**Email Spam Detection with Machine Learning**

**Objective**

The objective of this project is to develop a machine learning model to detect and classify emails as spam or non-spam (ham). Spam emails often contain unwanted advertisements, phishing content, and other forms of unsolicited messages that can be harmful to recipients. The goal is to build an efficient and accurate spam detector using Python and machine learning techniques.

**Solution**

We propose to use supervised machine learning techniques to train a model that can classify emails as spam or ham. The solution involves preprocessing the email text data, extracting relevant features, training various machine learning models, tuning their hyperparameters, and evaluating their performance. We will also create additional features to improve the model's accuracy.

**Procedure**

**Step 1: Data Collection**

The dataset used in this project is a collection of labeled emails, with each email classified as either spam or ham. The dataset is provided in a CSV file.

**Step 2: Data Preprocessing**

1. **Loading the Dataset**:
   * Load the dataset using pandas.
   * Drop unnecessary columns and handle missing values.
2. **Data Cleaning**:
   * Rename columns for better understanding.
   * Convert labels to binary format: 'spam' = 1 and 'ham' = 0.
3. **Feature Engineering**:
   * Create features such as email length, number of special characters, and number of uppercase words.

**Step 3: Text Feature Extraction**

1. **TF-IDF Vectorization**:
   * Use TfidfVectorizer to convert the text data into numerical features based on Term Frequency-Inverse Document Frequency (TF-IDF).

**Step 4: Model Building**

1. **Pipeline Creation**:
   * Create a pipeline to streamline preprocessing and model training.
   * Use ColumnTransformer to combine TF-IDF features with additional features.
2. **Model Selection**:
   * Evaluate multiple machine learning models: Naive Bayes, Logistic Regression, Random Forest, and Support Vector Machine (SVM).
   * Use GridSearchCV for hyperparameter tuning.

**Step 5: Model Evaluation**

1. **Performance Metrics**:
   * Evaluate models using accuracy, precision, recall, F1-score, and confusion matrix.

**Step 6: Conclusion**

Summarize the findings, including the best performing model and its performance metrics.

**Related Theory and Algorithms**

**Term Frequency-Inverse Document Frequency (TF-IDF)**

TF-IDF is a numerical statistic that reflects the importance of a word in a document relative to a collection of documents (corpus). It is calculated as: TF-IDF(t,d)=TF(t,d)×IDF(t)\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)TF-IDF(t,d)=TF(t,d)×IDF(t) where:

* TF(t,d)\text{TF}(t,d)TF(t,d) is the term frequency of term ttt in document ddd.
* IDF(t)\text{IDF}(t)IDF(t) is the inverse document frequency of term ttt.

**Support Vector Machine (SVM)**

SVM is a supervised learning algorithm used for classification and regression tasks. It finds the optimal hyperplane that maximizes the margin between different classes in the feature space. Key parameters include:

* CCC: Regularization parameter.
* γ\gammaγ: Kernel coefficient for 'rbf', 'poly', and 'sigmoid' kernels.
* kernel\text{kernel}kernel: Specifies the kernel type to be used in the algorithm.

**Code Execution**

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.svm import SVC

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import MinMaxScaler

from sklearn.pipeline import Pipeline

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

# Load the dataset

data = pd.read\_csv('spam.csv', encoding='latin-1')

# Drop unnecessary columns and rename the columns for better understanding

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

data = data.rename(columns={'v1': 'label', 'v2': 'text'})

data = data.dropna()

# Convert labels to binary format: 'spam' = 1 and 'ham' = 0

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

# Feature: Length of the email

data['email\_length'] = data['text'].apply(len)

# Feature: Number of special characters

data['special\_chars'] = data['text'].apply(lambda x: sum([1 for char in x if not char.isalnum() and not char.isspace()]))

# Feature: Number of uppercase words

data['uppercase\_words'] = data['text'].apply(lambda x: sum([1 for word in x.split() if word.isupper()]))

# Split the data into features and labels

X = data[['text', 'email\_length', 'special\_chars', 'uppercase\_words']]

y = data['label']

# Split the data 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 the TF-IDF Vectorizer with the best parameters

vectorizer = TfidfVectorizer(max\_df=0.75, min\_df=1, ngram\_range=(1, 1))

# Combine TF-IDF features with additional features using MinMaxScaler

preprocessor = ColumnTransformer(

    transformers=[

        ('tfidf', vectorizer, 'text'),

        ('num', MinMaxScaler(), ['email\_length', 'special\_chars', 'uppercase\_words'])

    ],

    remainder='drop'

)

# Define the parameter grid for SVM

svm\_params = {

    'model\_\_C': [0.1, 1, 10, 100],

    'model\_\_gamma': [1, 0.1, 0.01, 0.001],

    'model\_\_kernel': ['linear', 'rbf']

}

# Create a pipeline with the preprocessor and SVM

pipeline = Pipeline([

    ('preprocessor', preprocessor),

    ('model', SVC())

])

# Initialize GridSearchCV

grid\_search = GridSearchCV(pipeline, svm\_params, cv=5, scoring='accuracy', n\_jobs=-1, verbose=2)

# Fit GridSearchCV

grid\_search.fit(X\_train, y\_train)

# Get the best parameters and best score

best\_params = grid\_search.best\_params\_

best\_score = grid\_search.best\_score\_

print("Best Parameters:", best\_params)

print("Best Score:", best\_score)

# Remove the 'model\_\_' prefix

best\_params = {k.replace('model\_\_', ''): v for k, v in best\_params.items()}

# Train and evaluate the model with the best parameters

pipeline = Pipeline([

    ('preprocessor', preprocessor),

    ('model', SVC(\*\*best\_params))

