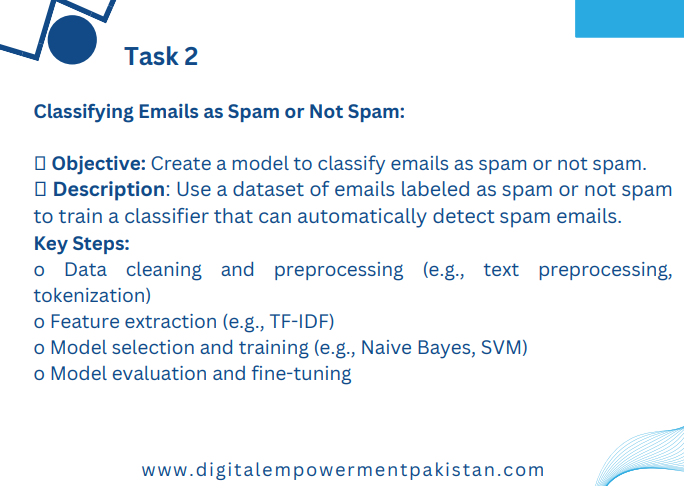
**ERRNUM NISHAT**

****

****

**Task 2:**

**Classifying Emails as Spam or Not Spam:**

** Objective: Create a model to classify emails as spam or not spam.**

** Description: Use a dataset of emails labeled as spam or not spam**

**to train a classifier that can automatically detect spam emails.**

**Key Steps:**

**o Data cleaning and preprocessing (e.g., text preprocessing,**

**tokenization)**

**o Feature extraction (e.g., TF-IDF)**

**o Model selection and training (e.g., Naive Bayes, SVM)**

**o Model evaluation and fine-tuning**

**Code:**

import pandas as pd

import re

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.svm import SVC

from sklearn.naive\_bayes import MultinomialNB

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

import matplotlib.pyplot as plt

import seaborn as sns

# Step 1: Load dataset

df = pd.read\_csv('spam\_or\_not\_spam.csv')

# Step 2: Data cleaning and preprocessing

def clean\_text(text):

    # Check if text is NaN

    if pd.isnull(text):

        return ''

    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove non-alphanumeric characters

    text = text.lower()  # Convert text to lowercase

    return text

# Apply cleaning function to 'email' column

df['cleaned\_email'] = df['email'].apply(clean\_text)

# Step 3: Feature extraction (TF-IDF Vectorization)

vectorizer = TfidfVectorizer(max\_features=5000)

X\_tfidf = vectorizer.fit\_transform(df['cleaned\_email'])

y = df['label']

# Step 4: Train-test split

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

# Step 5: Model selection and training (Naive Bayes)

clf\_nb = MultinomialNB()

clf\_nb.fit(X\_train, y\_train)

# Step 6: Model evaluation (Naive Bayes)

y\_pred\_nb = clf\_nb.predict(X\_test)

print("Naive Bayes Classifier:")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_nb))

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_nb))

# Step 7: Visualization (Confusion Matrix - Naive Bayes)

def plot\_confusion\_matrix(y\_test, y\_pred, title):

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

    plt.figure(figsize=(6, 4))

    sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])

    plt.xlabel('Predicted labels')

    plt.ylabel('True labels')

    plt.title(title)

    plt.show()

plot\_confusion\_matrix(y\_test, y\_pred\_nb, title='Confusion Matrix - Naive Bayes Classifier')

# Step 8: Model selection and training (SVM - optional)

clf\_svm = SVC(kernel='linear')

clf\_svm.fit(X\_train, y\_train)

# Step 9: Model evaluation (SVM - optional)

y\_pred\_svm = clf\_svm.predict(X\_test)

print("\nSVM Classifier:")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_svm))

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_svm))

# Step 10: Visualization (Confusion Matrix - SVM - optional)

plot\_confusion\_matrix(y\_test, y\_pred\_svm, title='Confusion Matrix - SVM Classifier')

# Step 11: Model fine-tuning (optional)

param\_grid = {'C': [0.1, 1, 10]}

grid\_search = GridSearchCV(clf\_svm, param\_grid, cv=5, scoring='accuracy', verbose=1)

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

print("Best parameters found:")

print(grid\_search.best\_params\_)

print("Best cross-validation score:")

print(grid\_search.best\_score\_)

