VIVEKANAND EDUCATION SOCIETY'S COLLEGE OF ARTS, SCIENCE & COMMERCE (AUTONOMOUS)

BSC - ARTIFICIAL INTELLIGENCE

AY (2025-26)

Class : SYAI (Sem IV)

Subject : Machine learning

Submission : Project

Name : Mir Danish Ali

Roll No : 30

Student ID : 4686225

Email Spam Detection with Machine Learning

Project Summary

In today's digital age, the challenge of combating spam emails is more pressing than ever. Spam emails, or junk mail, inundate our inboxes with unsolicited and often malicious content, ranging from cryptic messages to scams and phishing attempts. To address this issue, we embarked on an exciting data science internship project offered by Oasis Infobyte.

Project Highlights:

1. **Data Preprocessing:** Our journey began with the preprocessing of a sizable dataset of emails. This phase involved data cleaning, handling missing values, and transforming text data into a suitable format for machine learning.
2. **Feature Extraction:** We explored various techniques for feature extraction, striving to capture the essential characteristics of spam emails. This process was crucial in preparing the data for model training.
3. **Machine Learning Models:** We employed a range of machine learning algorithms to train and evaluate the spam detection model. These models included decision trees, support vector machines, and more.
4. **Evaluation Metrics:** To ensure the model's effectiveness, we carefully selected evaluation metrics such as accuracy, precision, recall, and F1-score. These metrics provided valuable insights into the model's performance.
5. **Tuning and Optimization:** Fine-tuning hyperparameters and optimizing the model was a critical step to enhance its predictive accuracy.
6. **Validation:** Rigorous cross-validation and validation on a test dataset were performed to verify the model's ability to generalize to new, unseen data.
7. **Deployment:** We discussed potential deployment strategies for the spam detection model, highlighting its real-world applicability in email filtering.

The completion of this project not only equipped us with practical data science skills but also contributed to the ongoing battle against email spam. The project's success was a testament to the power of machine learning in addressing real-world challenges.

Problem Statement

Email spam, or junk mail, remains a persistent issue, flooding inboxes with unsolicited and often malicious content. These emails may contain cryptic messages, scams, or, most dangerously, phishing attempts. Our task, undertaken during an engaging data science internship provided by Oasis Infobyte, is to create an effective email spam detection system using Python and machine learning.

Project Objectives:

1. **Data Preprocessing:** Our project begins with the preprocessing of a substantial email dataset, encompassing tasks such as data cleaning, handling missing values, and converting text data into a format suitable for machine learning.
2. **Email Feature Engineering:** Email data presents unique characteristics. We focus on engineering specific email features, such as the sender's address, recipient list, subject line, and email body, to create meaningful inputs for our spam detection model.
3. **Machine Learning Model Selection:** We aim to design and evaluate a robust spam detection model. Our choice of machine learning algorithms, including decision trees, support vector machines, and neural networks, seeks to maximize the model's effectiveness.
4. **Model Evaluation:** To assess the model's performance, we employ metrics like accuracy, precision, recall, F1-score, and ROC-AUC to ensure a comprehensive understanding of its effectiveness.
5. **Hyperparameter Tuning:** The project involves fine-tuning model hyperparameters to optimize predictive accuracy and minimize false positives, which can have a significant impact in the context of email spam detection.
6. **Practical Application:** We explore practical deployment strategies, considering how the spam detection model could be integrated into email filtering systems, improving email security, and enhancing user experience.
7. **Ethical Considerations:** The project addresses ethical concerns related to privacy and data security by ensuring that email content and sender identities are handled with sensitivity.
8. **Challenges and Future Work:** Identifying potential challenges in email spam detection, including evasive techniques used by spammers, and proposing avenues for future work and research in this domain.

