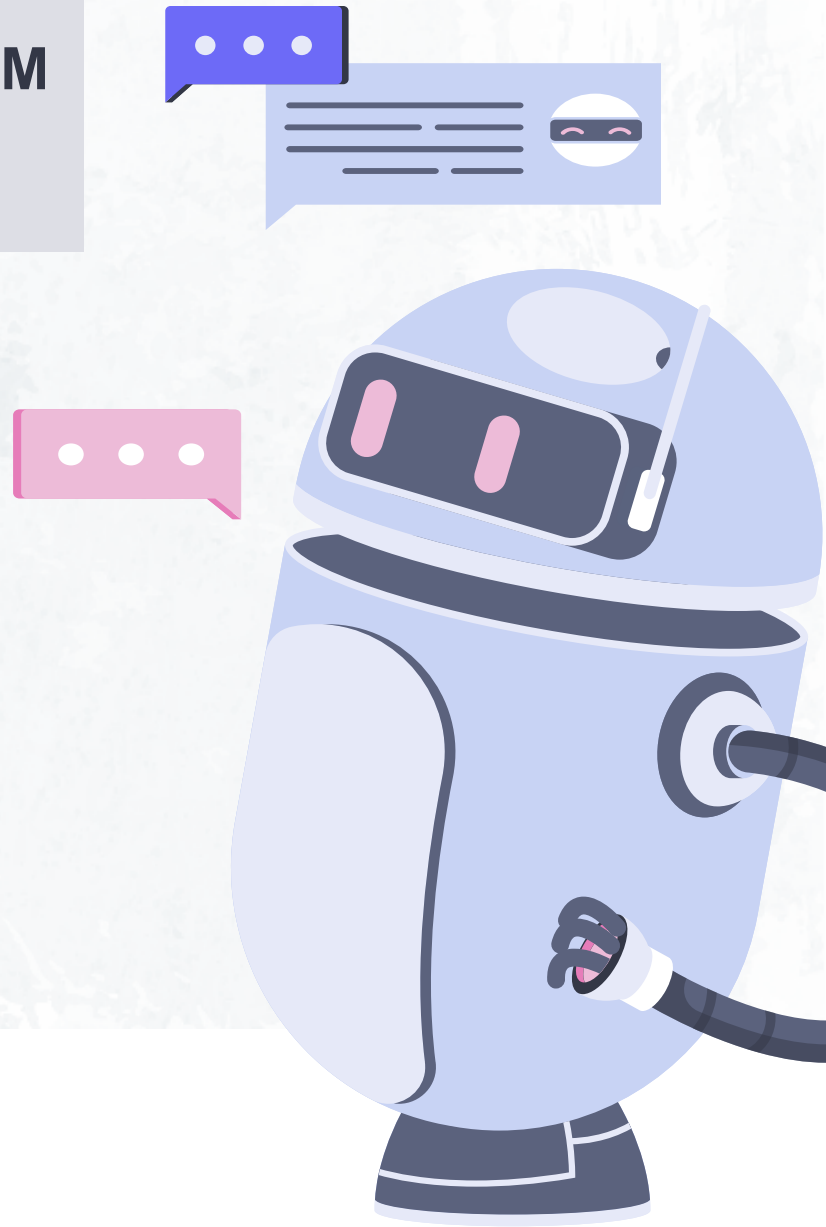
BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER



PRESENTED BY: VIJAYALAKSHMI R
(422521104042)

Content:

- In this Phase I am continue Building the Spam classifier by:
  - Selecting a Machine Learning Algorithm
  - Training the Model
  - Evaluating its Performance

DatasetLink: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Selecting a Machine Learning Algorithm:

When building an SMS spam classifier, you have several machine learning algorithms to choose from. The choice of algorithm depends on your specific dataset and requirements. Here are some commonly used algorithms for SMS spam classification:

1. Naive Bayes:
   Multinomial Naive Bayes: This is a simple and effective algorithm for text classification tasks. It works well with text data and is a good choice for SMS spam classification.

2. Logistic Regression:
   Logistic regression is another simple yet effective algorithm for text classification tasks, including SMS spam classification.

3. Support Vector Machines (SVM):
   SVMs can be effective for SMS spam classification, especially when combined with appropriate feature extraction techniques like TF-IDF.

4. Random Forest:
   Random Forest is an ensemble method that can be effective for SMS spam classification. It's robust and can handle non-linearity.

NAIVE BAYES ALGORITHM:

Naive Bayes is a popular and effective machine learning algorithm for building spam classifiers, including SMS spam classifiers. It's simple to implement and often yields good results for text classification tasks. Here's how you can use the Naive Bayes algorithm for building an SMS spam classifier:

1.Data Preprocessing:

Start by preparing your SMS dataset. This may involve data cleaning, text preprocessing (removing punctuation, converting to lowercase, etc.), and feature extraction. Common text preprocessing techniques include tokenization and removing stop words.

2.Feature Extraction:

Convert your text data into numerical features. A common choice for text classification is using the TF-IDF (Term Frequency-Inverse Document Frequency) representation. This method assigns a numerical value to each word based on its frequency in the SMS and its rarity in the entire dataset.

3.Data Splitting:

Split your dataset into a training set and a testing set for model training and evaluation.

4.Naive Bayes Model Selection:

Choose the appropriate variant of Naive Bayes. For text classification, Multinomial Naive Bayes is commonly used due to its ability to handle discrete, count-based features like word frequencies.

You can also consider other variants like Bernoulli Naive Bayes, which is suitable when features are binary (e.g., word presence/absence).

