-600 123

ANNA UNIVERSITY : CHENNAI 600 025

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING - CS3491

MINI PROJECT

PROJECT TITLE:

SMS SPAM DETECTION

DONE BY

1. SANTHA LAKSHMI S - 2021PITEC154
2. PRETHI A - 2021PITEC148

TABLE OF CONTENTS

ABSTRACT :

This AIML project aims to develop a spam detection system using the Multinomial Naive Bayes classifier. The system preprocesses SMS messages by cleaning special characters, converting to lowercase, tokenizing, removing stopwords, and stemming. Leveraging Countvectorizer, the text data is transformed into a matrix of token counts for feature extraction. The dataset undergoes a train-test split for model evaluation. A simple graphical user interface (GUI) is constructed using tkinter, enabling users to interactively input messages for spam prediction. The system achieves effective spam classification, providing a seamless user experience while addressing the pervasive issue of unsolicited messages.

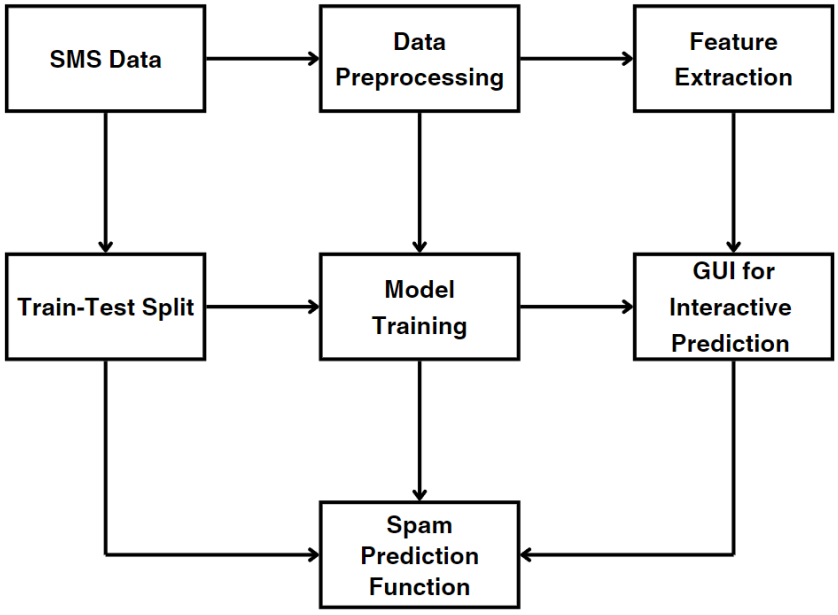
INTRODUCTION:

In today's digital age, the proliferation of spam messages poses a significant challenge to effective communication and information retrieval. This project addresses this issue by developing a spam detection system using the Multinomial Naive Bayes classifier. Leveraging natural language processing techniques, the system preprocesses SMS messages to extract meaningful features and train a robust classification model. The implementation includes a user-friendly graphical interface for real-time spam prediction. By employing machine learning algorithms and interactive technology, this project aims to enhance user experience and mitigate the impact of spam, contributing to a more secure and efficient communication environment.

PROBLEM STATEMENT:

Despite advancements in communication technology, the persistent influx of spam messages continues to undermine user experience and information security. Existing spam detection methods often lack efficiency and user-friendliness, leaving users vulnerable to unsolicited content. Addressing this challenge, this project seeks to develop a novel spam detection system utilizing the Multinomial Naive Bayes classifier. By integrating natural language processing techniques and interactive technology, the system aims to provide a seamless user experience while effectively identifying and filtering spam messages. Through this initiative, we endeavor to enhance communication reliability, protect user privacy, and mitigate the negative impact of spam on digital ecosystems.

BLOCK DIAGRAM:



PROJECT EXPLANATION (ALGORITHM):

The core algorithm driving our SMS spam detection system is the Multinomial Naive Bayes (MNB) classifier. Here's a detailed breakdown of how it operates within the context of our project:

Multinomial Naive Bayes (MNB) classifier:

The Multinomial Naive Bayes (MNB) classifier is a popular algorithm used in machine learning for text classification tasks, such as spam detection. Here's a detailed explanation of how it works and its role in our SMS spam detection project:

1. **Probability-Based Classification:**
   - The MNB classifier is based on Bayes' theorem, a fundamental concept in probability theory. It calculates the probability of a message belonging to a particular category (spam or ham) given its features (words).
   - In our project, the features are the word frequencies extracted from the SMS messages using techniques like CountVectorizer.
2. **Assumption of Conditional Independence:**
   - The "Naive" in Naive Bayes refers to the assumption of conditional independence among the features given the class label. In simpler terms, it assumes that the presence of a particular word in a message is independent of the presence of other words, given whether the message is spam or ham.