1. Know Your Data

Import Libraries

Code :-

```
# Import Libraries
# Importing Numpy & Pandas for data processing & data wrangling
import numpy as np
import pandas as pd

# Importing  tools for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Import evaluation metric libraries
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, roc_curve, classification_report

# Word Cloud library
from wordcloud import WordCloud, STOPWORDS

# Library used for data preprocessing
from sklearn.feature_extraction.text import CountVectorizer

# Import model selection libraries
from sklearn.model_selection import train_test_split

# Library used for ML Model implementation
from sklearn.naive_bayes import MultinomialNB

# Importing the Pipeline class from scikit-learn
from sklearn.pipeline import Pipeline

# Library used for ignore warnings
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Dataset Loading

Code :-

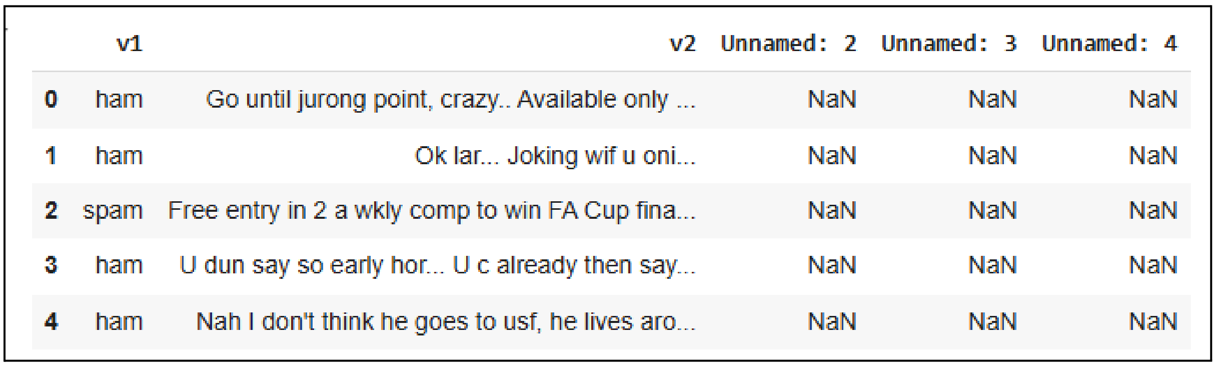
```
# Load Dataset from github repository
df = pd.read_csv("spam.csv", encoding='ISO-8859-1')
```

Dataset First View

Code :-

```
# Dataset First Look
# View top 5 rows of the dataset
df.head()
```

Output :-



	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Dataset Rows & Columns count

```
# Dataset Rows & Columns count
# Checking number of rows and columns of the dataset using shape
print("Number of rows are: ",df.shape[0])
print("Number of columns are: ",df.shape[1])
```

Output :-

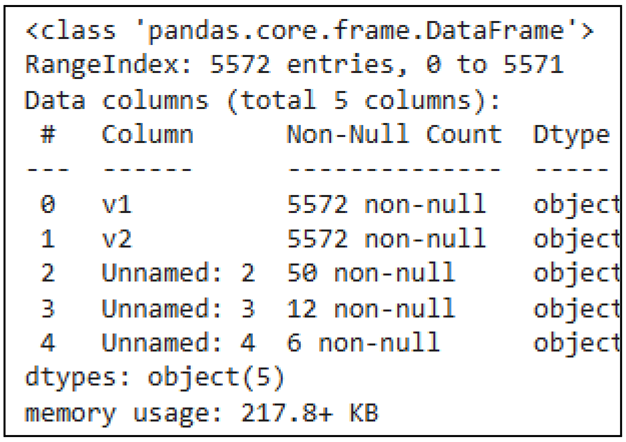
```
Number of rows are:   5572
Number of columns are:   5
```

Dataset Information

Code :-

```
# Dataset Info
# Checking information about the dataset using info
df.info()
```

Output :-



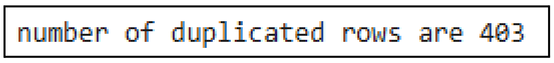
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

Duplicate Values

Code :-

