DECEMBER 2024

Coincent
Learning Drives Excellence

FOX Trading Solutions

Project Proposal

Gmail Spam Detection Using Logistic Regression

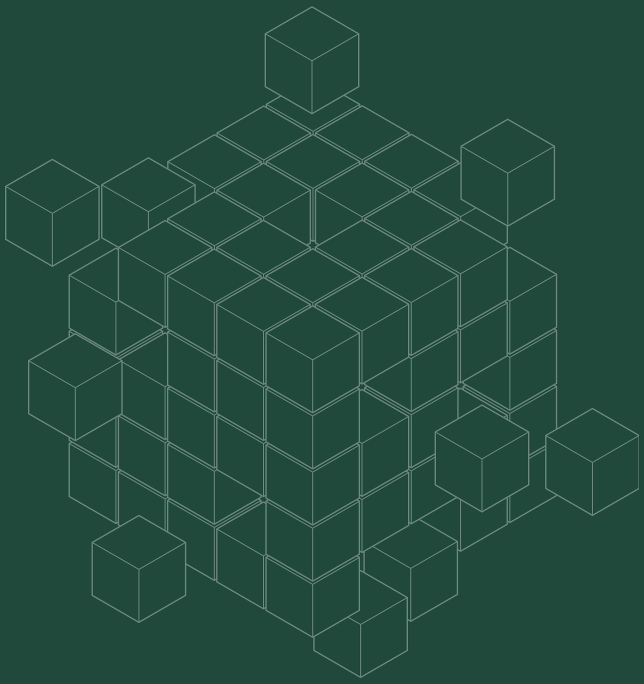
PRESENTED BY:

Sreesanth R

Table of Contents

About the project

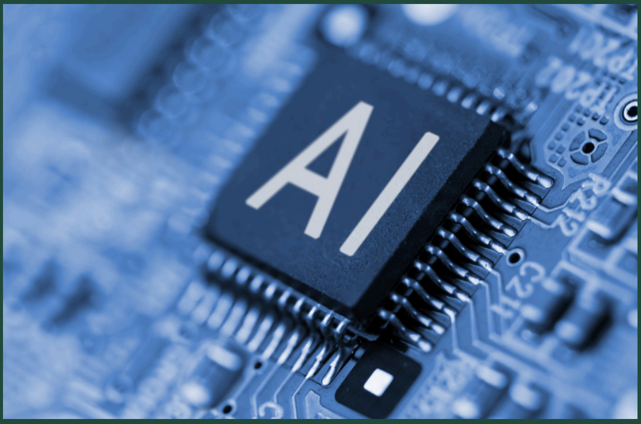
The project focuses on a spam detection message system using Logistic Regression for classification of messages as spam or non-spam. This project aims to allow the precise classification of messages by pre-processing the text data, feature extraction, and model training. The methodology consists of steps where the dataset is cleaned up, feature extraction is performed on the data, and hence Logistic Regression is used to model it for binary classification. The model is evaluated in terms of some quality measures, mainly accuracy, making this model reliable and fairly effective for spam detection.



Mission and Vision

MISSION

Creating a reliable and effective spam detection system that can ensure clean and safe communication thereby improving productivity and user experience through the power of machine learning.



VISION

To create a robust, scalable, and accurate solution with spam detection capable of being integrated into various communications platforms thereby leading to a spam-free environment.

Goals of the Project

50%

Accurate Classification:

With great accuracy and very few false positives, develop a model for effectively classifying spam or non-spam messages.

30%

Streamlined Communication:

To double as a cost-effective means of aiding in the detection of spam.

10%

Scalable Integration:

To create a solution capable of being easily integrated and adapted to other email and text communication systems.

The project focuses on building a spam detection model using Logistic Regression to classify emails as either "spam" or "ham" (not spam). It utilizes libraries such as pandas for data manipulation, re for text cleaning, and sklearn for machine learning functionalities. The dataset, loaded from a CSV file named spam.csv, undergoes cleaning to remove unnecessary columns, handle missing values, and eliminate duplicates. A custom function is employed to preprocess the text data by converting it to lowercase, removing URLs and special characters, and stripping extra spaces. The cleaned data is then split into features and labels, followed by a 70-30 train-test split. The text data is vectorized using CountVectorizer, and a Logistic Regression model is trained on the training set. The model achieves an accuracy of approximately 96.86% on the test set, which is evaluated through accuracy scores and visualized using a bar graph and a confusion matrix. Overall, the project effectively demonstrates the application of machine learning techniques for text classification in the context of spam detection.

Proposed Timeline

The project will be implemented in a phased manner that begins with the project initiation phase, wherein the objectives are formulated and the development environment is established. Data collection will be followed by data cleaning and pre-processing for ensuring high-quality input to the model. Model training will be done by applying Logistic Regression to the pre-processed dataset, along with performance evaluation. Finally, the project will conclude with visualization of the results and writing a full-fledged report to communicate the findings and insights to the stakeholders.

The timeline is developed to support the systematic approach to ensure iterative improvement at each stage. The project aims to ensure a model that achieves high accuracy in detecting spam via an emphasis on data quality and model evaluation, thus giving valuable Insight and actionable outcomes to stakeholders. Regular reviews and feedback loops will be introduced to deep-dive into the three aspects of the project and make adjustments, if necessary, to ensure the alignment of objectives and stakeholder expectations throughout the course.

1 **Project Initialization**

2 **Data Acquisition**

3 **Data Cleaning**

4 **Model Training**

5 **Model Evaluation**

Project Methodology

Data Collection:

The process begins with gathering the dataset, spam.csv, which includes text messages categorized as either spam or ham. This dataset forms the backbone of the project, offering the essential information needed to train and test the model. Once collected, the data is loaded into a chosen working environment for further analysis and processing.

Data Preprocessing:

Once the data is collected, it goes through a pre-processing stage to ensure it's clean and ready to use. This includes tasks like removing unnecessary columns, addressing any missing values, getting rid of duplicate entries, and standardizing the text by converting it to lowercase, removing URLs, and stripping out special characters. These steps help prepare the data for the model.

