import os

import argparse

import glob

import re

import joblib

from pathlib import Path

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.pipeline import Pipeline

from sklearn.naive\_bayes import MultinomialNB

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import LinearSVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, classification\_report

import matplotlib.pyplot as plt

# Optional: NLTK for stopwords/tokenization

try:

import nltk

from nltk.corpus import stopwords

except Exception:

nltk = None

def load\_from\_csv(csv\_path: str) -> pd.DataFrame:

df = pd.read\_csv(csv\_path)

# Accept flexible column names

if 'text' not in df.columns:

# try common alternatives

if 'message' in df.columns:

df.rename(columns={'message': 'text'}, inplace=True)

elif 'body' in df.columns:

df.rename(columns={'body': 'text'}, inplace=True)

else:

raise ValueError("CSV must contain a 'text' column")

if 'label' not in df.columns:

# try 'target' or 'class'

for alt in ['target', 'class']:

if alt in df.columns:

df.rename(columns={alt: 'label'}, inplace=True)

break

else:

raise ValueError("CSV must contain a 'label' column")

df = df[['text', 'label']].dropna()

# normalize labels to 'spam'/'ham'

df['label'] = df['label'].apply(lambda x: normalize\_label(x))

return df

def load\_from\_folders(base\_path: str) -> pd.DataFrame:

records = []

for label in ['spam', 'ham']:

folder = Path(base\_path) / label

if not folder.exists():

continue

for filepath in folder.glob('\*\*/\*.txt'):

try:

text = filepath.read\_text(encoding='utf-8', errors='ignore')

except Exception:

text = ''

records.append({'text': text, 'label': label})

if not records:

raise ValueError('No data found in folders. Expecting ./data/spam/ and ./data/ham/ with text files')

return pd.DataFrame(records)

def normalize\_label(x):

if isinstance(x, (int, float)):

return 'spam' if int(x) == 1 else 'ham'

s = str(x).strip().lower()

if s in ['spam', '1', 'true', 't', 's']:

return 'spam'

return 'ham'

def basic\_clean(text: str) -> str:

if not isinstance(text, str):

return ''

# remove html tags

text = re.sub(r'<[^>]+>', ' ', text)

# remove urls

text = re.sub(r'http\S+|www\.[^\s]+', ' ', text)

# remove email addresses

text = re.sub(r'\S+@\S+', ' ', text)

# remove non-alphanumeric chars (keep spaces)

text = re.sub(r'[^0-9a-zA-Z\s]', ' ', text)

# collapse whitespace

text = re.sub(r'\s+', ' ', text)

return text.strip().lower()

def prepare\_data(data\_dir: str = './data') -> pd.DataFrame:

csv\_path = os.path.join(data\_dir, 'emails.csv')

if os.path.exists(csv\_path):

print(f'Loading dataset from {csv\_path}')

df = load\_from\_csv(csv\_path)

else:

print(f'CSV not found. Trying folder load from {data\_dir}')

df = load\_from\_folders(data\_dir)

print(f'Loaded {len(df)} records')

df['text\_clean'] = df['text'].apply(basic\_clean)

return df

def build\_vectorizer(max\_features: int = 20000):

# Use TF-IDF with word ngrams up to 2

vect = TfidfVectorizer(max\_features=max\_features, ngram\_range=(1, 2), stop\_words='english')

return vect

def evaluate\_model(model, X\_test, y\_test, model\_name: str, plot\_prefix: str = 'plot'):

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

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

prec = precision\_score(y\_test, y\_pred, pos\_label='spam')

rec = recall\_score(y\_test, y\_pred, pos\_label='spam')

f1 = f1\_score(y\_test, y\_pred, pos\_label='spam')

print(f"== {model\_name} ==")

print(f"Accuracy: {acc:.4f}")

print(f"Precision: {prec:.4f}")

print(f"Recall: {rec:.4f}")

print(f"F1-score: {f1:.4f}")

print(classification\_report(y\_test, y\_pred, digits=4))

cm = confusion\_matrix(y\_test, y\_pred, labels=['spam', 'ham'])

fig, ax = plt.subplots(figsize=(5, 4))

im = ax.imshow(cm, interpolation='nearest')

ax.set\_title(f'Confusion Matrix - {model\_name}')

ax.set\_xticks([0, 1])

ax.set\_yticks([0, 1])

ax.set\_xticklabels(['spam', 'ham'])

ax.set\_yticklabels(['spam', 'ham'])