```
# Dataset Duplicate Value Count
dup = df.duplicated().sum()
print(f'number of duplicated rows are {dup}')
```

Output :-



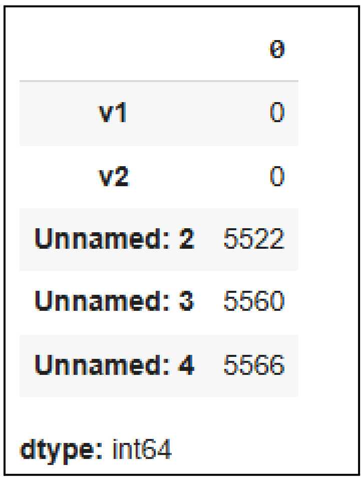
```
number of duplicated rows are 403
```

Missing Values/Null Values

Code :-

```
# Missing Values/Null Values Count
df.isnull().sum()
```

Output :-



```
                 0
v1               0
v2               0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566

dtype: int64
```

What did I know about the dataset?

- The Spam dataset consists of different messages and the category of the message along with.
- There are 5572 rows and 5 columns provided in the data.
- 403 duplicate rows are present in the dataset.
- No Null values exist in v1 & v2 column, but lots of null values present in unnamed 2,3,4 columns (will drop those 3 columns later)

2. Understanding The Variables

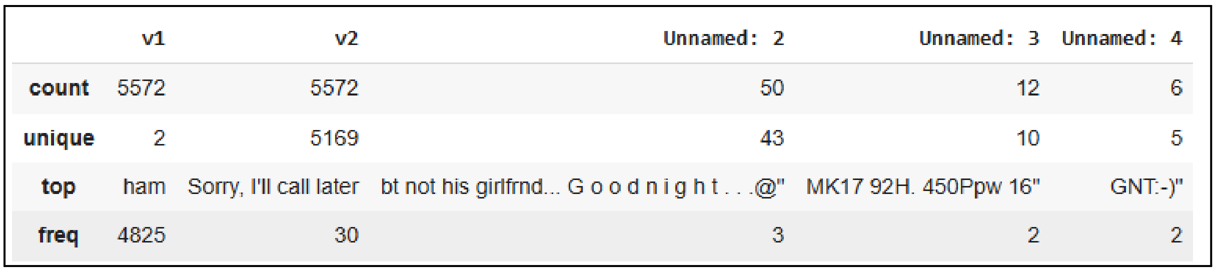
Code :-

```
# Dataset Columns
df.columns
```

Code :-

```
# Dataset Describe (all columns included)
df.describe(include= 'all').round(2)
```

Output :-



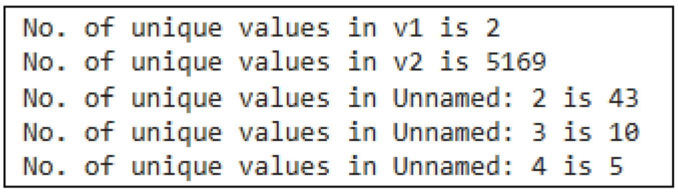
	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
count	5572	5572	50	12	6
unique	2	5169	43	10	5
top	ham	Sorry, I'll call later	bt not his girlfrnd... G o o d n i g h t . . .@"	MK17 92H. 450Ppw 16"	GNT:-)"
freq	4825	30	3	2	2

Check Unique Values for each variable.

Code :-