- Despite this simplifying assumption, Naive Bayes classifiers often perform well in practice, especially for text classification tasks.

3. **Multinomial Distribution:**
   - The "Multinomial" part of MNB indicates that it's specifically designed for data with multiple features, where each feature represents the frequency of a particular word in the text.
   - In our project, each SMS message is represented as a vector of word counts, and the classifier learns the probability distribution of word frequencies for both spam and ham messages.
4. **Model Training:**
   - During the training phase, the MNB classifier learns the probability distributions of word frequencies associated with spam and ham messages from the labeled training data.
   - It calculates the likelihood of observing each word given the class label (spam or ham) and also estimates the prior probability of each class.
5. **Spam Prediction:**
   - When presented with a new, unseen SMS message, the trained classifier calculates the probability of it belonging to each class (spam or ham) based on the word frequencies in the message.
   - It combines these probabilities using Bayes' theorem to determine the most likely class for the message, i.e., whether it's spam or ham.
6. **Evaluation and Performance:**
   - The performance of the MNB classifier is typically evaluated using metrics such as accuracy, precision, recall, and F1-score on a separate test dataset.
   - In our project, we use these metrics to assess how well the classifier can accurately classify SMS messages as spam or ham.

In summary, the Multinomial Naive Bayes classifier is a powerful and efficient algorithm for text classification tasks like spam detection. Its probabilistic approach, combined with the simplicity of implementation, makes it a popular choice for handling textual data in machine learning applications.

Program Explanation:

1. **Data Preprocessing:**
   Initially, we take the raw SMS messages and preprocess them to ensure consistency and remove unnecessary noise. This involves several steps:
   a. Converting all text to lowercase: This ensures that words like "Hello" and "hello" are treated as the same word.

b. Removing special characters: We eliminate characters like punctuation marks or symbols that don't contribute to the meaning of the message.
c. Eliminating common stopwords: Stopwords are common words like "the," "is," and "and" that occur frequently but typically don't carry much meaning. Removing them helps to focus on the important words in the messages.

2. **Feature Extraction:**
   With the text cleaned up, we use a technique called CountVectorizer to transform the messages into numerical features that our algorithm can understand. CountVectorizer essentially creates a tally of how often each word appears in each message. This creates a numerical representation of the messages, where each word becomes a feature, and its frequency becomes its value.
3. **Model Training:**
   Now that we have our numerical features, we can train our Multinomial Naive Bayes classifier. This involves feeding it a large set of labeled SMS messages, where each message is tagged as either spam or not spam (ham). The classifier learns from these examples by analyzing the frequency distributions of words associated with each category. It essentially builds a probabilistic model of what spam messages and non-spam messages look like based on the words they contain.
4. **Spam Prediction:**
   Once the classifier is trained, it's ready to make predictions on new, unseen SMS messages. When a new message comes in, the classifier calculates the likelihood that it belongs to each category (spam or ham) based on the words it contains. It compares these likelihoods and assigns the message to the category with the highest probability. This decision is made using Bayes' theorem, which calculates the probability of a message being spam or ham given its word frequencies.
5. **GUI Interaction:**
   To make our system user-friendly, we implement a graphical user interface (GUI). This interface allows users to input messages and receive instant predictions on whether they're spam or not. It provides a seamless and intuitive way for users to interact with our spam detection system.

In summary, the Multinomial Naive Bayes algorithm, coupled with effective data preprocessing and feature extraction techniques, forms the backbone of our SMS spam detection system. Its simplicity, efficiency, and effectiveness make it well-suited for handling the high-dimensional and sparse nature of text data, ultimately enabling us to build a robust and user-friendly solution for combating spam.

CODE :

1. Data Preprocessing:

- Software: Python (programming language), NLTK library for natural language processing tasks.
- **Code:**

```
import numpy as np
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
def preprocess_message(message):
    message = re.sub(pattern='[^a-zA-Z]', repl=' ', string=message)
    message = message.lower()
    words = message.split()
    words = [word for word in words if word not in
    set(stopwords.words('english'))]
    ps = PorterStemmer()
    words = [ps.stem(word) for word in words]
    return ' '.join(words)
def predict_spam(message):
    message = preprocess_message(message)
    temp = cv.transform([message]).toarray()
        return classifier.predict(temp)
```

Explanation:

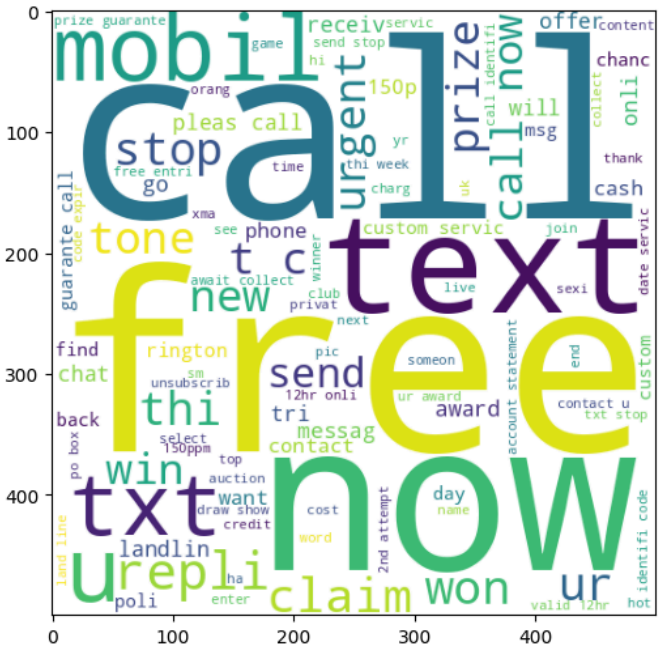
The imported libraries include numpy and pandas for data manipulation, re for regular expression operations, and nltk for natural language processing tasks. Specifically, stopwords and PorterStemmer are imported from nltk.corpus to eliminate common words and perform word stemming, respectively.

The preprocessing function, named preprocess_message, is designed to refine a given message for text analysis purposes. Initially, it employs re.sub to strip away non-alphabetic characters from the message. Subsequently, the message is converted to lowercase using message.lower(). The function then splits the message into individual words using split(). By utilizing stopwords.words('english'), common words such as 'the', 'is', and 'and' are eliminated from the message. Finally, the Porter stemming algorithm is applied to the remaining words to further streamline the text for analysis.

SPAM related word

```
plt.figure(figsize=(15,6))
plt.imshow(spam_wc)
```

```
<matplotlib.image.AxesImage at 0x7964d1120370>
```



HAM related words

```
plt.figure(figsize=(15,6))
plt.imshow(ham_wc)
```

```
<matplotlib.image.AxesImage at 0x7964e1c15b40>
```

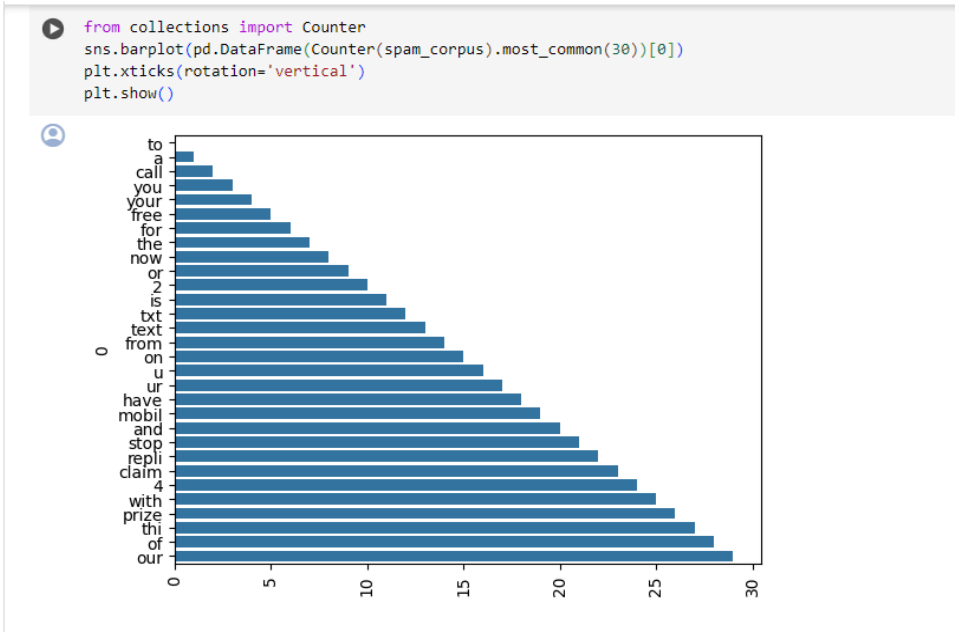
2. Data Cleaning:

- Software: Python
- **Code:**

```
sms = pd.read_csv('Spam SMS Collection.txt', sep='\t', names=['label','message'])
sms.drop_duplicates(inplace=True)
sms.reset_index(drop=True, inplace=True)
sms['message'] = sms['message'].apply(preprocess_message)
```

