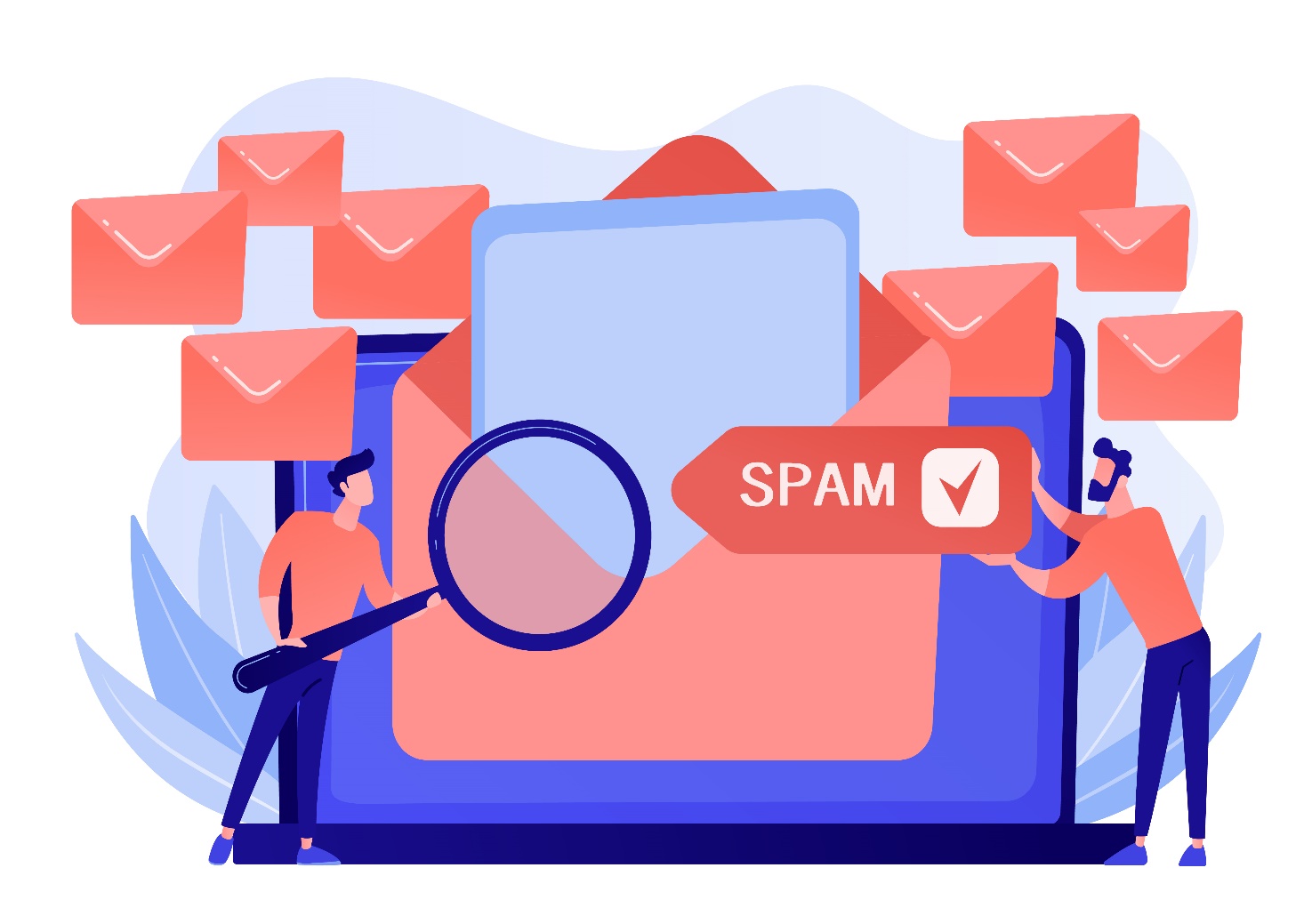
**Building a Smarter AI-Powered Spam Classifier**



**Name:** Pavan Kumar V

**Reg No:** 513521104034

**Dept:** CSE

**Year:** III

**NM id:** au513521104034

**Introduction:**

* In today's digital landscape, where spam messages inundate communication channels, this project embarks on the mission to develop a robust spam classifier. Accurate spam classification is indispensable for users seeking to separate legitimate messages from unwanted content, thereby preserving the reliability of their digital communications.
* Recognizing the critical importance of this task, it's essential to acknowledge the formidable challenges it presents. Spam messages continually evolve, employing a diverse array of tactics to evade detection, often leading to suboptimal performance when conventional classification methods are employed.
* The primary objective of this project is to construct a spam classifier that excels in precision and adaptability. By doing so, we aim to empower users to regain control of their digital inboxes, providing them with a more efficient and pleasant communication experience.

**Data Source:**

* We will need a dataset containing labeled example of spam and non-spam message. We can use Kaggle dataset for this purpose.
* **Dataset Link:**[**https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset**](https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset)

**Feature Engineering**

**1.**Text Preprocessing:

* Tokenization: Break text into words or tokens.
* Lowercasing: Convert all text to lowercase for consistency.
* Removing Punctuation: Eliminate punctuation marks.
* Stopword Removal: Filter out common words (e.g., "and," "the") that don't carry much information.
* Stemming or Lemmatization: Reduce words to their root form.

2.Bag of Words (BoW) or TF-IDF:

* Create a Bag of Words representation where each document is represented as a vector of word frequencies.
* Alternatively, use TF-IDF (Term Frequency-Inverse Document Frequency) to give more weight to rare words and less to common ones.