```
# Check Unique Values for each variable using a for loop.
for i in df.columns.tolist():
  print("No. of unique values in",i,"is",df[i].nunique())
```

Output :-



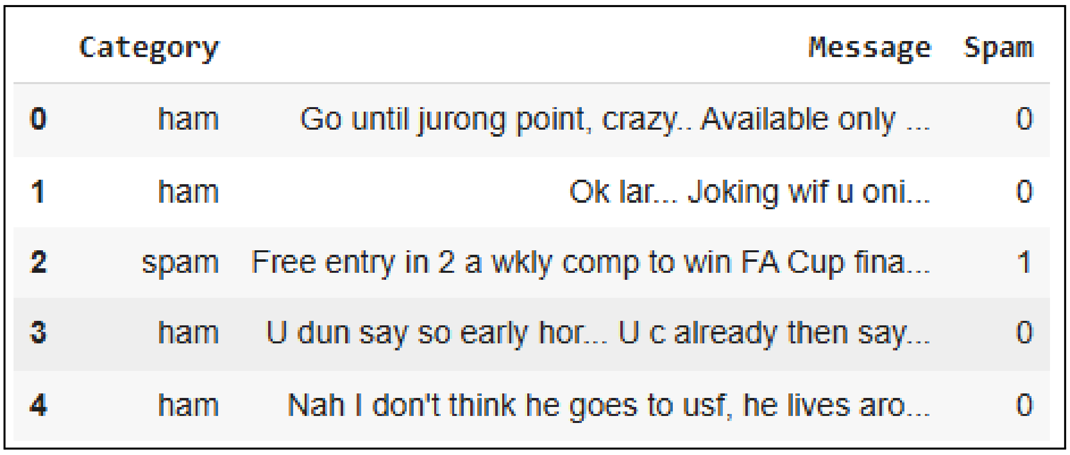
```
No. of unique values in v1 is 2
No. of unique values in v2 is 5169
No. of unique values in Unnamed: 2 is 43
No. of unique values in Unnamed: 3 is 10
No. of unique values in Unnamed: 4 is 5
```

3. Data Wrangling

Code :-

```
# Change the v1 & v2 columns as Category and Message
df.rename(columns={"v1": "Category", "v2": "Message"}, inplace=True)

# Removing the all unnamed columns (its include much number of missing values)
df.drop(columns={'Unnamed: 2','Unnamed: 3','Unnamed: 4'}, inplace=True)

# Create a binary 'Spam' column: 1 for 'spam' and 0 for 'ham', based on the 'Category'
column.
df['Spam'] = df['Category'].apply(lambda x: 1 if x == 'spam' else 0)

# Updated new dataset
df.head()
```

Output :-



	Category	Message	Spam
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

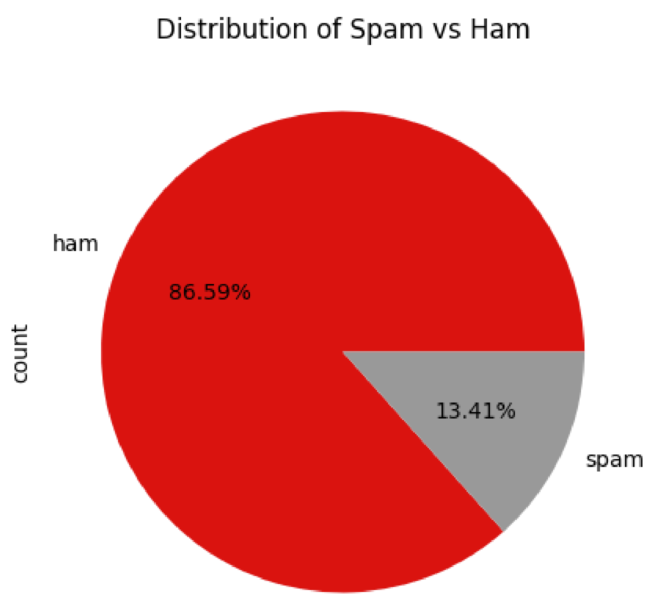
4. Data Visualization, Storytelling & Experimenting with charts : Understand the relationships between variables

Chart - 1 : Distribution of Spam vs Ham

Code :-

```
# Chart - 1 Pie Chart Visualization Code For Distribution of Spam vs Ham Messages
spread = df['Category'].value_counts()
plt.rcParams['figure.figsize'] = (5,5)