Collection of Stop words :

```
from collections import Counter
sns.barplot(pd.DataFrame(Counter(spam_corpus).most_common(30))[0])
plt.xticks(rotation='vertical')
plt.show()
```



Explanation:

The Spam Prediction Function (predict_spam) is a tool that preprocesses text messages and uses a pre-trained classifier to predict if a message is spam or not. The function transforms the message into a numerical format, converts it into a sparse matrix and dense array, and then predicts the class label using a pre-trained classifier.

pd.read_csv('Spam SMS Collection.txt', sep='\t', names=['label','message']): This line reads the dataset from a text file named 'Spam SMS Collection.txt'. The data is assumed to be tab-separated (sep='\t'), and the column names are explicitly defined as 'label' and 'message'. This creates a DataFrame named sms.

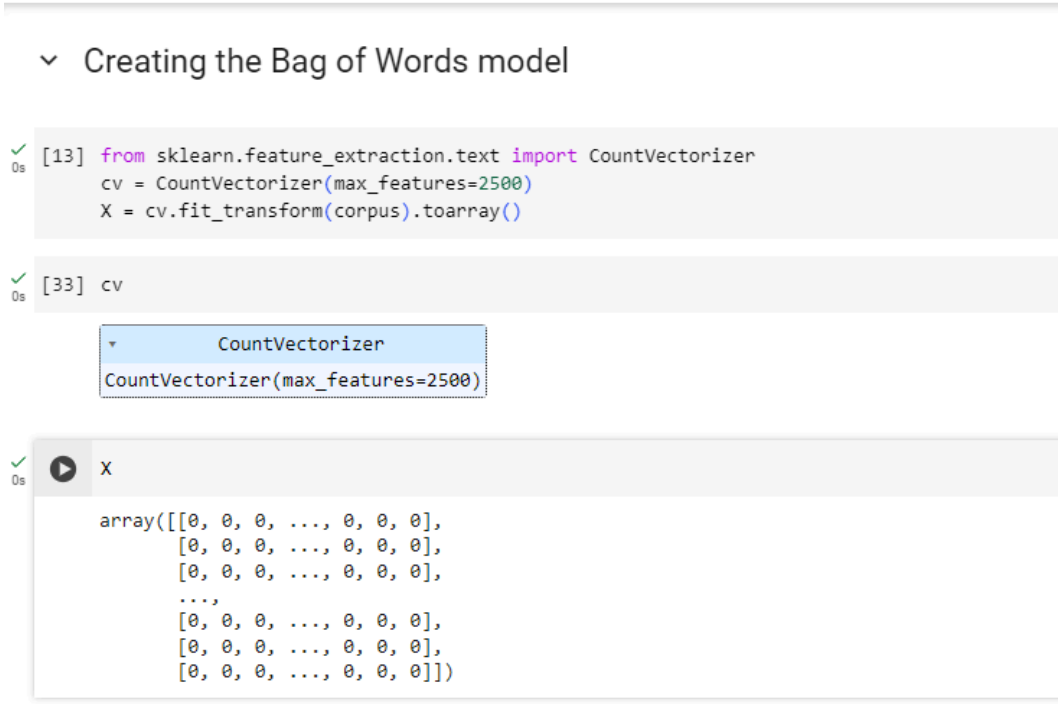
Sms.drop_duplicates(inplace=True) prepares the SMS dataset by removing duplicates and preprocessing text messages, making it suitable for machine learning models and text analysis tasks. It removes duplicate rows from the DataFrame sms based on all columns and resets the index. The drop=True parameter ensures that the old index is not added as a column in the DataFrame. The apply method efficiently applies a function to each element of a DataFrame column, resulting in preprocessed text messages for further analysis.

3. Feature Extraction:

- Software: Python, scikit-learn library for machine learning tasks.
- **Code:**

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=2500)
X = cv.fit_transform(sms['message']).toarray()
y = pd.get_dummies(sms['label'])['spam'].values
```

Explanation:



Count Vectorizer is a term frequency-based method in natural language processing, often used for converting a collection of text documents into a matrix of token counts. It essentially creates a vocabulary of all unique words (or tokens) in the documents and counts the frequency of each word in each document. In Python, it's commonly implemented as part of the scikit-learn library, which is a popular machine learning library in Python . The output shown below.

4. Model Training:

- Software: Python, scikit-learn.
- **Code:**

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
classifier = MultinomialNB(alpha=0.1)
classifier.fit(X_train, y_train)
```

Explanation:

The Multinomial Naive Bayes classifier is a probabilistic classifier based on Bayes' theorem. It is imported from scikit-learn and splits the dataset into training and testing subsets. The accuracy_score function is used to evaluate the classifier's performance.

