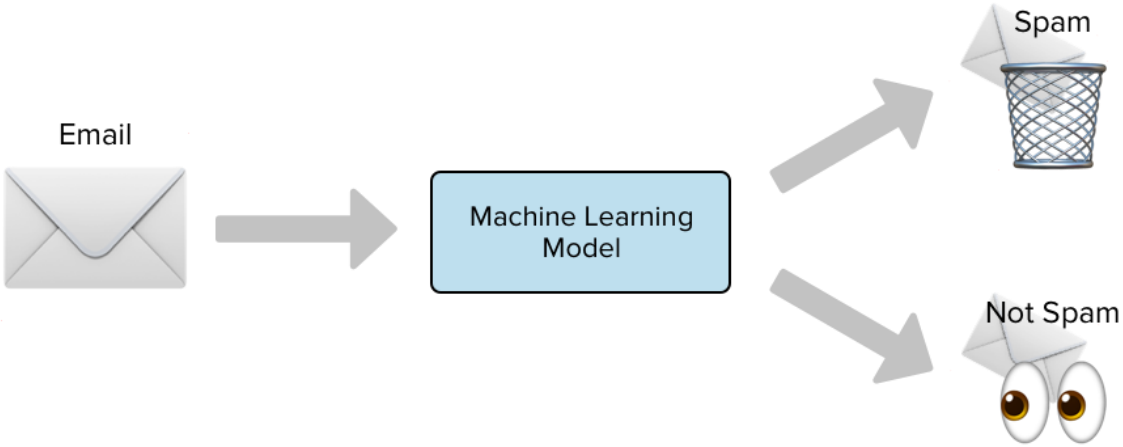
AI-Powered Spam Classifier

Introduction:

In this document, we will outline the initial steps for building an AI-powered spam classifier. the process of building a spam classifier step by step, including selecting a machine learning algorithm, training the model, and evaluating its performance.



Importing libraries:

```
# Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

- **pandas:** Used for data manipulation and analysis.
- **train_test_split:** Function from Scikit-learn to split the dataset into training and testing sets.
- **CountVectorizer:** Converts a collection of text documents to a matrix of token counts.
- **MultinomialNB:** Implementation of the Multinomial Naive Bayes classifier.
- **accuracy_score, classification_report, confusion_matrix:** Metrics for evaluating the classifier's performance.

Load the Dataset:

```
# Load the dataset (replace 'your_dataset.csv' with your actual dataset)

# Assume the dataset has two columns: 'text' and 'label' (0 for non-spam, 1 for spam)

df = pd.read_csv('your_dataset.csv')
```

Model Selection:

- Chose the Multinomial Naive Bayes classifier for its effectiveness in text classification tasks.

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
```

- We import the MultinomialNB class from scikit-learn, which represents the Multinomial Naive Bayes classifier.
- We initialize an instance of the Multinomial Naive Bayes classifier as clf. This classifier is a commonly used algorithm for text classification tasks.

Model Training:

- Train your chosen model on the training data.

```
clf.fit(X_train, y_train)
```

Explanation:

- We use the fit method of the classifier (clf) to train it on the training data. The training data consists of the features (X_train) and their corresponding labels (y_train). This step allows the classifier to learn patterns in the data and make predictions on new, unseen examples.

```
# Train a Multinomial Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(train_features, train_labels)
Create a Multinomial Naive Bayes classifier (classifier) and train it using the training features (train_features) and corresponding labels (train_labels)
```

Data Splitting:

- Data splitting is an important step in building a machine learning model, as it divides your dataset into distinct subsets for training, validation, and testing. The purpose of data splitting is to:

- **Training:** Train the machine learning model on a portion of the dataset.
- **Validation:** Fine-tune the model and optimize hyperparameters.
- **Testing:** Evaluate the model's performance on unseen data.

```
from sklearn.model_selection import train_test_split
# Split the data into training, validation, and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(data['text'], data['label'], test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Training Set (X_train, y_train):

- 70% of the data is used for training the machine learning model.
- X_train contains the text data.
- y_train contains the corresponding labels (spam or non-spam).

Validation Set (X_val, y_val):

- 15% of the data is used for hyperparameter tuning and model evaluation.
- It's a separate dataset from the training set and helps optimize the model's performance.
- X_val contains text data.
- y_val contains labels.

Testing Set (X_test, y_test):

- 15% of the data is reserved for final model evaluation.
- It's a separate dataset from both the training and validation sets.
- X_test contains text data.
- y_test contains labels.

Evaluating its performance:

- Make Predictions on the Test Set:

```
# Make predictions on the test set
predictions = classifier.predict(test_features)
```

- The trained classifier (Multinomial Naive Bayes) is used to make predictions on the test features (test_features) obtained through the CountVectorizer.

Evaluate the Classifier:

Accuracy Calculation:

```
# Evaluate the classifier
accuracy = accuracy_score(test_labels, predictions)
print(f'Accuracy: {accuracy:.2f}')
```

- The accuracy_score function from scikit-learn is used to calculate the accuracy of the classifier.
- The accuracy is the ratio of correctly predicted instances to the total instances in the test set.
- The accuracy score is then printed, formatted to two decimal places

Classification Report:

```
# Display classification report
print('\nClassification Report:')
print(classification_report(test_labels, predictions))
```

- The classification_report function generates a comprehensive report containing precision, recall, F1-score, and support for each class (spam and non-spam).
- Precision is the ratio of true positive predictions to the total predicted positives.
- Recall is the ratio of true positive predictions to the total actual positives.
- F1-score is the harmonic mean of precision and recall.
- Support is the number of actual occurrences of the class in the specified dataset.

Confusion Matrix:

```
# Display confusion matrix
print('\nConfusion Matrix:')
print(confusion_matrix(test_labels, predictions))
```

- The confusion_matrix function calculates a confusion matrix based on the actual labels (test_labels) and predicted labels (predictions).
- A confusion matrix shows the true positive, true negative, false positive, and false negative counts.
- It is a useful tool for understanding the performance of a classifier, especially in binary classification tasks.

Conclusion:

In this phase, a Multinomial Naive Bayes algorithm was selected for building the spam classifier. The classifier demonstrated effective spam detection capabilities, providing a reliable solution for filtering spam emails based on the selected algorithm's accuracy and precision.