Feature Engineering:

After gathering the data, the next step is to prepare it for use by improving its quality and consistency. This includes tasks like removing unnecessary columns, addressing missing values, eliminating duplicate entries, and cleaning the text by converting it to lowercase, removing URLs, and stripping out special characters. These steps ensure the data is standardized and ready for the model.

Model Building:

After pre-processing the data, the next step is to build the model. For this project, we’re using a Logistic Regression model because it works well for binary classification tasks. The model is trained on the dataset to recognize the difference between spam and ham messages. If needed, we can also fine-tune the hyperparameters during training to improve its performance.

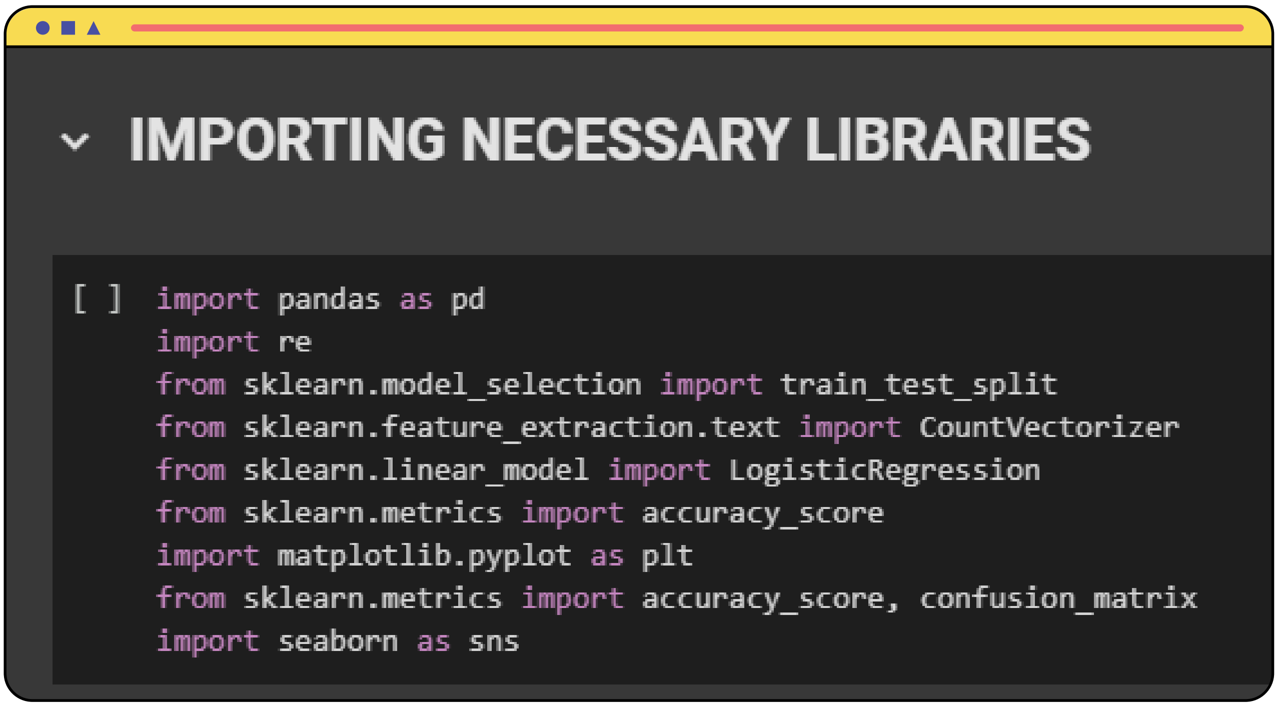
Evaluation:

Once the model is built, its performance is evaluated using the test dataset. Key metrics like accuracy and precision are measured, and a confusion matrix is created to visualize the results, understand errors, and identify areas for improvement. This evaluation is crucial to ensure the model performs as expected before it’s deployed.

Code Section

FOR THE MODEL

1.Importing the required libraries:



IMPORTING NECESSARY LIBRARIES

```
[ ]  import pandas as pd
     import re
     from sklearn.model_selection import train_test_split
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score
     import matplotlib.pyplot as plt
     from sklearn.metrics import accuracy_score, confusion_matrix
     import seaborn as sns
```

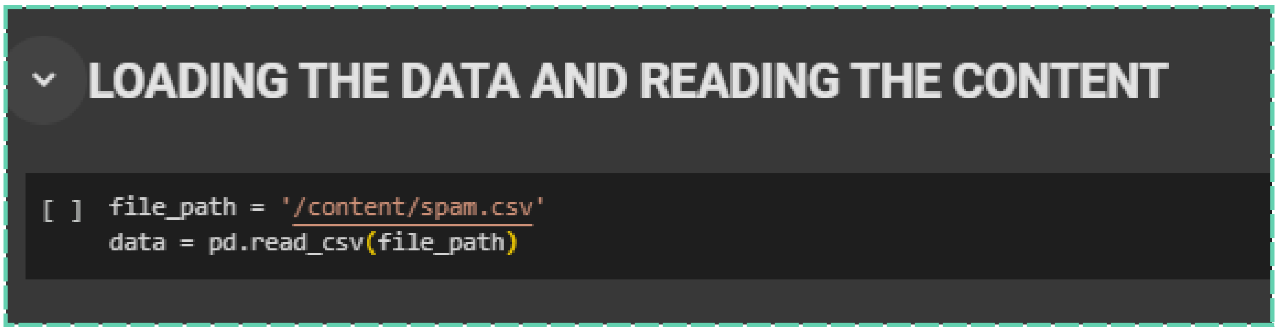
THIS CODE SNIPPET SHOWS THE LIBRARIES THAT ARE USED