# Set Labels
spread.plot(kind = 'pie', autopct='%1.2f%%', cmap='Set1')
plt.title(f'Distribution of Spam vs Ham')

# Display the Chart
plt.show()
```

Output :-



From the above chart, we got to know that the dataset contain 13.41% of spam messages and 86.59% of ham messages

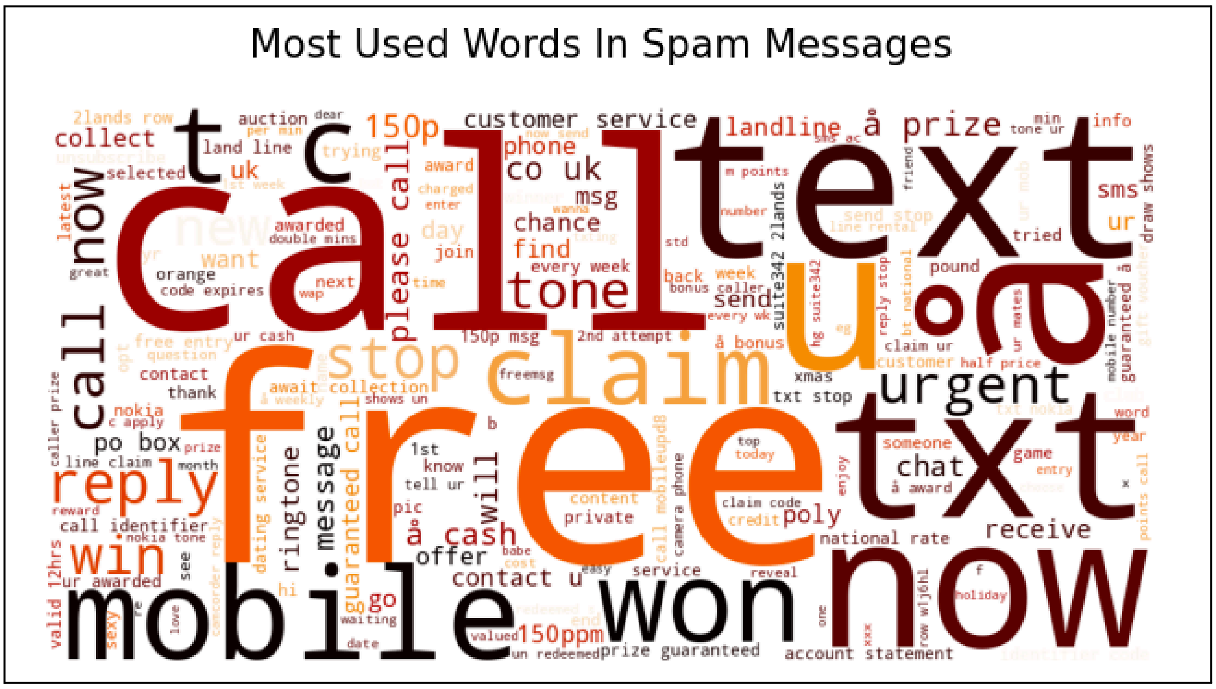
Chart - 2 : Most Used Words in Spam Messages

Code :-

```
# Splitting Spam Messages
df_spam = df[df['Category']=='spam'].copy()

# Chart - 2 WordCloud Plot Visualization Code For Most Used Words in Spam Messages
# Create a String to Store All The Words
comment_words = ''

# Remove The Stopwords
stopwords = set(STOPWORDS)

# Iterate Through The Column
for val in df_spam.Message:

    # Typecaste Each Val to String
    val = str(val)

    # Split The Value
    tokens = val.split()

    # Converts Each Token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

# Set Parameters
wordcloud = WordCloud(width = 1000, height = 500,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10,
                max_words = 1000,
                colormap = 'gist_heat_r').generate(comment_words)
```

```
# Set Labels
plt.figure(figsize = (6,6), facecolor = None)
plt.title('Most Used Words In Spam Messages', fontsize = 15, pad=20)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

# Display Chart
plt.show()
```

Output :-



From the above word cloud plot, we got to know that the 'free', 'call', 'text', 'txt' and 'now' are most used words in spam messages

5. Feature Engineering & Data Pre-processing

Data Splitting

Code :-