])

pipeline.fit(X\_train, y\_train)

y\_pred = pipeline.predict(X\_test)

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

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Classification Report:\n", report)

print("Confusion Matrix:\n", conf\_matrix)

**Output**

| **v1** | **v2** | **Unnamed: 2** | **Unnamed: 3** | **Unnamed: 4** |
| --- | --- | --- | --- | --- |
| **0** | ham | Go until jurong point, crazy.. Available only ... | NaN | NaN | NaN |
| **1** | ham | Ok lar... Joking wif u oni... | NaN | NaN | NaN |
| **2** | spam | Free entry in 2 a wkly comp to win FA Cup fina... | NaN | NaN | NaN |
| **3** | ham | U dun say so early hor... U c already then say... | NaN | NaN | NaN |
| **4** | ham | Nah I don't think he goes to usf, he lives aro... | NaN | NaN | NaN |

**Distributions**

A graph of blue bars

Description automatically generated with medium confidence

**Categorical distributions**

A graph with a rectangle and a green rectangle

Description automatically generated

A close-up of a chart

Description automatically generated

**Time series**

A line on a white background

Description automatically generated

A white background with black text and black text

Description automatically generated

**Values**

A blue line graph with white text

Description automatically generated

**2-d categorical distributions**

A blue rectangle with black lines

Description automatically generated

**Faceted distributions**

<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

A green rectangular object with a black line

Description automatically generated

<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

A close-up of a piece of paper

Description automatically generated

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 5572 entries, 0 to 5571

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 v1 5572 non-null object

1 v2 5572 non-null object

2 Unnamed: 2 50 non-null object

3 Unnamed: 3 12 non-null object

4 Unnamed: 4 6 non-null object

dtypes: object(5)

memory usage: 217.8+ KB

|  | **v1** | | **v2** | | **Unnamed: 2** | | **Unnamed: 3** | **Unnamed: 4** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | ham | | Go until jurong point, crazy.. Available only ... | | NaN | | NaN | NaN |
| **1** | ham | | Ok lar... Joking wif u oni... | | NaN | | NaN | NaN |
| **2** | spam | | Free entry in 2 a wkly comp to win FA Cup fina... | | NaN | | NaN | NaN |
| **3** | ham | | U dun say so early hor... U c already then say... | | NaN | | NaN | NaN |
| **4** | ham | | Nah I don't think he goes to usf, he lives aro... | | NaN | | NaN | NaN |
| **label** | | **text** | |
| **0** | | 0 | | Go until jurong point, crazy.. Available only ... | |
| **1** | | 0 | | Ok lar... Joking wif u oni... | |
| **2** | | 1 | | Free entry in 2 a wkly comp to win FA Cup fina... | |
| **3** | | 0 | | U dun say so early hor... U c already then say... | |
| **4** | | 0 | | Nah I don't think he goes to usf, he lives aro... | |

(5572, 3000)

Accuracy: 0.9757847533632287

Classification Report:

precision recall f1-score support

0 0.97 1.00 0.99 965

1 1.00 0.82 0.90 150

accuracy 0.98 1115

macro avg 0.99 0.91 0.94 1115

weighted avg 0.98 0.98 0.97 1115

Confusion Matrix:

[[965 0]

[ 27 123]]

Email: Congratulations! You've won a free ticket to the Bahamas. Claim now.

Prediction: spam

Email: Hey, can we reschedule our meeting to next week?

Prediction: ham

Email: You've been selected for a $1000 Walmart gift card. Click here to claim.

Prediction: spam

Fitting 5 folds for each of 54 candidates, totalling 270 fits

Best Parameters: {'nb\_\_alpha': 0.1, 'tfidf\_\_max\_df': 0.75, 'tfidf\_\_min\_df': 5, 'tfidf\_\_ngram\_range': (1, 2)}

Best Score: 0.9858646253265088

Accuracy: 0.979372197309417

Classification Report:

precision recall f1-score support

0 0.98 1.00 0.99 965

1 0.98 0.86 0.92 150

accuracy 0.98 1115

macro avg 0.98 0.93 0.95 1115

weighted avg 0.98 0.98 0.98 1115

Confusion Matrix:

[[963 2]

[ 21 129]]

| **label** | **text** | **email\_length** | **special\_chars** | **uppercase\_words** |
| --- | --- | --- | --- | --- |
| **0** | 0 | Go until jurong point, crazy.. Available only ... | 111 | 9 | 0 |
| **1** | 0 | Ok lar... Joking wif u oni... | 29 | 6 | 0 |
| **2** | 1 | Free entry in 2 a wkly comp to win FA Cup fina... | 155 | 6 | 2 |
| **3** | 0 | U dun say so early hor... U c already then say... | 49 | 6 | 2 |
| **4** | 0 | Nah I don't think he goes to usf, he lives aro... | 61 | 2 | 1 |

**Conclusion**

In this project, we successfully developed an email spam detection model using various machine learning techniques. The Support Vector Machine (SVM) model with the parameters {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'} achieved the highest accuracy of 0.9829. The model was evaluated using various performance metrics, and it demonstrated excellent precision, recall, and F1-score.

The additional features such as email length, number of special characters, and number of uppercase words contributed to the model's performance. Further improvements can be made by exploring more features, advanced models, and ensemble methods.

This project provides a robust foundation for building effective spam detection systems, highlighting the importance of feature engineering and model tuning in achieving high accuracy.