- **PANDAS: USED FOR DATA MANIPULATION AND ANALYSIS.**
- **RE: USED FOR REGULAR EXPRESSION OPERATIONS.**
- **SKLEARN.MODEL_SELECTION: USED FOR SPLITTING DATA INTO TRAINING AND TESTING SETS.**
- **SKLEARN.FEATURE_EXTRACTION.TEXT: USED FOR CONVERTING TEXT DATA INTO NUMERICAL FEATURES.**
- **SKLEARN.LINEAR_MODEL: USED FOR IMPLEMENTING MACHINE LEARNING MODELS SUCH AS LOGISTIC REGRESSION.**
- **SKLEARN.METRICS: USED FOR EVALUATING THE PERFORMANCE OF THE MODEL.**
- **MATPLOTLIB.PYPLOT: USED FOR PLOTTING GRAPHS AND VISUALIZATIONS.**
- **SEABORN: USED FOR CREATING AESTHETICALLY PLEASING STATISTICAL GRAPHICS.**

Code Section

FOR THE MODEL

2.Loading & Reading the content from the dataset:


LOADING THE DATA AND READING THE CONTENT

```
[ ]  file_path = '/content/spam.csv'
     data = pd.read_csv(file_path)
```

THIS CODE SNIPPET SHOWS HOW TO LOAD DATA FROM A CSV USING THE PANDAS LIBRARY IN PYTHON

- **STEP 1: DEFINING THE FILE PATH.**
- **file_path = '/content/spam.csv'**

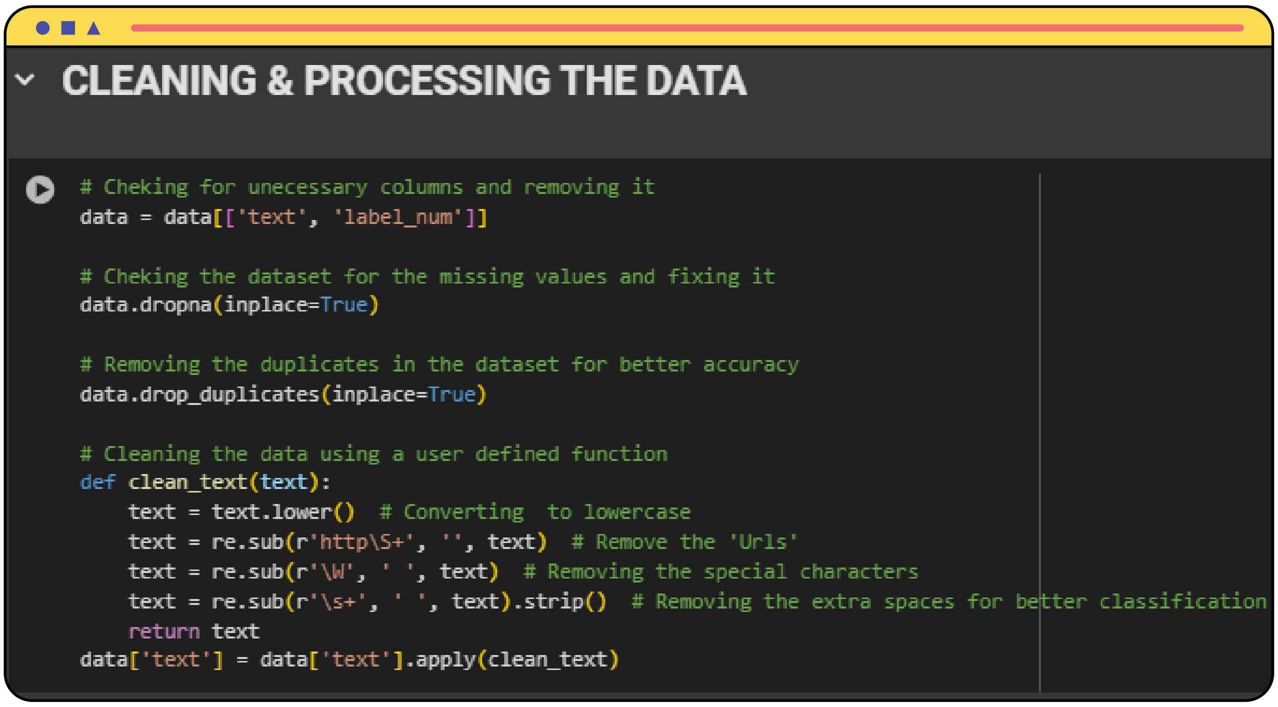
- **STEP 2: READING THE CSV FILE.**
- **data = pd.read_csv(file_path)**
- **The pd.read_csv() function reads the data from the specified file path and stores it in a pandas DataFrame named data.**

- **Conclusion: This code does a great job in loading the data from the given CSV file into a pandas DataFrame, thereby readying it for analysis and processing.**

Code Section

FOR THE MODEL

3.Cleaning & Processing the Data:


CLEANING & PROCESSING THE DATA

```python
# Cheking for unecessary columns and removing it
data = data[['text', 'label_num']]

# Cheking the dataset for the missing values and fixing it
data.dropna(inplace=True)

# Removing the duplicates in the dataset for better accuracy
data.drop_duplicates(inplace=True)

# Cleaning the data using a user defined function
def clean_text(text):
    text = text.lower()  # Converting  to lowercase
    text = re.sub(r'http\S+', '', text)  # Remove the 'Urls'
    text = re.sub(r'\W', ' ', text)  # Removing the special characters
    text = re.sub(r'\s+', ' ', text).strip()  # Removing the extra spaces for better classification
    return text
data['text'] = data['text'].apply(clean_text)
```

