Email Spam Filtering Project

This project aims to create a machine learning model that classifies email messages as either "spam" or "ham" (not spam). We use natural language processing (NLP) techniques to clean the text data, extract features using the TF-IDF (Term Frequency-Inverse Document Frequency) method, and train a Naive Bayes model for classification. The project also includes data exploration, visualization, and performance evaluation.

1. Import Libraries

```
python
Copy code
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.metrics import accuracy_score
import warnings
```

The project starts by importing necessary Python libraries:

- **Pandas and Numpy**: Used for handling and manipulating data.
- **Matplotlib and Seaborn**: For data visualization.
- **TfidfVectorizer**: From `sklearn`, used for text feature extraction.
- **MultinomialNB**: Naive Bayes algorithm suited for classification tasks on text data.
- **nltk**: Natural Language Toolkit for text preprocessing, tokenization, and stemming.
- **warnings**: To suppress warnings for cleaner output.

To ensure that stopwords and the `punkt` tokenizer are available for processing, we download them using `nltk.download()`.

```
python
Copy code
nltk.download('stopwords')
nltk.download('punkt')
```

2. Load and Inspect the Dataset

```
python
Copy code
data_path = r'D:\# DATA SCIENCE\# PROJECTS\- PROJECTS
INTERNSHIPS\TECHNOHACKS SOLUTIONS - DATA SCIENCE\Email Spam Filtering
Project'
df = pd.read_csv(f'{data_path}/spam.csv', encoding='latin1')
print(df.head())
```

Here, we load the dataset using `pd.read_csv()`. The dataset is in CSV format and is encoded in `latin1`, a commonly used encoding for text files. The first few rows are displayed using `df.head()`, which allows us to inspect the structure of the data.

---

3. Rename Columns and Basic Data Exploration

```
python
Copy code
df = df[['Type', 'Email']]
df.columns = ['label', 'message']
```

The dataset contains several columns, but only the 'Type' (class label) and 'Email' (message content) columns are relevant to this task. We rename the columns to 'label' and 'message' for simplicity.

```
python
Copy code
print(df.info())
print(df.describe())
print(df.isnull().sum())
```

The `info()` method provides an overview of the dataset, including column names, data types, and non-null counts. `describe()` gives summary statistics for numeric columns (though not very informative in text data), and `isnull().sum()` checks for missing values, ensuring data quality.

---

4. Class Distribution and Visualizations

```
python
Copy code
print(df['label'].value_counts())
plt.figure(figsize=(8, 6))
sns.countplot(x='label', data=df, palette={'ham': 'blue', 'spam': 'red'})
plt.title('Distribution of Ham and Spam Messages')
plt.show()
```

Here, we explore the class distribution to check for data imbalance between spam and ham messages. This is an important step because imbalanced datasets can affect the performance of classification models.

We also use `seaborn` to visualize the class distribution using a bar plot, where each bar represents the count of 'ham' and 'spam' messages in the dataset. The colors red (for spam) and blue (for ham) enhance the visual contrast.

---

5. Text Preprocessing

Tokenization, Stopword Removal, and Stemming

Text preprocessing is critical to converting raw text into a format suitable for machine learning models. The following steps are applied:

1. **Tokenization**: Splitting the text into individual words or tokens.
2. **Lowercasing**: Converting all characters to lowercase to avoid treating 'Spam' and 'spam' as different words.
3. **Stopword Removal**: Removing common words (e.g., 'the', 'is') that don't carry significant meaning.
4. **Stemming**: Reducing words to their root form using the PorterStemmer, e.g., converting 'running' to 'run'.

```
python
Copy code
def preprocess_text(text):
    tokens = word_tokenize(text)
    tokens = [word.lower() for word in tokens]
    tokens = [word for word in tokens if word not in
stopwords.words('english')]
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]
    return ' '.join(tokens)

df['clean_message'] = df['message'].apply(preprocess_text)
```

The `preprocess_text` function is applied to each message in the dataset using the `apply()` method, resulting in a new column 'clean_message', which contains the preprocessed text.

---

6. Feature Extraction using TF-IDF

```
python
Copy code
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['clean_message'])
y = df['label'].apply(lambda x: 1 if x == 'spam' else 0)
```

In this step, we convert the cleaned text data into numerical features using **TF-IDF (Term Frequency-Inverse Document Frequency)**. TF-IDF is a widely used feature extraction technique in NLP that reflects the importance of a word in a document relative to a collection of documents (corpus).

The target variable `y` is converted into binary values: spam is labeled as 1 and ham as 0.

---

7. Splitting the Dataset

```
python
Copy code
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

The dataset is split into training (70%) and test (30%) sets using the `train_test_split()` method. This ensures that we can evaluate the model's performance on unseen data.

---

8. Model Training using Naive Bayes

```
python
Copy code
model = MultinomialNB()
model.fit(X_train, y_train)
```

We train a **Multinomial Naive Bayes** model, which is well-suited for text classification problems. Naive Bayes models assume that the features (words) are conditionally independent given the class (spam or ham), making it computationally efficient and effective for high-dimensional data.

---

9. Model Evaluation

After training, we evaluate the model using several metrics:

```
python
Copy code
y_pred = model.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

The predictions (`y_pred`) are made on the test set, and we compute the **confusion matrix** to visualize the model's performance in terms of true positives, false positives, true negatives, and false negatives.

The **classification report** provides a summary of precision, recall, F1-score, and accuracy for both classes (ham and spam). We also calculate and print the accuracy:

```
python
Copy code
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Finally, we visualize the confusion matrix using a heatmap:

```
python
Copy code
labels = ['Ham', 'Spam']
cm = confusion_matrix(y_test, y_pred)
```

```
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=labels,
yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

---

10. Additional Exploratory Data Analysis (EDA)

We perform additional visualizations to gain insights into the words that frequently appear in spam and ham messages using **Word Clouds**.

```
python
Copy code
from wordcloud import WordCloud
spam_messages = ' '.join(df[df['label'] == 'spam']['clean_message'])
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(spam_messages)
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Spam Messages')
plt.show()

ham_messages = ' '.join(df[df['label'] == 'ham']['clean_message'])
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(ham_messages)
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Ham Messages')
plt.show()
```

Word clouds provide a visual representation of the most frequent words in spam and ham messages. Larger words are more frequent, giving us insights into common patterns in the messages.

Conclusion

This project demonstrates how to build a spam filtering model using a Naive Bayes classifier and NLP techniques. The model achieves good accuracy by effectively preprocessing the text data, extracting meaningful features using TF-IDF, and evaluating the performance using multiple metrics.