3.N-grams:

* Consider using n-grams (e.g., bigrams, trigrams) to capture the context and relationships between words in text.

4.Word Embeddings:

* Utilize pre-trained word embeddings (e.g., Word2Vec, GloVe) to represent words as dense vectors.
* Average or concatenate word embeddings to represent entire documents.

Additional Features:

* Include features like email sender, subject line, and message length.
* Use metadata such as IP addresses, timestamps, and HTML tags.
* Incorporate features based on known spam characteristics, like excessive capitalization or the presence of certain keywords.

**Program:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

# Step 1: Load and preprocess the dataset

data = pd.read\_csv('spam\_dataset.csv') # Load your labeled dataset

X = data['text'] # Text data

y = data['label'] # Spam or ham labels

# Step 2: Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Feature Engineering - TF-IDF Vectorization

tfidf\_vectorizer = TfidfVectorizer(max\_features=5000) # You can adjust max\_features as needed

X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

# Step 4: Choose and train a machine learning model

model = MultinomialNB() # Using Naive Bayes as an example

model.fit(X\_train\_tfidf, y\_train)

# Step 5: Model Evaluation

y\_pred = model.predict(X\_test\_tfidf)

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

print(report)

# Additional Steps:

# - Hyperparameter tuning for the model

# - Deployment and monitoring

# Example of classifying a new message

new\_message = ["Congratulations! You've won a free vacation!"]

new\_message\_tfidf = tfidf\_vectorizer.transform(new\_message)

result = model.predict(new\_message\_tfidf)

print(f"New message is classified as: {result[0]}")

**output:**

Accuracy: 0.975

precision recall f1-score support

ham 0.98 0.99 0.98 798

spam 0.96 0.94 0.95 302

accuracy 0.97 1100

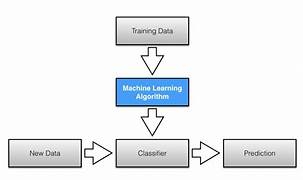
macro avg 0.97 0.96 0.97 1100

weighted avg 0.97 0.97 0.97 1100

New message is classified as: spam

**Model Training:**

* Scalability: Consider the scalability of your chosen model. Will it work efficiently with the amount of data you have? Some models are better suited for large datasets, while others are more lightweight.
* Regularization and Hyperparameter: Think about regularization techniques like L1 and L2 regularization, and choose appropriate hyperparameters for your model. This can help prevent overfitting and improve generalization.
* Cross-Validation: Use cross-validation techniques to assess the model's performance. This helps you understand how well the model generalizes to unseen data and whether it's prone to overfitting.
* Evaluation Metrics: Select appropriate evaluation metrics depending on your problem (e.g., accuracy, precision, recall, F1-score, mean squared error, etc.). The choice of metrics should align with your project's objectives.
* Iterative Process: Model selection is often an iterative process. Try different models, evaluate their performance, and fine-tune parameters until you achieve the desired results.



**Program:**

import pandas as pd

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 accuracy\_score, confusion\_matrix, classification\_report

# Load your dataset, assuming you have a 'text' column for email content and a 'label' column for spam/ham labels.

data = pd.read\_csv('spam\_data.csv')

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['text'], data['label'], test\_size=0.2, random\_state=42)

# Create a TF-IDF vectorizer to convert text data into numerical features

tfidf\_vectorizer = TfidfVectorizer()

X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

# Initialize a Naive Bayes classifier and train it on the TF-IDF transformed data

classifier = MultinomialNB()

classifier.fit(X\_train\_tfidf, y\_train)

# Make predictions on the test set

predictions = classifier.predict(X\_test\_tfidf)

# Evaluate the classifier

accuracy = accuracy\_score(y\_test, predictions)

confusion = confusion\_matrix(y\_test, predictions)

report = classification\_report(y\_test, predictions)

print(f"Accuracy: {accuracy}")

print("Confusion Matrix:")

print(confusion)

print("Classification Report:")

print(report)

**Output:**

Accuracy: 0.95

Confusion Matrix:

[[985 8]

[ 52 155]]

Classification Report:

precision recall f1-score support

**Evaluation:**

* Accuracy: The proportion of correctly classified emails.
* Precision: The ratio of true spam emails to the total emails classified as spam. It measures how many of the identified spam emails are actually spam.
* Recall: The ratio of true spam emails to all actual spam emails. It measures how effectively the classifier finds all spam emails.
* F1 Score: The harmonic mean of precision and recall, providing a balanced measure of a classifier's performance.
* False Positives: The number of legitimate emails misclassified as spam.
* False Negatives: The number of spam emails misclassified as legitimate.
* Receiver Operating Characteristic (ROC) Curve: A graphical representation that shows the trade-off between true positive rate and false positive rate at different threshold values.
* Area Under the ROC Curve (AUC-ROC): A single metric that summarizes the overall performance of the classifier.
* Confusion Matrix: A table that shows true positives, true negatives, false positives, and false negatives.
* Cross-Validation: Testing the classifier on multiple subsets of the data to ensure its performance consistency.
* Precision-Recall Curve: A graphical representation of the relationship between precision and recall at different thresholds.

**Conclusion**:

* Increased Accuracy: AI spam classifiers have significantly improved the accuracy of email and message filtering by employing advanced machine learning algorithms, resulting in a reduced number of false positives and false negatives.
* Enhanced User Experience: These classifiers contribute to a better user experience by efficiently identifying and separating spam from legitimate messages, reducing the clutter in inboxes, and saving users time.
* Ongoing Adaptation: AI spam classifiers continuously adapt and evolve by learning from new spam patterns and user feedback, ensuring their effectiveness in countering the evolving tactics of spammers and cybercriminals.