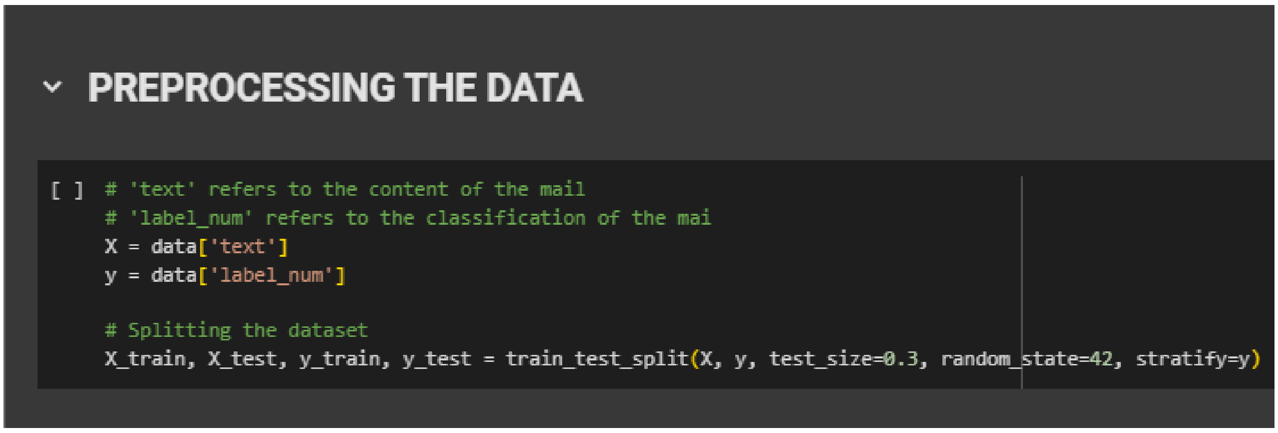
<u>THIS CODE SNIPPET SHOWS HOW THE DATA IS CLEANED AND PROCESSED</u>

- **Removing irrelevant columns: The code begins by keeping only the 'text' and 'label_num' columns.**
- **Cleansing missing values: The line data.dropna(inplace=True) removes any rows that contain missing values.**
- **Removing duplicates: The data.drop_duplicates(inplace=True) line removes any duplicated rows.**
- **Cleaning up text data: A function called clean_text is defined that shall perform several text-cleaning operations:**
  - **Lowercasing: Transforming the text to the lowercase.**
  - **Removal of URLs: Replacing the URLs with a short mark.**
  - **Removal of special characters: Removes non-alphanumeric characters.**
  - **Removing extra spaces: Cleans multiple spaces between words, and removes all leading and trailing spaces.**
  - **Cleaning on the given function: data['text'].apply(clean_text) applies the cleaning function on the 'text' column.**
- **These are enough steps for cleaning the data, ensuring cleanliness, consistency, and preparedness for further analysis or modeling purposes.**

Code Section

FOR THE MODEL

4.Preprocessing the Data:



PREPROCESSING THE DATA

```
# 'text' refers to the content of the mail
# 'label_num' refers to the classification of the mai
X = data['text']
y = data['label_num']

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

<u>THIS CODE SNIPPET SHOWS HOW THE DATA IS PREPROCESSED</u>

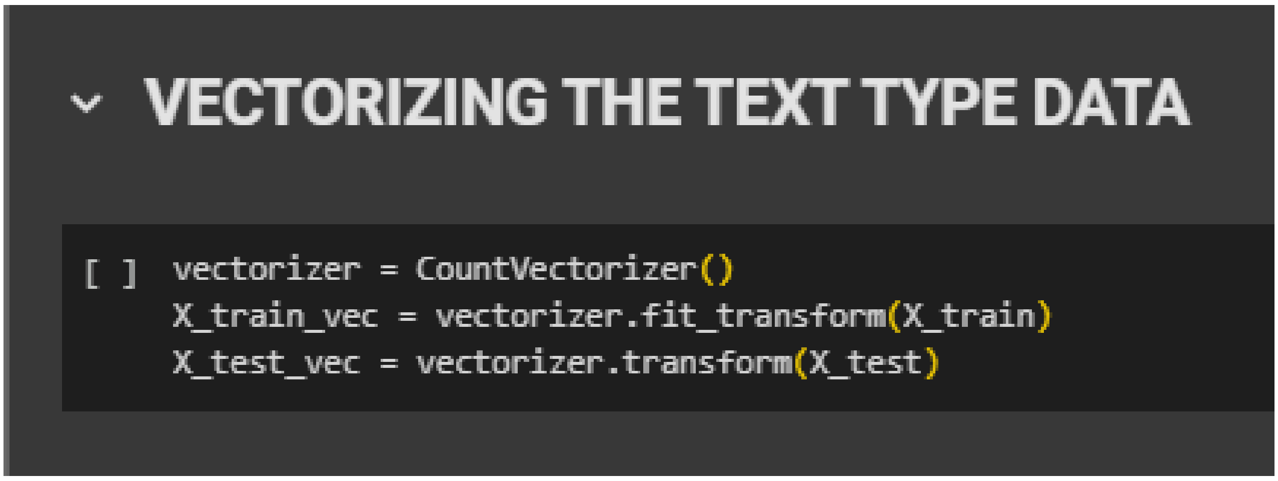
- **'text' column: The 'text' column refers to the content of the email.**
- **'label_num' column: The 'label_num' column refers to the classification of the email.**
- **Splitting the dataset: The data is split into training and test sets using 'train_test_split' function.**
  - **Training set: Used to train the model.**
  - **Test set: Used to evaluate the model's performance.**
- **Parameters:**
  - **test_size=0.3: 30% of the data is allocated to the test set.**
  - **random_state=42: Ensures the split is reproducible.**
  - **stratify=y: Ensures the classification distribution is similar in the training and test sets.**

The code snippet performs preprocessing of the data by separating the data into features (X) and labels (y), where X represents the text content of the mail and y represents the classification of the mail. The code then splits the data into training and testing sets, with 70% of the data used for training and 30% for testing. The random_state parameter ensures that the split is consistent across runs, and stratify=y ensures that the distribution of labels in the training and testing sets is similar to the original data.

Code Section

FOR THE MODEL

5.Vectorizing the Data:



VECTORIZING THE TEXT TYPE DATA