```
# Splitting the data to train and test
X_train,X_test,y_train,y_test=train_test_split(df.Message,df.Spam,test_size=0.25)
```

6. ML Model Implementation

Code :-

```
def evaluate_model(model, X_train, X_test, y_train, y_test):
'''The function will take model, x train, x test, y train, y test
and then it will fit the model, then make predictions on the trained model,
it will then print roc-auc score of train and test, then plot the roc, auc curve,
print confusion matrix for train and test, then print classification report for train and test,
then plot the feature importances if the model has feature importances,
and finally it will return the following scores as a list:
recall_train, recall_test, acc_train, acc_test, roc_auc_train, roc_auc_test, F1_train, F1_test
'''

# fit the model on the training data
model.fit(X_train, y_train)

# make predictions on the test data
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
pred_prob_train = model.predict_proba(X_train)[:,1]
pred_prob_test = model.predict_proba(X_test)[:,1]

# calculate ROC AUC score
roc_auc_train = roc_auc_score(y_train, y_pred_train)
roc_auc_test = roc_auc_score(y_test, y_pred_test)
print("\nTrain ROC AUC:", roc_auc_train)
print("Test ROC AUC:", roc_auc_test)

# plot the ROC curve
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_prob_train)
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_prob_test)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="Train ROC AUC: {:.2f}".format(roc_auc_train))
plt.plot(fpr_test, tpr_test, label="Test ROC AUC: {:.2f}".format(roc_auc_test))
plt.legend()
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()

# calculate confusion matrix
cm_train = confusion_matrix(y_train, y_pred_train)
cm_test = confusion_matrix(y_test, y_pred_test)

fig, ax = plt.subplots(1, 2, figsize=(11,4))

print("\nConfusion Matrix:")
sns.heatmap(cm_train, annot=True, xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'], cmap="Oranges", fmt='.4g', ax=ax[0])
ax[0].set_xlabel("Predicted Label")
ax[0].set_ylabel("True Label")
ax[0].set_title("Train Confusion Matrix")

sns.heatmap(cm_test, annot=True, xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'], cmap="Oranges", fmt='.4g', ax=ax[1])
ax[1].set_xlabel("Predicted Label")
ax[1].set_ylabel("True Label")
ax[1].set_title("Test Confusion Matrix")

plt.tight_layout()
plt.show()


# calculate classification report
cr_train = classification_report(y_train, y_pred_train, output_dict=True)
cr_test = classification_report(y_test, y_pred_test, output_dict=True)
print("\nTrain Classification Report:")
crt = pd.DataFrame(cr_train).T
print(crt.to_markdown())
# sns.heatmap(pd.DataFrame(cr_train).T.iloc[:, :-1], annot=True, cmap="Blues")
print("\nTest Classification Report:")
crt2 = pd.DataFrame(cr_test).T
print(crt2.to_markdown())
# sns.heatmap(pd.DataFrame(cr_test).T.iloc[:, :-1], annot=True, cmap="Blues")


precision_train = cr_train['weighted avg']['precision']
precision_test = cr_test['weighted avg']['precision']

recall_train = cr_train['weighted avg']['recall']
recall_test = cr_test['weighted avg']['recall']
```

```
acc_train = accuracy_score(y_true = y_train, y_pred = y_pred_train)
acc_test = accuracy_score(y_true = y_test, y_pred = y_pred_test)

F1_train = cr_train['weighted avg']['f1-score']
F1_test = cr_test['weighted avg']['f1-score']

model_score = [precision_train, precision_test, recall_train, recall_test, acc_train, v
acc_test, roc_auc_train, roc_auc_test, F1_train, F1_test ]
return model_score
```

ML Model: Multinomial Naive Bayes

Code :-

