Project :

Machine learning Email Spam Detector with Python

Machine learning is a field of study and research that focuses on developing algorithms and models that enable computers and systems to learn from and make predictions or decisions based on data, without being explicitly programmed. It is a subset of artificial intelligence (AI) that relies on statistical techniques to enable systems to learn and improve from experience.

Machine learning has numerous applications across various domains, including image and speech recognition, natural language processing, recommendation systems, fraud detection, autonomous vehicles, healthcare, finance, and many others. It continues to advance rapidly, with new algorithms, models, and techniques being developed to tackle increasingly complex problems and improve performance.

We've all been the recipient of spam emails before. Spam mail, or junk mail, is a type of email that is sent to a massive number of users at one time, frequently containing cryptic messages, scams, or most dangerously, phishing content.

Machine learning finds applications in various domains, including:

1. **Image and Video Processing:**
    - **Object detection and recognition**
    - **Image classification and segmentation**
    - **Video analysis and action recognition**
2. **Recommendation Systems:**
    - **Personalized product recommendations**
    - **Movie or music recommendations**
    - **Content-based filtering and collaborative filtering**
3. **Fraud Detection and Anomaly Detection:**
    - **Credit card fraud detection**
    - **Network intrusion detection**
    - **Unusual pattern detection in time series data**
4. **Healthcare and Biomedical Research:**
    - **Disease diagnosis and prognosis**
    - **Medical image analysis**
    - **Drug discovery and development**

While spam emails are sometimes sent manually by a human, most often, they are sent using a bot. Most popular email platforms, like Gmail and Microsoft Outlook, automatically filter spam emails by screening for recognizable phrases and patterns. A few common spam emails include fake advertisements, chain emails, and impersonation attempts. While these built-in spam detectors are usually effective, sometimes, a particularly well-disguised spam email may fall through the cracks, landing in your inbox instead of your spam folder.

Clicking on a spam email can be dangerous, exposing your computer and personal information to different types of malware. Therefore, it's important to implement additional safety measures to protect your device, especially when it handles sensitive information like user data.

In this tutorial, we'll use Python to build an email spam detector. Then, we'll use machine learning to train our spam detector to recognize and classify emails into spam and non-spam.

Pandas is a library used mostly used by data scientists for data cleaning and analysis.

Scikit-learn, also called Sklearn, is a robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling, including classification, regression, clustering, and dimensionality reduction via a consistent interface.Run the command below to import the necessary dependencies:

```
# Step 3: Implement the Naive Bayes Algorithm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
```

ALGORITHM

Step 1: Gather the Data

- The code imports the necessary libraries, including pandas for data manipulation.
- It loads the dataset from a CSV file using the **pd.read_csv()** function.

Step 2: Preprocess the Data

- This step involves performing preprocessing steps on the text data, such as removing stopwords, tokenization, and converting the text into numerical features.
- This part is not included in the code snippet provided, and you would need to customize it based on your specific preprocessing requirements.

Step 3: Implement the Naive Bayes Algorithm

- The code imports the required libraries, including **TfidfVectorizer** for text feature extraction, **MultinomialNB** for the Naive Bayes model, and **train_test_split** for dataset splitting.
- It defines the input features (**X**) and the target variable (**y**) based on the dataset.
- The dataset is split into training and testing sets using **train_test_split()**.
- A TF-IDF vectorizer is created and applied to the training and testing data to convert the text features into numerical representations.

- **A Multinomial Naive Bayes model is instantiated and trained on the vectorized training data using the fit() function.**

Step 4: Evaluate the Model

- **The code makes predictions on the testing dataset using the trained model's predict() function.**
- **Performance metrics such as accuracy, precision, recall, and F1 score are computed using the accuracy_score(), precision_score(), recall_score(), and f1_score() functions from sklearn.metrics.**
- **The metrics are then printed to evaluate the model's performance.**

Please note that this code assumes you have a labeled dataset where the target variable is labeled as 'label' and the email text is stored under the 'email_text' column. You will need to adjust these parts based on your specific dataset structure.