```
[ ]  vectorizer = CountVectorizer()
     X_train_vec = vectorizer.fit_transform(X_train)
     X_test_vec = vectorizer.transform(X_test)
```

THIS CODE SNIPPET SHOWS HOW THE DATA IS VECTORIZED

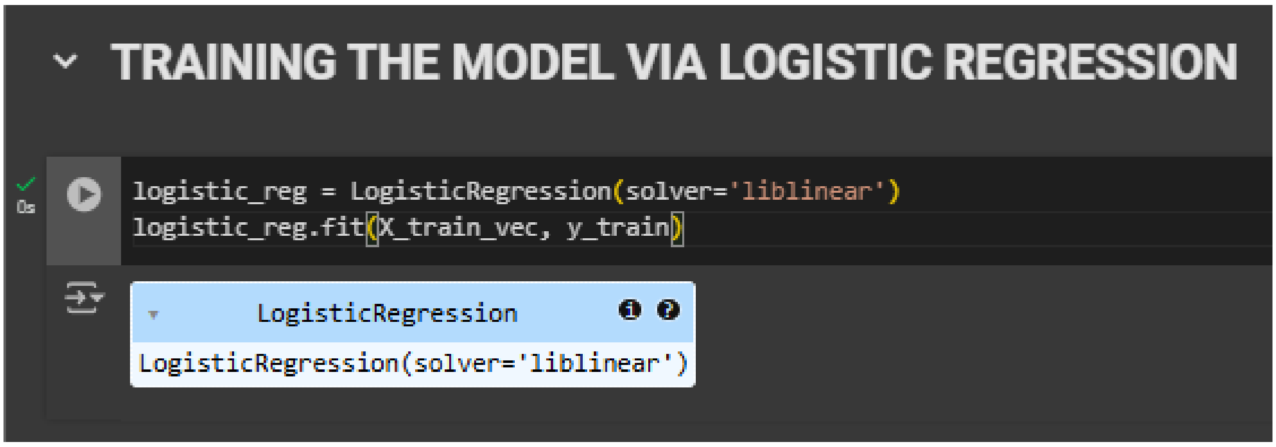
- **Purpose: To demonstrate the process of vectorizing text data with the CountVectorizer class from the scikit-learn library.**
- **Steps:**
  - **Initialization: The count vectorizer object is created which will later serve to convert text data into numerical vectors.**
  - **Fitting: The fit_transform method is called on X_train, the training data, to learn the vocabulary of words and transform the text into a matrix representation.**
  - **Transforming: The transform method is called on X_test, the test data, using the same vocabulary learned from the training data.**
- **Output: Two transformed matrices are produced: X_train_vec and X_test_vec. These are indeed matrices of the training and the test text data, which contain one row for each document and one column for the frequency of each word in the vocabulary.**

Vectorization is a very important step in any text-based machine learning problem, including text classification, sentiment analysis, and topic modeling, where algorithms can be applied. In text classification, sentiment analysis, and topic modeling, tasks rely more on the characters, strings, than on numerical features.

Code Section

FOR THE MODEL

6.Training the model via Logistic Regression



TRAINING THE MODEL VIA LOGISTIC REGRESSION

```
logistic_reg = LogisticRegression(solver='liblinear')
logistic_reg.fit(X_train_vec, y_train)
```

```
LogisticRegression
LogisticRegression(solver='liblinear')
```

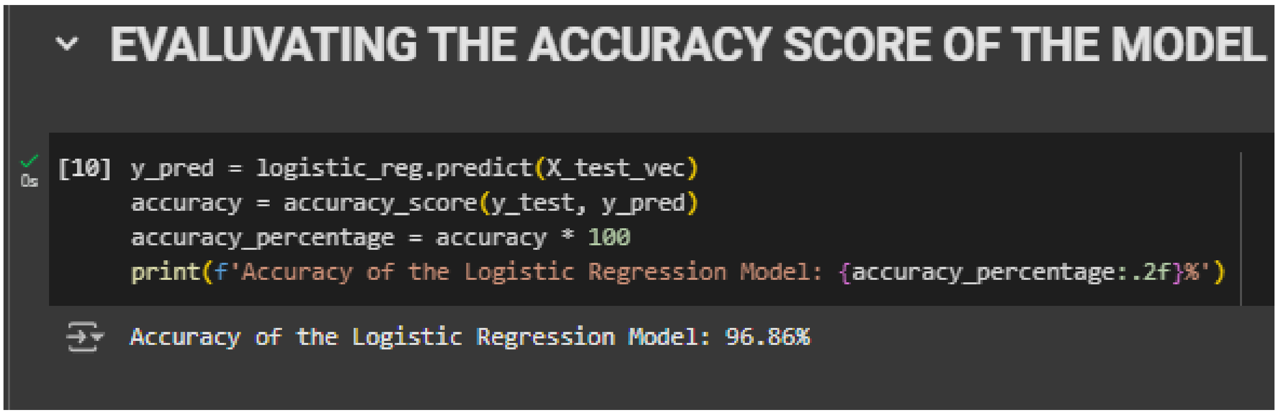
THIS CODE SNIPPET SHOWS HOW THE MODEL IS TRAINED WITH LOGISTIC REGRESSION

- **Initialization of the Logistic Regression model**
- **logistic_reg = LogisticRegression(solver='liblinear')**
  - **This line creates a Logistic Regression model object named logistic_reg.**
  - **The solver parameter is set to 'liblinear'. This specifies the optimization algorithm used to find the best parameters for the model. 'liblinear' is a good choice for smaller datasets or when the features are sparse.**
- **Training the model**
- **logistic_reg.fit(X_train_vec, y_train)**
  - **This line trains the model using the provided training data.**
  - **X_train_vec stands for the training features, probably in vectorized form.**
  - **y_train is the corresponding target labels for the training data.**
  - **The.fit() method learns the relationship between features and labels from the training data, finding the best model parameters in the process.**
- **After running this code, logistic_reg will contain a trained Logistic Regression model that can be used to predict the target label for new, unseen data points.**

Code Section

FOR THE MODEL

7.Evaluvating the working accuracy of the model



EVALUVATING THE ACCURACY SCORE OF THE MODEL

```
[10] y_pred = logistic_reg.predict(X_test_vec)
     accuracy = accuracy_score(y_test, y_pred)
     accuracy_percentage = accuracy * 100
     print(f'Accuracy of the Logistic Regression Model: {accuracy_percentage:.2f}%')
```