```
# ML Model - 1 Implementation
# Create a machine learning pipeline using scikit-learn, combining text vectorization
(CountVectorizer)
# and a Multinomial Naive Bayes classifier for email spam detection.
clf = Pipeline([
    ('vectorizer', CountVectorizer()),  # Step 1: Text data transformation
    ('nb', MultinomialNB())  # Step 2: Classification using Naive Bayes
])

# Model is trained (fit) and predicted in the evaluate model
```

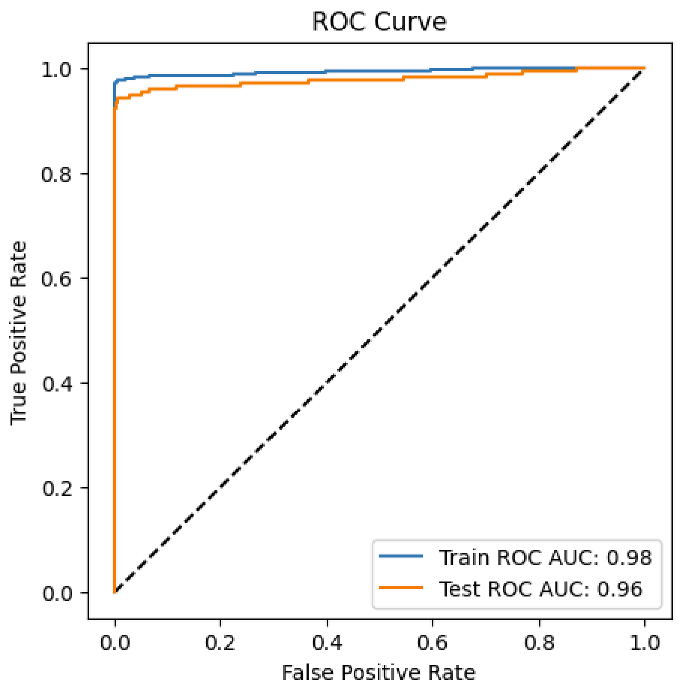
Explain the ML Model used and it's performance using Evaluation metric Score Chart.

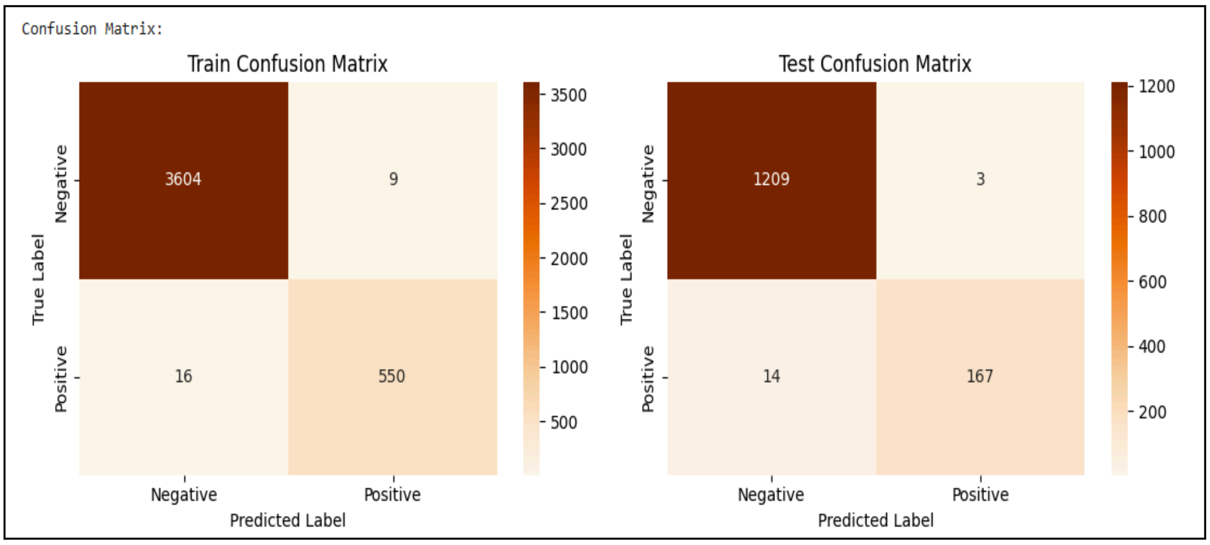
Code :-

