### **Step 7: Evaluate the Model - Detailed Breakdown**

#### **1. Calculate Accuracy**

Accuracy is the ratio of correctly predicted instances (both true positives and true negatives) to the total instances. It gives an overall measure of how many emails were correctly classified as spam or ham.

python

Copy code

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

Here’s what each part does:

* metrics.accuracy\_score(y\_test, y\_pred): Calculates the accuracy score by comparing the actual labels (y\_test) with the predicted labels (y\_pred).

#### **2. Generate Classification Report**

The classification report provides a detailed breakdown of the model's performance, including precision, recall, and F1-score for each class (spam and ham).

python

Copy code

report = metrics.classification\_report(y\_test, y\_pred, target\_names=['ham', 'spam'])

print(report)

Here’s what each part does:

* metrics.classification\_report(y\_test, y\_pred, target\_names=['ham', 'spam']): Generates a report with precision, recall, F1-score, and support (the number of true instances for each label).

**Understanding the Metrics:**

* **Precision**: The ratio of correctly predicted positive observations to the total predicted positives. High precision means that a high percentage of the emails flagged as spam were actually spam.
* **Recall (Sensitivity)**: The ratio of correctly predicted positive observations to all observations in the actual class. High recall means that the model identified most of the spam emails.
* **F1-Score**: The weighted average of precision and recall. It provides a balance between precision and recall, especially useful when you have an uneven class distribution.
* **Support**: The number of actual occurrences of the class in the test set.

### **Running the Code in Your Development Environment**

#### **Where to Run the Code**

You can run the code in any Python development environment, such as:

* **Jupyter Notebook**: Interactive environment that allows you to run code in cells.
* **VS Code**: Popular code editor that supports Python.
* **PyCharm**: Integrated Development Environment (IDE) for Python.
* **Any other Python IDE or code editor**.

### **Complete Evaluation Code to Run**

Make sure you have already trained your model and have y\_test and y\_pred ready. Here’s the complete code for evaluating the model:

python

Copy code

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import Pipeline

from sklearn import metrics

import nltk

from nltk.corpus import stopwords

import string

import joblib

# Download stopwords

nltk.download('stopwords')

stop\_words = set(stopwords.words('english'))

# Preprocessing function

def preprocess\_text(text):

# Remove punctuation

text = ''.join([char for char in text if char not in string.punctuation])

# Convert to lowercase

text = text.lower()

# Remove stopwords

text = ' '.join([word for word in text.split() if word not in stop\_words])

return text

# Function to read and clean the CSV file

def read\_and\_clean\_csv(filepath):

rows = []

with open(filepath, 'r', encoding='latin-1') as file:

for line in file:

parts = line.split(',', 1) # Split on the first comma only

if len(parts) == 2:

label = parts[0].strip() # Strip whitespace

message = parts[1].strip() # Strip whitespace

rows.append([label, message])

else:

print(f"Skipping malformed line: {line}")

return pd.DataFrame(rows, columns=['v1', 'v2'])

# Load and clean the dataset

try:

data = read\_and\_clean\_csv('spam.csv')

print("Initial data shape:", data.shape)

data = data[['v1', 'v2']]

data.columns = ['label', 'message']

print("Data shape after selecting columns:", data.shape)

print("First few rows of the dataset:")

print(data.head())

except Exception as e:

print(f"Error loading CSV: {e}")

exit()

# Print unique values in the label column before any processing

print("Unique labels before mapping:", data['label'].unique())

# Drop rows with missing values

data.dropna(inplace=True)

print("Data shape after dropping NAs:", data.shape)

# Remove special characters and whitespace from labels

data['label'] = data['label'].str.strip().str.replace(r'[^\w\s]', '', regex=True).str.lower()

print("Unique labels after stripping special characters:", data['label'].unique())

# Correctly map labels to numerical values (spam=1, ham=0)

data['label'] = data['label'].map({'ham': 0, 'spam': 1})

print("Unique labels after mapping:", data['label'].unique())

# Check for rows where the label could not be encoded and drop them

data = data.dropna(subset=['label'])

print("Data shape after encoding labels and dropping rows with NaNs in labels:", data.shape)

# Preprocess messages

data['message'] = data['message'].apply(preprocess\_text)

print("Data shape after preprocessing messages:", data.shape)

# Ensure there is data left to split

if data.shape[0] == 0:

print("No data available after cleaning. Please check the CSV file for formatting issues.")

exit()

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['message'], data['label'], test\_size=0.2, random\_state=42)

print(f"Train set size: {len(X\_train)}, Test set size: {len(X\_test)}")

# Build and train the pipeline

pipeline = Pipeline([

('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('clf', MultinomialNB())

])

pipeline.fit(X\_train, y\_train)

# Predict on test set

y\_pred = pipeline.predict(X\_test)

# Evaluate the model

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

# Ensure there are at least two classes in the test set

if len(set(y\_test)) > 1:

report = metrics.classification\_report(y\_test, y\_pred, target\_names=['ham', 'spam'])

print(report)

else:

print("Not enough classes to generate a classification report.")

### **Running the Code in Jupyter Notebook**

1. **Open Jupyter Notebook**: You can start it by running jupyter notebook in your terminal or command prompt.
2. **Create a New Notebook**: Click on "New" and select "Python 3".
3. **Copy and Paste the Code**: Copy the evaluation code above and paste it into a cell in your notebook.
4. **Run the Cell**: Press Shift + Enter to run the cell.

### **Running the Code in VS Code**

1. **Open VS Code**: Ensure you have the Python extension installed.
2. **Create a New Python File**: Save the file with a .py extension.
3. **Copy and Paste the Code**: Copy the evaluation code above and paste it into the file.
4. **Run the Code**: You can run the code by opening the terminal in VS Code and typing python filename.py.

By following these steps, you should be able to evaluate your spam detection model and understand its performance metrics in detail.