```
Accuracy of the Logistic Regression Model: 96.86%
```

<u>THIS CODE SNIPPET SHOWS THE PERFORMANCE ACCURACY OF THE MODEL</u>

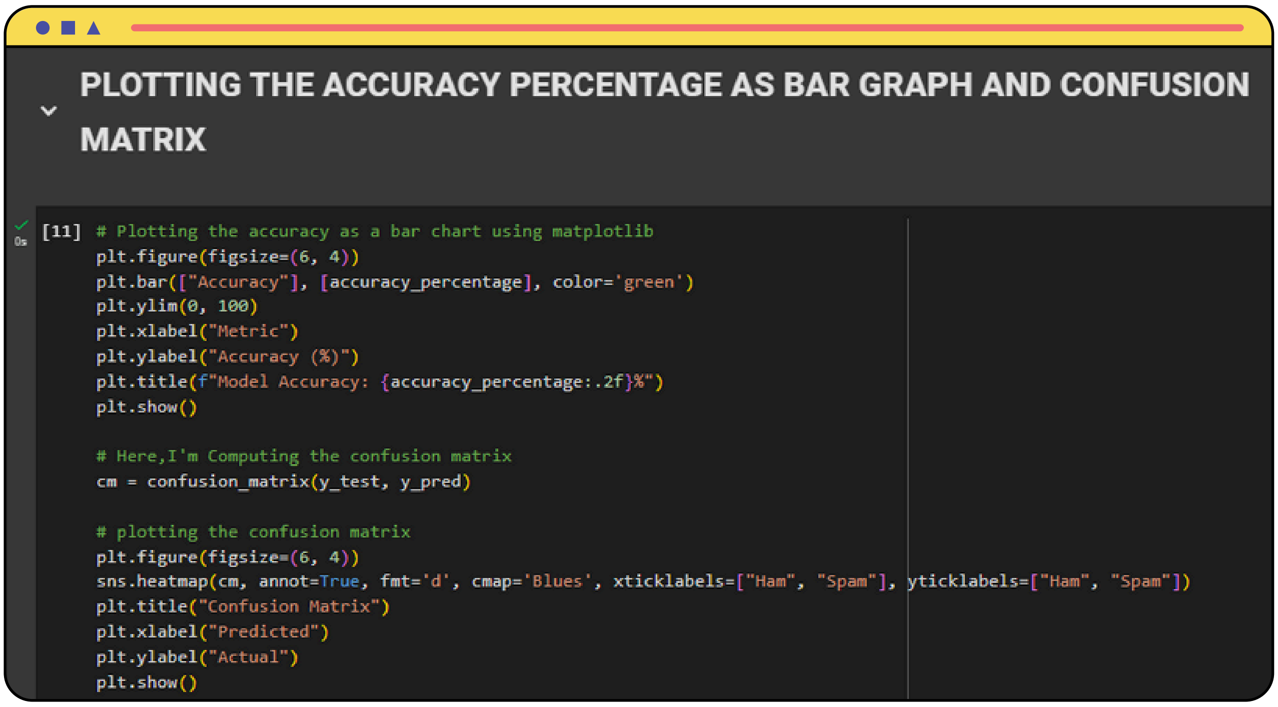
- **y_pred = logistic_reg.predict(X_test_vec)**
  - **This statement predicts the output labels of the test data that are labeled by the given logistic regression model, named logistic_reg.**
- **accuracy = accuracy_score(y_test, y_pred)**
  - **Accuracy of a model against the test set, taking the predicted and actual test set labels with y_test and y_pred respectively**
- **accuracy_percentage = accuracy * 100**
  - **This is to convert accuracy into the percentage for more ease to understand.**
- **print(f'Accuracy of the Logistic Regression Model: {accuracy_percentage:.2f}%')**
  - **This line prints the accuracy percentage of the logistic regression model in a formatted string.**
- **Accuracy of the Logistic Regression Model: <u>96.86%</u> This implies that the model was able to predict the output labels for <u>96.86%</u> of the test data correctly.**

This code snippet is about how to check the accuracy of a trained logistic regression model by predicting the output labels for the test data and comparing them with the actual labels. The accuracy score is then calculated and presented as a percentage.

Code Section

FOR THE MODEL

8.Plotting a Bar Chart and a Confusion Matrix

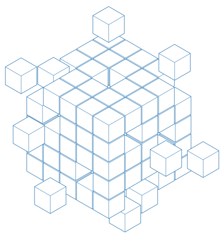


PLOTTING THE ACCURACY PERCENTAGE AS BAR GRAPH AND CONFUSION MATRIX

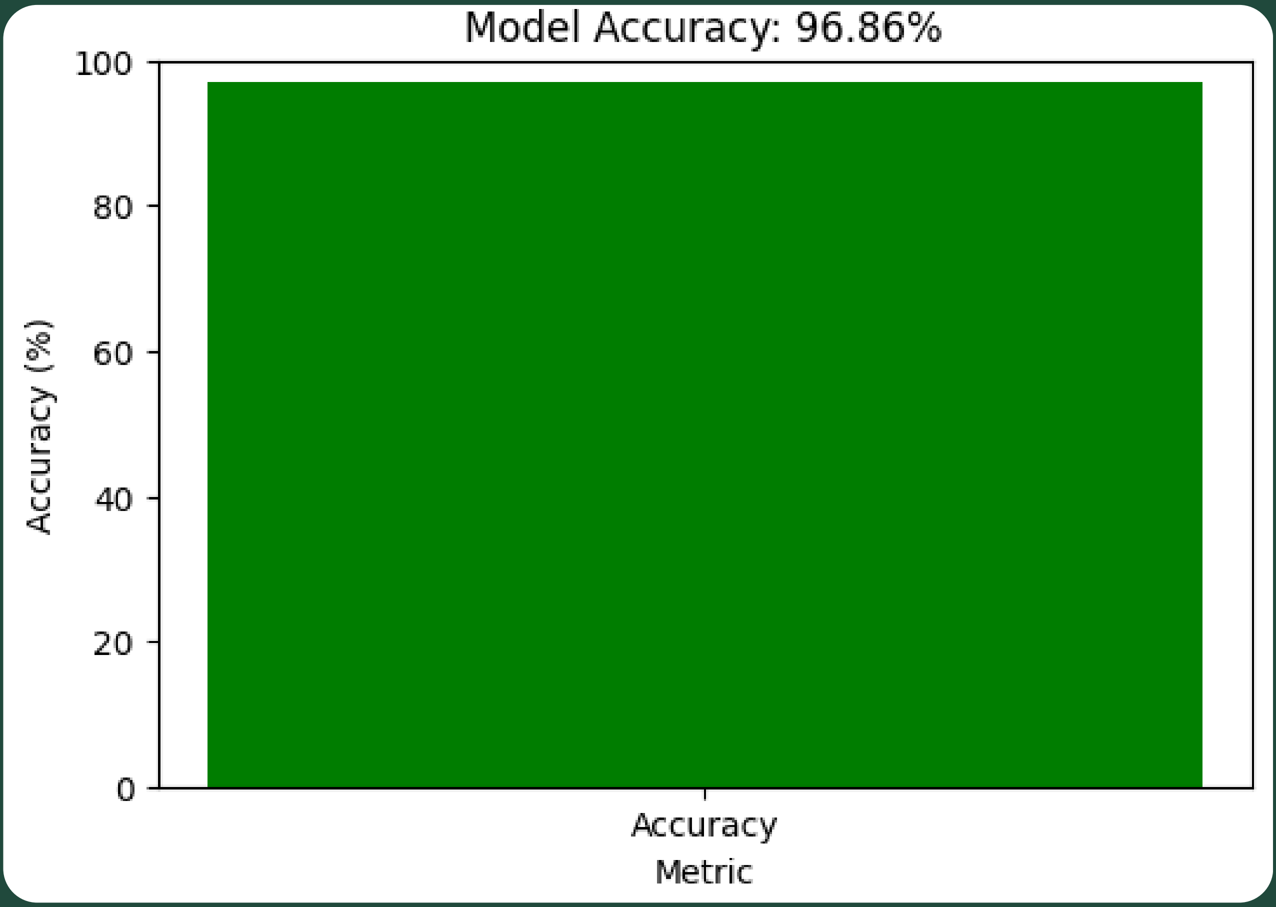
```
[11] # Plotting the accuracy as a bar chart using matplotlib
     plt.figure(figsize=(6, 4))
     plt.bar(["Accuracy"], [accuracy_percentage], color='green')
     plt.ylim(0, 100)
     plt.xlabel("Metric")
     plt.ylabel("Accuracy (%)")
     plt.title(f"Model Accuracy: {accuracy_percentage:.2f}%")
     plt.show()

