Spam Detection

Objective

The objective of this project is to build a machine learning model that can classify text messages as either spam or ham (non-spam). The model uses the Multinomial Naive Bayes algorithm, which is effective for text classification tasks.

Tools:

This project is built using the following Python libraries:

- **pandas**: For data manipulation and analysis.
- **numpy**: For numerical operations.
- **matplotlib**: For data visualization.
- **seaborn**: For advanced data visualization.
- **scikit-learn**: For machine learning models and utilities like train-test split and vectorization.
- **re**: For text preprocessing using regular expressions.

Data Source

The dataset used for this project is a CSV file named spam.csv that contains text messages labeled as "ham" (non-spam) or "spam." It is assumed that the file is hosted or uploaded within the working directory.

The dataset has the following columns:

1. **v1**: The category (ham/spam).
2. **v2**: The message content.

The CSV file is read and processed to prepare it for model training.

1. Importing Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

These libraries are essential for data handling, numerical operations, visualization, and machine learning tasks.

2. Reading the Data

data = pd.read_csv('/content/spam.csv', encoding='ISO-8859-1')

The dataset is read into a pandas DataFrame. The encoding 'ISO-8859-1' is used to handle potential character encoding issues.

3. Data Inspection

print(data.head())

Displays the first few rows of the dataset to understand its structure and check the data quality.

4. Data Cleaning

- **Removing Unnecessary Columns:**

data = data.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'])

Columns that are unnecessary or empty (such as Unnamed: 2, Unnamed: 3, Unnamed: 4) are dropped from the dataset.

- **Renaming Columns:**

data = data.rename(columns={'v1': 'Category', 'v2': 'Message'})

The columns are renamed for better clarity: v1 becomes Category and v2 becomes Message.

5. Exploring Category Distribution

print(data['Category'].value_counts())

The Category column (ham/spam) is analyzed to check the distribution of labels.

6. Mapping Labels to Numeric Values

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

The labels "ham" and "spam" are converted to numeric values (0 for ham and 1 for spam), as machine learning models work better with numerical data.

7. Handling Missing Data

print(data.isnull().sum())

The code checks for missing values in the dataset.

8. Removing Duplicates

data = data.drop_duplicates()

Duplicate rows are removed to ensure that the model trains on unique data.

9. Text Preprocessing

import re

data['Message'] = data['Message'].apply(lambda x: re.sub(r'\W+', ' ', x.lower()))

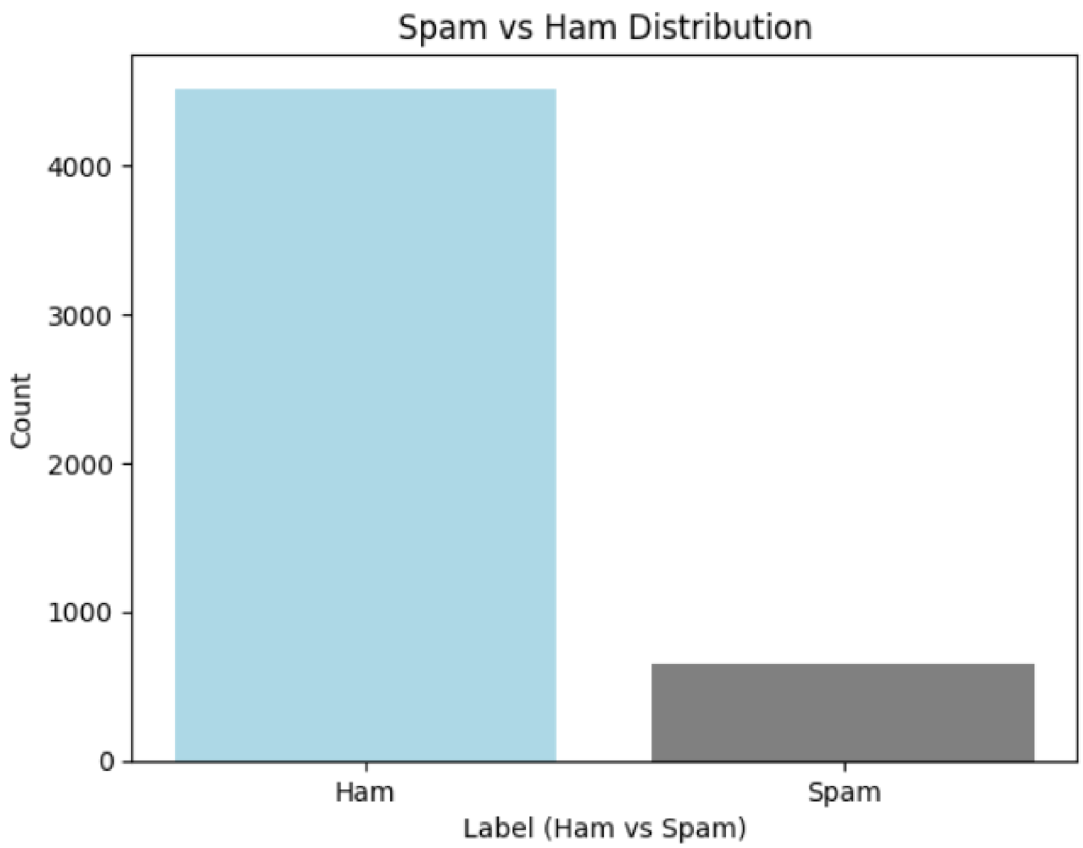
A regular expression (re.sub) is used to remove all non-word characters (e.g., punctuation) and convert the text to lowercase. To standardize the text data.