The dataset is split into training and testing sets using the train_test_split function. The random_state parameter ensures reproducibility by fixing the random seed. The classifier is initialized using the Multinomial Naive Bayes classifier object, initialized with the Laplace smoothing parameter alpha=0.1. The classifier is trained using the training data by estimating the probabilities of different classes based on the frequency of features in the training set. The classifier is ready to make predictions on new data and its performance can be evaluated using metrics like accuracy, which can be computed using the testing set and the accuracy_score function.

5. GUI Development:

- Software: Python, tkinter library for building graphical user interfaces.
- **Code:**

```
import tkinter as tk
from tkinter import messagebox
root = tk.Tk()
root.title("Spam Detection")
root.geometry("400x200")
label = tk.Label(root, text="Enter the Message:")
label.pack()
entry = tk.Entry(root, width=50)
entry.pack()
predict_button = tk.Button(root, text="Predict", command=predict)
```

```
predict_button.pack()
print("Spam Prediction is ongoing...")
root.mainloop()
print("Spam Prediction is Ended")
```

Explanation:

The Tkinter library is a standard GUI toolkit for Python, which is used to create a GUI window. The main window is created using the root variable, which is set to "Spam Detection" and has dimensions of 400x200 pixels. Widgets are added to the GUI using the label widget, entry widget, and predict_button widgets. The label widget is placed inside the root window and packed into the GUI window. The entry widget is a text box where the user can input a message, with a width of 50 characters. The predict button is created and packed into the GUI window.

The Tkinter event loop is started by starting the root.mainloop() function, which listens for user interactions with the GUI. The program continues until the user closes the GUI window. The message "Spam Prediction is ongoing..." is printed, indicating that spam prediction is ongoing. The program ends when the user closes the GUI window and the mainloop() function exits.

6. Spam Prediction:

- Software: Python.
- **Code:**

```
def predict_spam(message):
    message = preprocess_message(message)
    temp = cv.transform([message]).toarray()
    return classifier.predict(temp)
```

Explanation:

The function predict_spam is defined as a method that uses a single argument message to determine if a text message is spam or not. The message is preprocessed using the preprocess_message function, which removes special characters, lowercase, stopwords, and stems words. The message is then transformed into a numerical format suitable for machine learning algorithms using the CountVectorizer object. The transformed message is then used to predict whether the input message is spam or not. The classifier's predict method is applied to the transformed message to obtain the predicted class label. The function is a useful tool for detecting spam in text messages.

TOTAL CODE:

```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(),labels=['ham','spam'],autopct="%0.2f")
plt.show()
```