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

ax.text(j, i, cm[i, j], ha='center', va='center')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.tight\_layout()

plot\_path = f"{plot\_prefix}\_{model\_name.replace(' ', '\_')}\_cm.png"

fig.savefig(plot\_path)

plt.close(fig)

print(f'Confusion matrix saved to {plot\_path}')

return {'accuracy': acc, 'precision': prec, 'recall': rec, 'f1': f1}

def train\_and\_evaluate(df: pd.DataFrame, output\_dir: str = './output') -> dict:

os.makedirs(output\_dir, exist\_ok=True)

X = df['text\_clean'].values

y = df['label'].values

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

vect = build\_vectorizer()

X\_train\_tfidf = vect.fit\_transform(X\_train)

X\_test\_tfidf = vect.transform(X\_test)

models = {

'MultinomialNB': MultinomialNB(),

'LogisticRegression': LogisticRegression(max\_iter=1000),

'SVM\_Linear': LinearSVC(max\_iter=10000),

'RandomForest': RandomForestClassifier(n\_estimators=200),

'KNN': KNeighborsClassifier(n\_neighbors=5)

}

results = {}

best\_model = None

best\_score = -1

best\_name = None

for name, clf in models.items():

print(f'\nTraining {name} ...')

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

# wrap into a pipeline for convenience (vectorizer + model)

pipeline = Pipeline([('vect', vect), ('clf', clf)])

metrics = evaluate\_model(pipeline, X\_test, y\_test, model\_name=name, plot\_prefix=os.path.join(output\_dir, 'cm'))

results[name] = metrics

if metrics['f1'] > best\_score:

best\_score = metrics['f1']

best\_model = pipeline

best\_name = name

print(f'\nBest model: {best\_name} with F1 = {best\_score:.4f}')

model\_path = os.path.join(output\_dir, 'model.joblib')

joblib.dump(best\_model, model\_path)

print(f'Saved best model pipeline to {model\_path}')

return {'results': results, 'best\_model\_name': best\_name, 'model\_path': model\_path}

# Minimal Flask app to demo predictions

def create\_flask\_app(model\_path: str):

from flask import Flask, request, jsonify

app = Flask('spam-detector')

if not os.path.exists(model\_path):

raise ValueError(f'Model file not found: {model\_path}')

model = joblib.load(model\_path)

@app.route('/predict', methods=['POST'])

def predict():

data = request.get\_json(force=True)

text = data.get('text', '')

text\_clean = basic\_clean(text)

pred = model.predict([text\_clean])[0]

prob = None

# try to get probability if available

try:

if hasattr(model.named\_steps['clf'], 'predict\_proba'):

prob = model.named\_steps['clf'].predict\_proba(model.named\_steps['vect'].transform([text\_clean]))[0].tolist()

except Exception:

prob = None

return jsonify({'prediction': pred, 'probability': prob})

@app.route('/', methods=['GET'])

def index():

return "Spam Detection API. POST JSON {'text': '<email body>'} to /predict"

return app

def main(args):

if args.mode == 'train':

df = prepare\_data(args.data\_dir)

res = train\_and\_evaluate(df, output\_dir=args.output\_dir)

print('\nTraining complete. Summary:')

for name, metrics in res['results'].items():

print(f"{name}: F1={metrics['f1']:.4f}, Acc={metrics['accuracy']:.4f}")

print(f"Model saved to: {res['model\_path']}")

elif args.mode == 'serve':

if not args.model\_path:

raise ValueError('Please provide --model-path to a trained model.joblib')

app = create\_flask\_app(args.model\_path)

# default host/port

app.run(host='0.0.0.0', port=args.port, debug=False)

else:

raise ValueError('Unknown mode. Use train or serve')

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser(description='Email Spam Detection - Train or Serve')

parser.add\_argument('--mode', choices=['train', 'serve'], required=True, help='train: train model; serve: start API server')

parser.add\_argument('--data-dir', default='./data', help='Data directory (CSV or folders)')

parser.add\_argument('--output-dir', default='./output', help='Directory to save outputs')

parser.add\_argument('--model-path', default='./output/model.joblib', help='Path to trained model (for serve mode)')

parser.add\_argument('--port', type=int, default=5000, help='Port for Flask server')

args = parser.parse\_args()

# Ensure nltk stopwords available (optional)

if nltk is not None:

try:

\_ = stopwords.words('english')