# Step 12: Evaluate best model (SVM - optional)

best\_clf\_svm = grid\_search.best\_estimator\_

y\_pred\_best\_svm = best\_clf\_svm.predict(X\_test)

print("\nBest SVM Classifier:")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_best\_svm))

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_best\_svm))

# Step 13: Visualization (Confusion Matrix - Best SVM - optional)

plot\_confusion\_matrix(y\_test, y\_pred\_best\_svm, title='Confusion Matrix - Best SVM Classifier')

**OUTPUT:**

Naive Bayes Classifier:

Accuracy: 0.97

Classification Report:

precision recall f1-score support

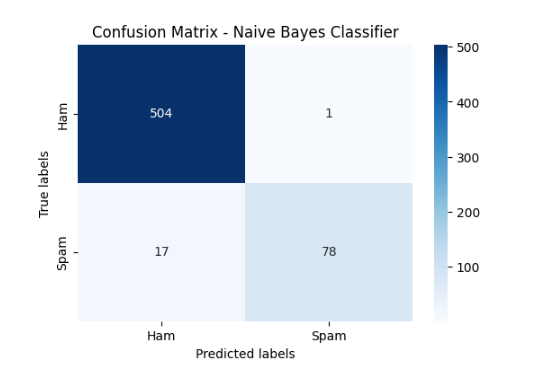
0 0.97 1.00 0.98 505

1 0.99 0.82 0.90 95

accuracy 0.97 600

macro avg 0.98 0.91 0.94 600

weighted avg 0.97 0.97 0.97 600

****

SVM Classifier:

Accuracy: 0.99

Classification Report:

precision recall f1-score support

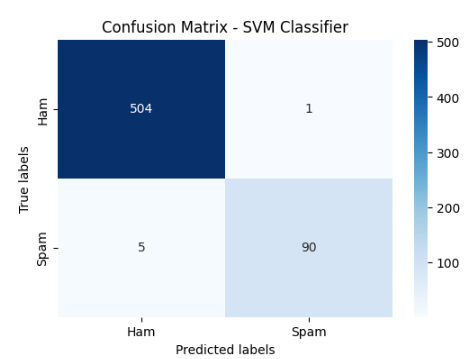
0 0.99 1.00 0.99 505

1 0.99 0.95 0.97 95

accuracy 0.99 600

macro avg 0.99 0.97 0.98 600

weighted avg 0.99 0.99 0.99 600

****

Fitting 5 folds for each of 3 candidates, totalling 15 fits

Best parameters found:

{'C': 10}

Best cross-validation score:

0.9891666666666667

Best SVM Classifier:

Accuracy: 0.99

Classification Report:

precision recall f1-score support

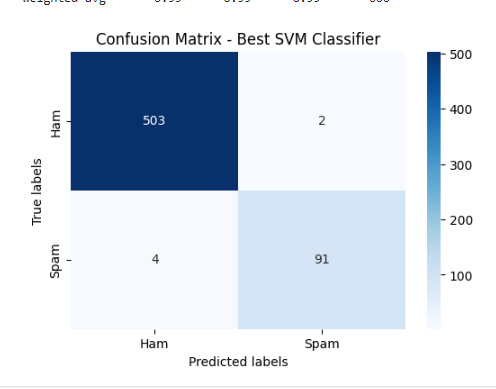
0 0.99 1.00 0.99 505

1 0.98 0.96 0.97 95

accuracy 0.99 600

macro avg 0.99 0.98 0.98 600

weighted avg 0.99 0.99 0.99 600

****

**Explaination:**

**Steps Integrated:**

* + **Data loading**: Load the dataset using pd.read\_csv.
  + **Data cleaning and preprocessing**: Define clean\_text function to clean email text.
  + **Feature extraction**: Use TfidfVectorizer to convert cleaned text into TF-IDF features.
  + **Train-test split**: Split data into training and testing sets.
  + **Model selection and training**: Train Naive Bayes classifier (clf\_nb) and optionally SVM classifier (clf\_svm).
  + **Model evaluation**: Evaluate classifiers using accuracy score and classification report.
  + **Visualization**: Plot confusion matrices using plot\_confusion\_matrix function.
  + **Model fine-tuning (optional)**: Perform grid search for SVM classifier hyperparameter tuning.
  + **Evaluate best model (optional)**: Evaluate SVM classifier with best parameters found from grid search.