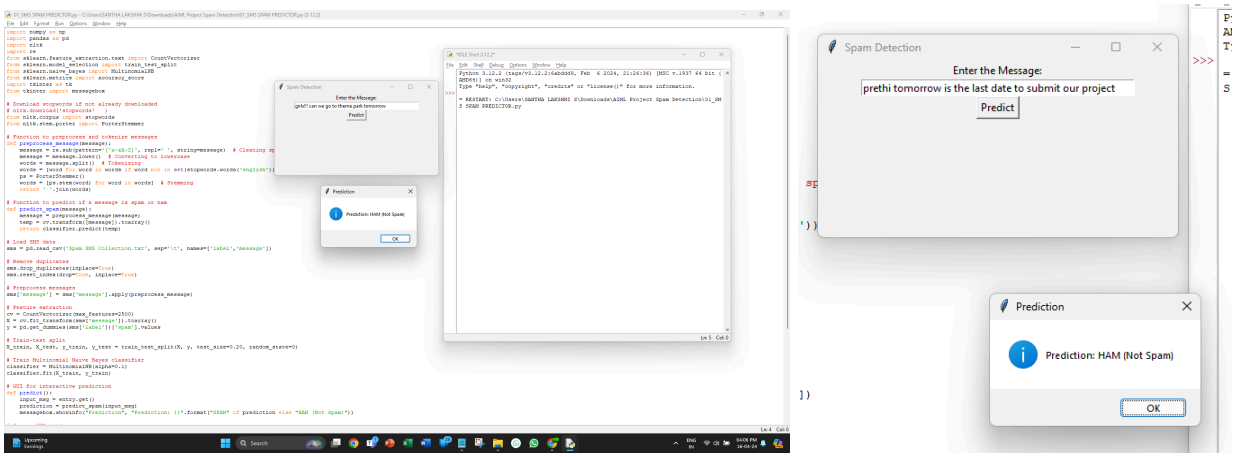
```
import numpy as np
import pandas as pd
import nltk
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
import tkinter as tk
from tkinter import messagebox
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(),labels=['ham','spam'],autopct="%0.2f")
plt.show()

# Download stopwords if not already downloaded
# nltk.download('stopwords'   )
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

# Function to preprocess and tokenize messages
def preprocess_message(message):
        message = re.sub(pattern='[^a-zA-Z]', repl=' ', string=message)  # Cleaning special characters
        message = message.lower()  # Converting to lowercase
        words = message.split()  # Tokenizing
        words = [word for word in words if word not in set(stopwords.words('english'))]  # Removing stopwords
        ps = PorterStemmer()
        words = [ps.stem(word) for word in words]  # Stemming
        return ' '.join(words)

# Function to predict if a message is spam or ham
def predict_spam(message):
        message = preprocess_message(message)
        temp = cv.transform([message]).toarray()
        return classifier.predict(temp)

# Load SMS data
sms = pd.read_csv('Spam SMS Collection.txt', sep='\t', names=['label','message'])
```

```
# Remove duplicates
sms.drop_duplicates(inplace=True)
sms.reset_index(drop=True, inplace=True)

# Preprocess messages
sms['message'] = sms['message'].apply(preprocess_message)

# Feature extraction
cv = CountVectorizer(max_features=2500)
X = cv.fit_transform(sms['message']).toarray()
y = pd.get_dummies(sms['label'])['spam'].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

# Train Multinomial Naive Bayes classifier
classifier = MultinomialNB(alpha=0.1)
classifier.fit(X_train, y_train)
#Accuracy and prediction check
from sklearn.ensemble import StackingClassifier
clf = StackingClassifier (estimators=estimators,
final_estimator=final_estimator)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
print("Accuracy", accuracy_score(y_test,y_pred))
print("Precision", precision_score(y_test,y_pred))

# GUI for interactive prediction
def predict():
        input_msg = entry.get()
        prediction = predict_spam(input_msg)
        messagebox.showinfo("Prediction", "Prediction: {}".format("SPAM" if prediction else
        "HAM (Not Spam)"))

# Create GUI window
root = tk.Tk()
root.title("Spam Detection")
root.geometry("400x200")

# Create label and entry widget
```

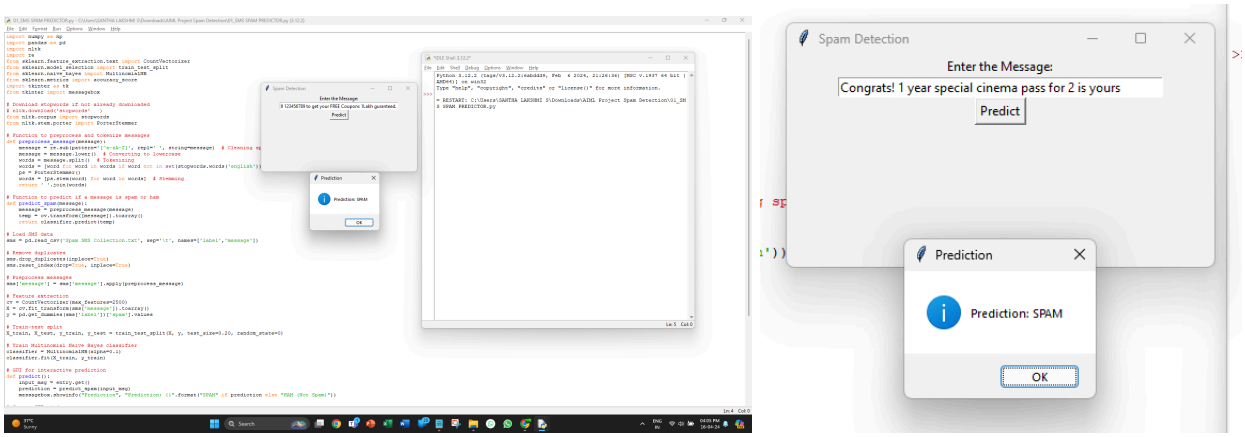
```
label = tk.Label(root, text="Enter the Message:")
label.pack()
entry = tk.Entry(root, width=50)
entry.pack()

# Create predict button
predict_button = tk.Button(root, text="Predict", command=predict)
predict_button.pack()
print("Spam Prediction is ongoing...")

# Run the GUI
root.mainloop()
print("Spam Prediction is Ended")
```

OUTPUT / RESULTS:

HAM Prediction:

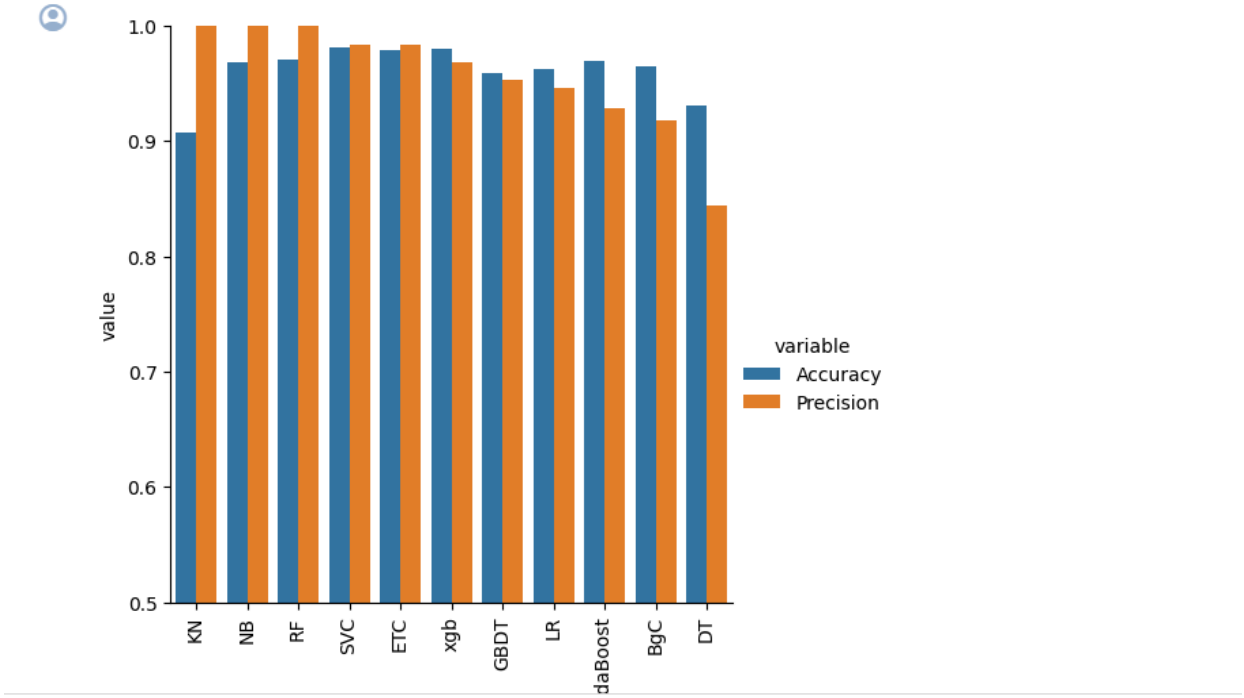


SPAM Prediction:



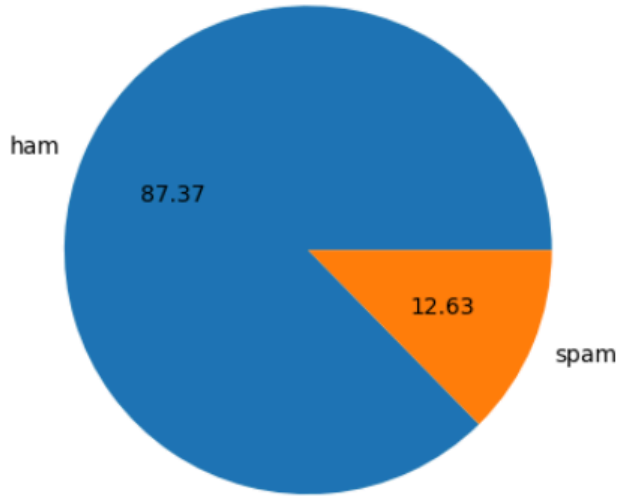
Accuracy and Prediction in bar chart :

```
sns.catplot(x = 'Algorithm', y='value', hue = 'variable', data=performance_df1, kind='bar', height=5)
plt.ylim(0.5,1.0)
plt.xticks(rotation='vertical')
plt.show()
```