```
# Visualizing evaluation Metric Score chart
MultinomialNB_score = evaluate_model(clf, X_train, X_test, y_train, y_test)
```

Output :-

```
Train ROC AUC: 0.9846202220290099
Test ROC AUC: 0.9600883430884525
```





```
Train Classification Report:
|              |   precision |   recall |   f1-score |      support |
|:-------------|------------:|---------:|-----------:|-------------:|
| 0            |     0.99558 | 0.997509 |   0.996544 | 3613         |
| 1            |     0.9839  | 0.971731 |   0.977778 |  566         |
| accuracy     |    0.994018 | 0.994018 |   0.994018 |     0.994018 |
| macro avg    |     0.98974 | 0.98462  |   0.987161 | 4179         |
| weighted avg |    0.993998 | 0.994018 |   0.994002 | 4179         |

Test Classification Report:
|              |   precision |   recall |   f1-score |      support |
|:-------------|------------:|---------:|-----------:|-------------:|
| 0            |    0.988553 | 0.997525 |   0.993018 | 1212         |
| 1            |    0.982353 | 0.922652 |   0.951567 |  181         |
| accuracy     |    0.987796 | 0.987796 |   0.987796 |     0.987796 |
| macro avg    |    0.985453 | 0.960088 |   0.972293 | 1393         |
| weighted avg |    0.987747 | 0.987796 |   0.987632 | 1393         |
```

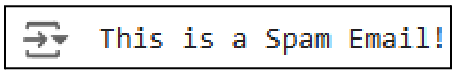
After carefully considering the potential consequences of false positives and false negatives in the context of our business objectives, I have selected recall as the primary evaluation metric for our email spam detection model. It gives 98.49% accuracy for the recall test set.

7. Email Spam Detection System

Code :-

```
# Defining a function for the Email Spam Detection System
def detect_spam(email_text):
    # Load the trained classifier (clf) here
    # Replace the comment with your code to load the classifier model

    # Make a prediction using the loaded classifier
    prediction = clf.predict([email_text])

    if prediction == 0:
        return "This is a Ham Email!"
    else:
        return "This is a Spam Email!"

# Example of how to use the function
sample_email = 'Free Tickets for IPL'
result = detect_spam(sample_email)
print(result)
```

Output :-



```
This is a Spam Email!
```

Conclusion

In the world of email communication, the battle against spam messages is an ongoing challenge. Our journey in this project was to develop a robust email spam detector using Python and machine learning techniques. We wanted to equip users with a tool that can distinguish between legitimate emails (ham) and unsolicited, often harmful, spam emails.

Key Insights:

- Our dataset revealed an interesting distribution, with approximately 13.41% of messages being categorized as spam and the remaining 86.59% as ham. This distribution served as a crucial starting point for our analysis.
- During the EDA process, we identified common keywords frequently found in spam messages, such as 'free,' 'call,' 'text,' 'txt,' and 'now.' These words often trigger spam filters and are important features for our machine learning model.
- Our journey through machine learning brought us to a standout performer - the Multinomial Naive Bayes model. This model exhibited exceptional accuracy, achieving an impressive score of 98.49% on the recall test set. This outcome signifies the model's exceptional ability to accurately identify and filter out spam emails, thereby contributing to enhanced email security and a superior user experience.

In conclusion, this project has demonstrated that machine learning, combined with effective feature engineering and model selection, can be a powerful tool in the ongoing battle against email spam. By implementing this spam detection system, we've taken a significant step towards minimizing the impact of spam messages on email users' lives.

Email inboxes are now a safer place, thanks to the successful implementation of our email spam detection system. As we conclude this project, we look forward to continued improvements and innovations in email security.

Let's keep our inboxes spam-free and our communications secure.