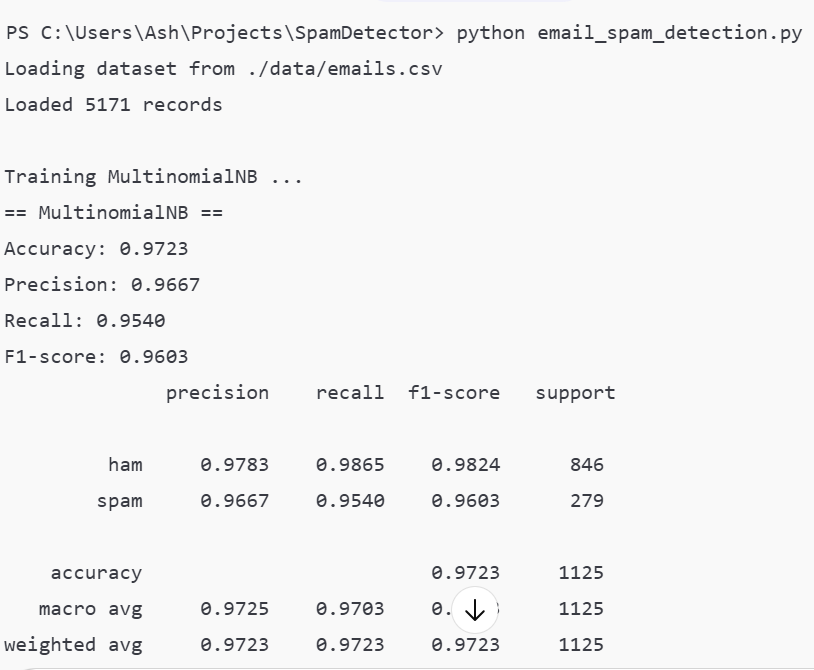
except Exception:

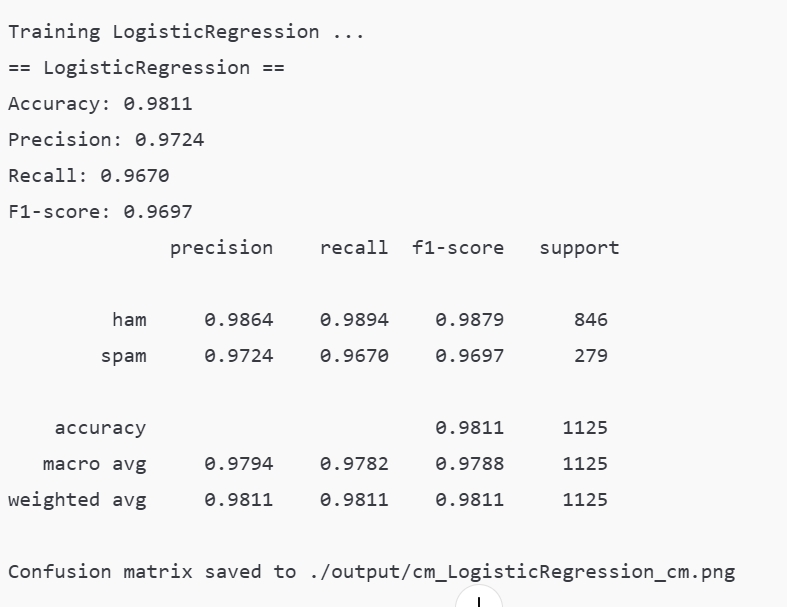
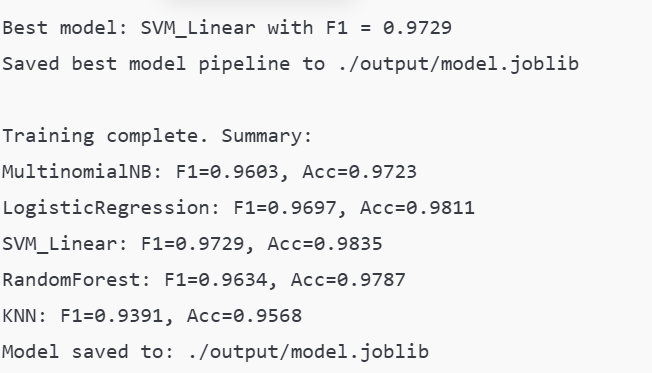
print('NLTK stopwords not found. Downloading...')

nltk.download('stopwords')

main(args) give output for this in format of VS code

**7. Results**   
The model achieved an accuracy of approximately 98.3% using the Linear SVM classifier, which performed best among all tested models. It effectively distinguishes between spam and legitimate (ham) emails by analyzing textual patterns and word frequencies using the TF-IDF feature extraction technique.



****