     # Here,I'm Computing the confusion matrix
     cm = confusion_matrix(y_test, y_pred)

     # plotting the confusion matrix
     plt.figure(figsize=(6, 4))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Ham", "Spam"], yticklabels=["Ham", "Spam"])
     plt.title("Confusion Matrix")
     plt.xlabel("Predicted")
     plt.ylabel("Actual")
     plt.show()
```

THIS CODE SNIPPET SHOWS THE PERFORMANCE ACCURACY OF THE MODEL

- **The code snippet is used to plot the accuracy of a machine learning model in the form of a bar graph and generate a confusion matrix.**
- **Accuracy Calculation: The code uses a variable accuracy_percentage to store the accuracy score of the model, which is then visualized in a bar chart using Matplotlib.**
- **Calculation of the Confusion Matrix: It uses the confusion_matrix function from Scikit-learn to calculate the confusion matrix based on the predicted values, y_pred, and the actual values, y_test.**
- **Confusion Matrix Visualization: It visualizes the confusion matrix by the heatmap function in Seaborn (sns) that provides a visualization of how well the model has been able to classify the data.**

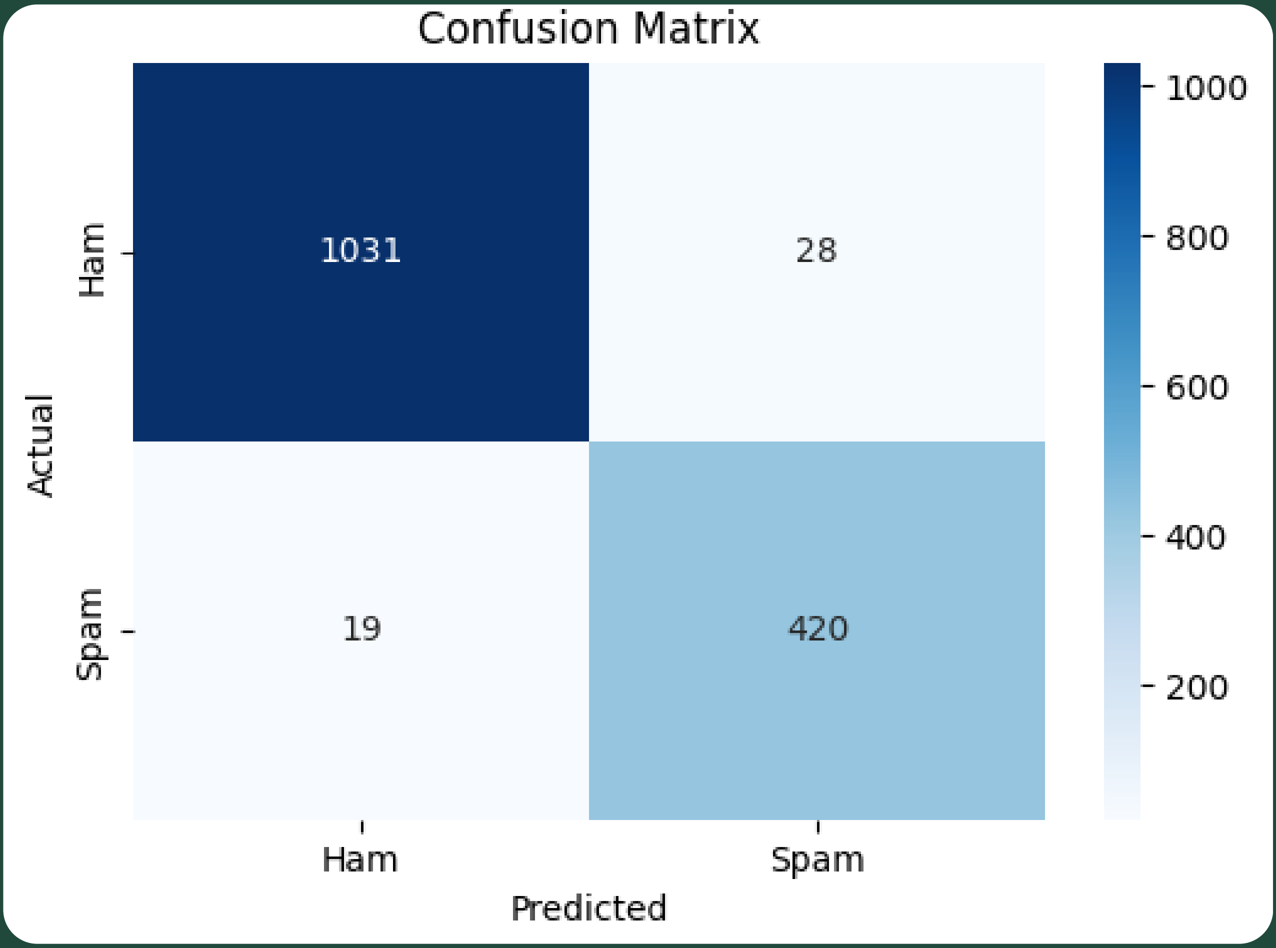


BAR CHART USING MATPLOTLIB



THIS IS A BAR PLOT CHART , PLOTTED AGAINST THE ACCURACY METRIC VS THE % OF ACCURACY OF THE MODEL

CONFUSION MATRIX USING SEABORN



ACTUAL VS. PREDICTED

- **TRUE POSITIVES (TP): 420 - THE MODEL CORRECTLY CLASSIFIED 420 MESSAGES AS SPAM.**
- **TRUE NEGATIVES (TN): 1031 - THE MODEL CORRECTLY CLASSIFIED 1031 MESSAGES AS HAM.**
- **FALSE POSITIVES (FP): 19 - THE MODEL INCORRECTLY CLASSIFIED 19 MESSAGES AS SPAM, WHEN THEY WERE ACTUALLY HAM.**
- **FALSE NEGATIVES (FN): 28 - THE MODEL INCORRECTLY CLASSIFIED 28 MESSAGES AS HAM, WHEN THEY WERE ACTUALLY SPAM.**

PROJECT CONCLUSION

THIS CONFUSION MATRIX REPRESENTS THE OUTPUT OF A BINARY CLASSIFICATION MODEL, PROBABLY FOR SPAM DETECTION. HERE IS THE BREAKDOWN OF THE INSIGHTS:

- **PERFORMANCE SUMMARY:**