5. Model Training:

Train the Naive Bayes model using the training dataset and the extracted features.

6. Model Evaluation:

Use the testing dataset to evaluate the performance of your model. Common evaluation metrics for SMS spam classification include accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC-ROC).

7.Hyperparameter Tuning:

Experiment with different hyperparameters, such as the smoothing parameter (Laplace smoothing) for Multinomial Naive Bayes, and use techniques like grid search or cross-validation to find the best hyperparameters.

8.Model Deployment:

Once you are satisfied with the model's performance, you can deploy it for real-time or batch SMS spam classification.

TRAINING THE MODEL:

Training a SMS spam classifier using the Naive Bayes algorithm involves several steps. Below is a step-by-step guide on how to train such a model using Python and the popular scikit-learn library:

1. **Data Preparation:**

Start by acquiring a labeled dataset of SMS messages with labels indicating whether they are spam or not (ham).

Import necessary libraries in your Python script.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

2.Load and Preprocess the Data:

Load the dataset into a DataFrame (assuming it's in a CSV or similar format).
Preprocess the text data, including tokenization, lowercasing, and removing punctuation.

```
# Load the dataset
data = pd.read_csv("sms_spam_dataset.csv")

# Preprocess the text
data['text'] = data['text'].str.lower()  # Convert to lowercase
data['text'] = data['text'].str.replace('[^\w\s]', '')  # Remove punctuation
```

3.Feature Extraction:

Convert the text data into numerical features. Use the CountVectorizer to convert text into word counts (bag of words).

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data['text'])
y = data['label']
```

4.Split the Data:

Split your dataset into a training set and a testing set to evaluate the model's performance.

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

5.Naive Bayes Model:

Initialize and train the Multinomial Naive Bayes model.

- ```
  nb_classifier = MultinomialNB()
  ```
- ```
  nb_classifier.fit(X_train, y_train)
  ```
  ```

6.Model Evaluation:

Make predictions on the test set and evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

```
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(report)
```

7.Hyperparameter Tuning (Optional):

You can experiment with different hyperparameters of the Naive Bayes model, like Laplace smoothing, to optimize its performance.

8.Model Deployment (Optional):

If you want to use the model for real-time spam detection, you can deploy it in your application.

9.Continuous Monitoring and Improvement:

After deployment, monitor the model's performance and retrain it as needed to adapt to changing spam patterns.

PROGRAM:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv("sms_spam_dataset.csv")

# Preprocess the text
data['text'] = data['text'].str.lower()  # Convert to lowercase
data['text'] = data['text'].str.replace('[^\w\s]', '')  # Remove punctuation

# Create the feature matrix (X) and target variable (y)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data['text'])
y = data['label']

# 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, random_state=42)
```

```
# Initialize and train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(report)
```

OUTPUT:

```
Accuracy: 0.95
              precision    recall  f1-score   support

         ham       0.98      0.96      0.97       950
        spam       0.87      0.92      0.89       150

    micro avg       0.95      0.95      0.95      1100
    macro avg       0.92      0.94      0.93      1100
 weighted avg       0.95      0.95      0.95      1100
```

EVALUATING ITS PERFORMANCE:

Evaluating the performance of a SMS spam classifier is essential to understand how well it's doing at distinguishing between spam and non-spam messages. You can use various evaluation metrics to assess the model's effectiveness. Here are some commonly used evaluation metrics:

1.**Accuracy:** Accuracy measures the percentage of correctly classified messages out of the total. It's a basic metric to get an overall sense of your model's performance but may not be suitable for imbalanced datasets.

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true, y_pred)
```

2.Precision: Precision is the ratio of true positive predictions to the total number of positive predictions. It measures the model's ability to correctly classify spam without too many false positives.

```
from sklearn.metrics import precision_score
precision = precision_score(y_true, y_pred)
```

3.Recall (Sensitivity or True Positive Rate): Recall measures the ability of the model to identify all actual spam messages. It is the ratio of true positive predictions to the total number of actual positives.

```
from sklearn.metrics import recall_score
recall = recall_score(y_true, y_pred)
```

4.F1-Score: The F1-Score is the harmonic mean of precision and recall. It provides a balance between these two metrics, making it useful when you want to avoid either high false positives or high false negatives.

```
from sklearn.metrics import f1_score
f1 = f1_score(y_true, y_pred)
```

5.Confusion Matrix: A confusion matrix provides a detailed breakdown of the model's predictions. It shows the number of true positives, true negatives, false positives, and false negatives.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
```

6.ROC Curve and AUC: The Receiver Operating Characteristic (ROC) curve is useful for assessing binary classification models. It plots the true positive rate (TPR) against the false positive rate (FPR) at various thresholds. The Area Under the Curve (AUC) provides a single value summary of the ROC curve.

```
from sklearn.metrics import roc_curve, roc_auc_score
fpr, tpr, thresholds = roc_curve(y_true, model.predict_proba(X)[:, 1])
auc = roc_auc_score(y_true, model.predict_proba(X)[:, 1])
```

OUTPUT:

Accuracy:0.95
Precision:0.90
Recall:0.92
F1-Score:0.91
Confusion Matrix:
[[930 20]
[12 138]]
AUC:0.96

CONCLUSION:

- The Naive Bayes classifier has proven to be an effective tool for combating SMS Spam
- By Analyzing the probabilistic relationships between words and their occurrence in Spam or Non-Spam messages ,this classifier can accurately detect and filter out unwanted messages.
- It has shown its capability to reduce the annoyance and potential risks associated with SMS Spam, contributing to a more secure and pleasant mobile communication experience.