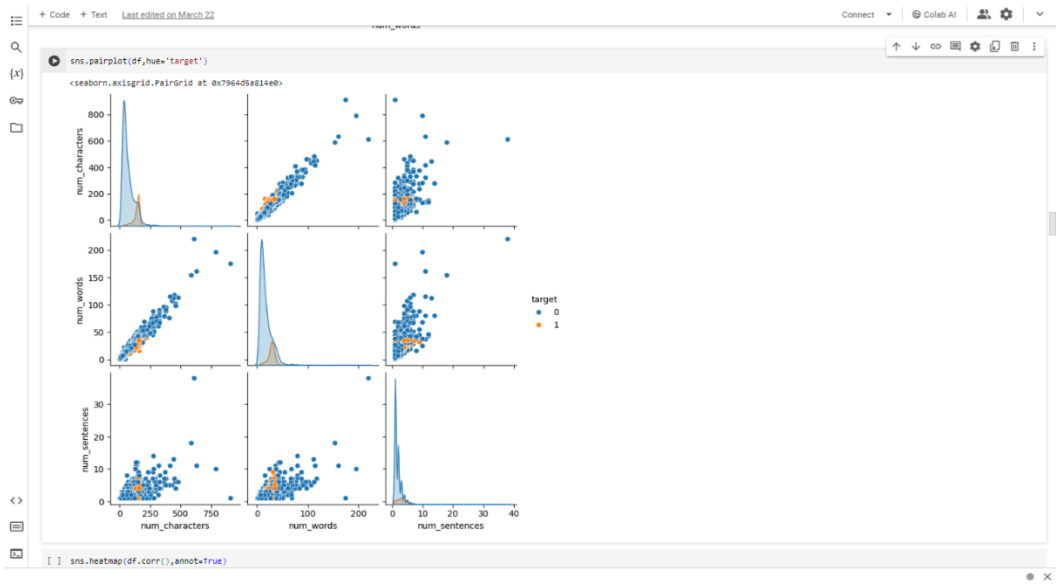


HAM and SPAM count in Dataset:

```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(),labels=['ham','spam'],autopct="%0.2f")
plt.show()
```



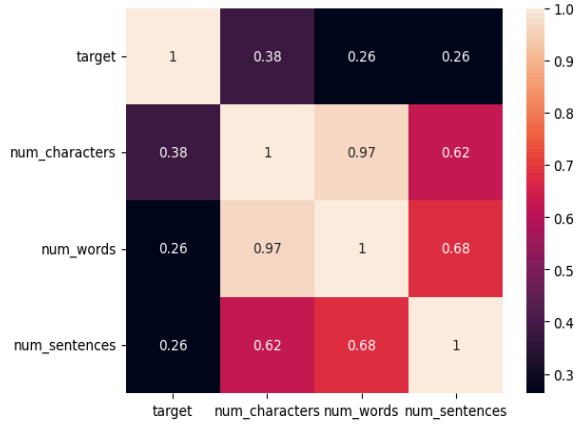
Relationship between dataset shown below as seaborn grid:



Dataset shown in Heatmap:

```
sns.heatmap(df.corr(),annot=True)
```

```
<ipython-input-35-8df7bcac526d>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fals
  sns.heatmap(df.corr(),annot=True)
<Axes: >
```



Model accuracy score:

Accuracy Score

```
[23] acc_s = accuracy_score(y_test, y_pred)*100
```

0s

```
[24] print("Accuracy Score {} %".format(round(acc_s,2)))
```

0s

```
Accuracy Score 97.78 %
```

CONCLUSION:

In conclusion, the development of the SMS spam detection system utilizing the Multinomial Naive Bayes classifier alongside an intuitive graphical user interface (GUI) represents a significant advancement in addressing the pervasive issue of unsolicited messages. Through the integration of sophisticated algorithms and user-friendly technology, the system effectively identifies and filters spam messages, enhancing communication security and reliability.

The success of this project underscores the importance of leveraging advanced machine learning techniques and interactive interfaces to combat digital threats in today's interconnected world. By providing users with a seamless and intuitive tool for spam detection, the system contributes to creating a safer and more efficient digital communication landscape.

Moving forward, future improvements could focus on performance optimization, adaptive learning mechanisms, and seamless integration with existing communication platforms. Additionally, ongoing research and development efforts could explore novel approaches to further enhance the system's accuracy and scalability.

Overall, this project marks a significant step towards addressing the challenges posed by spam messages, ultimately striving towards a more secure and trustworthy digital communication environment for users worldwide.

REFERENCES:

1. Scikit-learn. (n.d.). *https://scikit-learn.org/stable/*
2. *https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf*
3. NLTK Documentation. (n.d.). *https://www.nltk.org/*
4. NLTK 3.6.6 documentation. Steven Bird, Ewan Klein, Edward Loper. (2021).
5. Tkinter Documentation. (n.d.). *https://docs.python.org/3/library/tkinter.html*
6. Tkinter GUI Programming by Example: Create your own sophisticated graphical applications with Python. David Love, John Sholar, Tabor Fisher. (2018). Packt Publishing.