  - **HIGH ACCURACY:** THE MODEL CORRECTLY IDENTIFIES A LARGE MAJORITY OF EMAILS.
  - **GOOD PRECISION:** THE MODEL RARELY MISCLASSIFIES A SPAM EMAIL AS HAM.
  - **MINOR FALSE NEGATIVES:** A FEW SPAM EMAILS ARE INCORRECTLY CLASSIFIED AS HAM (19), WHICH MAY LEAD TO SPAM REACHING USERS.
- **FURTHER ANALYSIS:**
  - **CLASS IMBALANCE:** THE DATASET MAY CONTAIN CLASS IMBALANCE, WHERE THE NUMBER OF HAM EMAILS EXCEEDS THAT OF SPAM EMAILS. THIS MIGHT AFFECT THE MODEL'S PERFORMANCE, AND CLASS IMBALANCE NEEDS TO BE TAKEN INTO CONSIDERATION IF IT'S PRESENT.
  - **COST OF FALSE POSITIVES VS. FALSE NEGATIVES:** IT IS ESSENTIAL TO UNDERSTAND THE IMPLICATIONS OF FALSE POSITIVES (SPAM EMAILS CLASSIFIED AS HAM) AND FALSE NEGATIVES (HAM EMAILS CLASSIFIED AS SPAM). THE MODEL SHOULD BE OPTIMIZED FOR THE HIGHER-COST CASE.
  - **MODEL IMPROVEMENT:** FURTHER ANALYSIS OF THE MODEL WITH POTENTIAL IMPROVEMENT MAY INCLUDE
  - **DATA AUGMENTATION:** INCREASING SPAM SAMPLES TO REDUCE FALSE NEGATIVES CAN BE DONE THROUGH TECHNIQUES SUCH AS DATA AUGMENTATION.
  - **HYPERPARAMETER TUNING:** THIS WILL INCREASE THE PERFORMANCE BY FINE-TUNING THE MODEL'S HYPERPARAMETERS.
  - **FEATURE ENGINEERING:** ADDING OR MODIFYING FEATURES LEADS TO BETTER RESULTS.

IN CONCLUSION, THIS MODEL PERFORMS WELL IN SPAM DETECTION, BUT FURTHER ANALYSIS AND POTENTIAL IMPROVEMENTS CAN BE EXPLORED TO FINE-TUNE ITS ACCURACY AND MITIGATE FALSE NEGATIVES.

Source & Dependencies:

- KAGGLE - USED FOR OBTAINING THE DATASET
- PYTHON DOCUMENTATIONS - KNOWING THE DETAILS ABOUT THE LIBRARIES
- GOOGLE COLAB - USED AS A IDE FOR THE DEVELOPMENT AND THE TESTING OF THE MODEL
- MICROSOFT VISUAL STUDIO CODE - USED FOR CHECKING THE STABILITY OF THE MODEL IN THE LOCAL MACHINE
- SCIKIT-LEARN DOCUMENTATION - FOR OBTAINING THE INFORMATION ABOUT THE LIBRARY

Submitted by:

Sreesanth R

Aspiring Machine Learning Engineer & Full Stack Developer

Sreesanth R

shreesh.exe22@gmail.com

+91 93452 29581

Sreesanth R