Remember to complete Step 2 (preprocessing) and provide the necessary dataset to run the code successfully.

INITIATION OF PROJECT:

To get started, first, run the code below:

```
# Step 1: Gather the Data
import pandas as pd

# Load the dataset (replace 'dataset.csv' with the actual file path)

# data = open(r"C:\Users\hp\Documents\Zoom\emails.csv", "r", encoding="utf8")
data = pd.read_csv(r"C:\Users\hp\Documents\Zoom\mail_data.csv")
```

In the code above, we created a mail_data.csv file, which we'll turn into a data frame and save to our folder spam. A data frame is a structure that aligns data in a tabular fashion in rows and columns, like the one seen in the following image.

Python train_test_split()

We'll use a train-test split method to train our email spam detector to recognize and categorize spam emails. The train-test split is a technique for evaluating the performance of a machine learning algorithm. We can use it for either classification or regression of any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two separate datasets. The first dataset is used to fit the model and is referred to as the training dataset. For the second dataset, the test dataset, we provide the input element to the model. Finally, we make predictions, comparing them against the actual output.

- Train dataset: used to fit the machine learning model
- Test dataset: used to evaluate the fit of the machine learning model

In practice, we'd fit the model on available data with known inputs and outputs. Then, we'd make predictions based on new examples

for which we don't have the expected output or target values. We'll take the data from our sample .csv file, which contains examples pre-classified into spam and non-spam, using the labels spam and ham, respectively.

To split the data into our two datasets, we'll use scikit-learn's train_test_split() method.

Let's say we have 100 records in the loaded dataset. If we specify the test dataset is 30 percent, we'll split 70 records for training and use the remaining 30 records for testing.

Run the command below:

```
# Define the input features (X) and the target variable (y)
X = data['Message']
y = data['Category']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

X=data['Message'] assigns the column Message from spam to x. It contains the data that we'll run through the model. Y= data["Category"] assigns the column Category from spam to Y, telling the model to correct the answer. You can see a screenshot of the raw dataset below.

The function X_train, X_test,Y_train, Y_test = train_test_split(X,Y,test_size = 0.2) divides columns X and Y into X_train for training inputs, Y_train for training labels, X_test for testing inputs, and Y_test for testing labels.

test_size=0.2 sets the testing set to 20 percent of z and y. You can see an example of this in the screenshot below, where the ham label indicates non-spam emails, and spam.

Step 3: Implement the Naive Bayes Algorithm

- Create a Python script or notebook to implement the Naive Bayes algorithm for classification. You can use libraries like scikit-learn to simplify the implementation.
- Define the input features (e.g., TF-IDF vectors) and the target variable (spam or not spam).
- Train the Naive Bayes model using the training dataset.

```
# Step 3: Implement the Naive Bayes Algorithm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

# Define the input features (X) and the target variable (y)
X = data['Message']
y = data['Category']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create TF-IDF vectorizer and transform the input text
vectorizer = TfidfVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Create and train the Naive Bayes model
model = MultinomialNB()
model.fit(X_train_vectorized, y_train)
```

Step 4: Evaluate the Model

- Use the trained model to make predictions on the testing dataset.
- Evaluate the model's performance using appropriate evaluation metrics such as accuracy, precision, recall, or F1 score.
- Adjust the model or experiment with different algorithms if necessary.

```python
# Step 4: Evaluate the Model
# Make predictions on the testing dataset
y_pred = model.predict(X_test_vectorized)

# Evaluate the model's performance
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label='spam')
recall = recall_score(y_test, y_pred, pos_label='spam')
f1 = f1_score(y_test, y_pred, pos_label='spam')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 Score',f1)
```

OUTPUT

```
Accuracy: 0.9632286995515695
Precision: 1.0
Recall: 0.7337662337662337
F1 Score 0.8464419475655431
```