10. Exploratory Data Analysis (EDA)

- **Distribution of Spam vs. Ham:**

```
label_counts = data['Category'].value_counts()
plt.bar(['Ham', 'Spam'], label_counts, color=['lightblue', 'grey'])
plt.title('Spam vs Ham Distribution')
plt.xlabel('Label (Ham vs Spam)')
plt.ylabel('Count')
plt.show()
```

A bar plot is created to visualize the distribution of spam and ham messages in the dataset.



- **Average Word Count by Category:**

data['word_count'] = data['Message'].apply(lambda x: len(x.split()))

word_count_by_category = data.groupby('Category')['word_count'].mean()

plt.bar(['Ham', 'Spam'], word_count_by_category.values, color='lightblue')

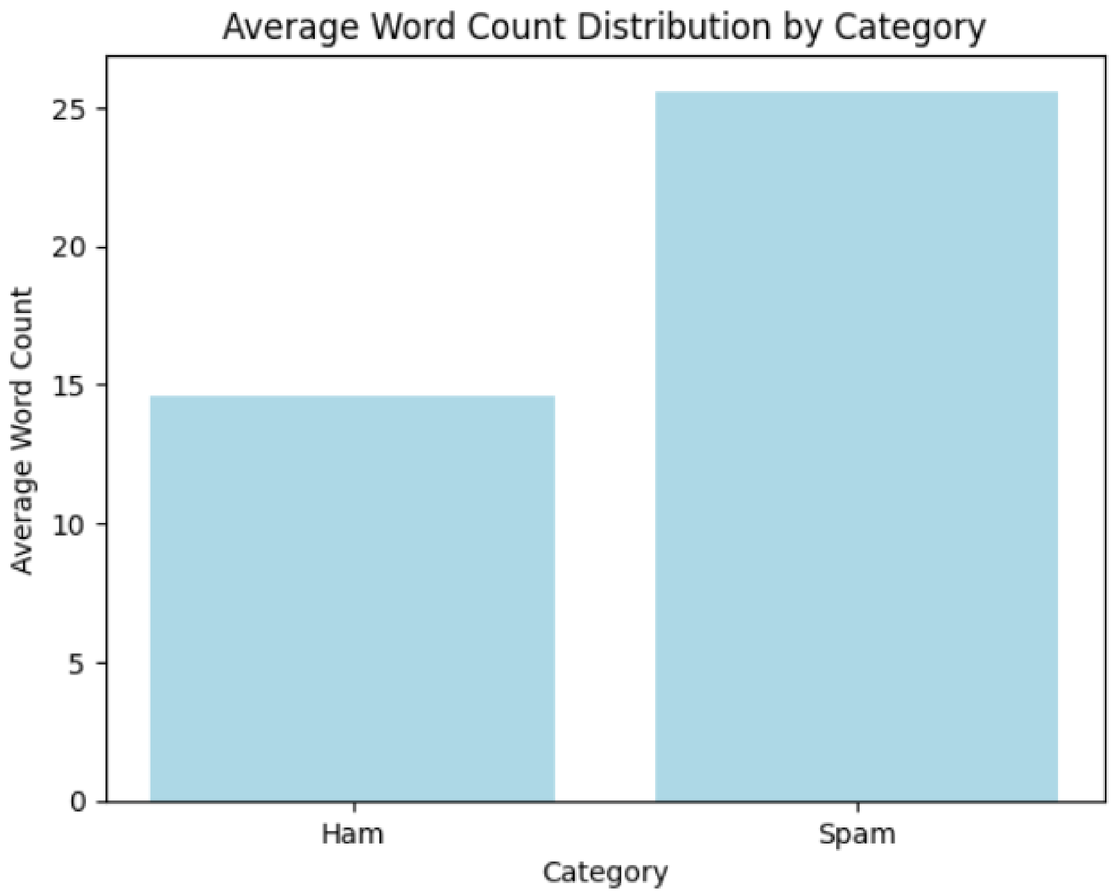
plt.xlabel('Category')

plt.ylabel('Average Word Count')

plt.title('Average Word Count Distribution by Category')

plt.show()

The average word count of messages in each category (spam and ham) is plotted to observe if there is a significant difference.



11. Text Vectorization

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=5000)

X = tfidf.fit_transform(X).toarray()

The text data is transformed into numerical data using the **TF-IDF** (Term Frequency-Inverse Document Frequency) vectorizer, limiting to the 5000 most important words.

12. Data Splitting and Train-Test Split

X = data['Message']

y = data['Category']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

The dataset is split into training and testing sets, with 20% of the data used for testing.

13. Model Training

from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()

model.fit(X_train, y_train)

The *Multinomial Naive Bayes* classifier is trained on the training data.

14. Model Evaluation

from sklearn.metrics import accuracy_score, confusion_matrix

print("Accuracy:", accuracy_score(y_test, y_pred))

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

The accuracy of the model is evaluated, and a confusion matrix is generated to show the number of correct and incorrect predictions.

- **Confusion Matrix Visualization:**

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

A heatmap is created to visually represent the confusion matrix.

Conclusion

- The model was trained and tested on a spam message dataset.
- The Multinomial Naive Bayes classifier performed well, as shown by the accuracy and confusion matrix.
- Future improvements could involve experimenting with different models, hyperparameter tuning, or incorporating more complex text preprocessing techniques.