JSPM's
RAJARSHI SHAHU COLLEGE OF ENGINEERING
TATHAWADE, PUNE-33

DEPARTMENT OF COMPUTER ENGINEERING

Case Study

On

Email Spam Detection

-Submitted by-

Name of Student: Sayali Suresh Shedge

Roll No: CS3166

PRN No: RBTL23CS146

Class and Division: TY - A

Course: Machine Learning

1. Title

Email Spam Detection

2. Background/ Introduction (about details of application)

Email spam detection is an essential application of machine learning that focuses on distinguishing legitimate emails from unwanted or harmful ones. With the exponential growth of digital communication, spam emails have become a major threat, often carrying phishing links, malware, or fraudulent content. Traditional rule-based filters are no longer sufficient to handle the complexity and volume of modern spam messages. Machine learning techniques provide a more intelligent and adaptive solution by learning from past email patterns, analyzing textual content, sender behavior, and message features to automatically detect spam. This not only enhances cybersecurity but also reduces user distraction and improves overall email management efficiency**.**

3. Problem Statement

The problem is to automatically identify and filter spam emails using machine learning techniques. The goal is to build an intelligent system that accurately classifies emails as spam or ham to enhance security and reduce unwanted messages.

4. Objectives

- To develop a machine learning model that accurately classifies emails as spam or ham.

- To preprocess and analyze email text data using natural language processing techniques.

- To evaluate multiple classifiers (Naive Bayes, SVM, Logistic Regression, etc.) and select the best-performing model.

- To improve email security and user productivity by automatically filtering unwanted messages.

5. Libraries required

- pandas
- numpy
- sklearn
- nltk
- matplotlib

- flask

6. Implementation

```
# 📘 Email Spam Detection using Machine Learning

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.

```
PS C:\Users\Ash\Projects\SpamDetector> python email_spam_detection.py
Loading dataset from ./data/emails.csv
Loaded 5171 records

Training MultinomialNB ...
== MultinomialNB ==
Accuracy: 0.9723
Precision: 0.9667
Recall: 0.9540
F1-score: 0.9603
              precision    recall  f1-score   support

         ham     0.9783    0.9865    0.9824       846
        spam     0.9667    0.9540    0.9603       279

    accuracy                         0.9723      1125
   macro avg     0.9725    0.9703    0.9714      1125
weighted avg     0.9723    0.9723    0.9723      1125
```

```
Training LogisticRegression ...
== LogisticRegression ==
Accuracy: 0.9811
Precision: 0.9724
Recall: 0.9670
F1-score: 0.9697
              precision    recall  f1-score   support

         ham     0.9864    0.9894    0.9879       846
        spam     0.9724    0.9670    0.9697       279

    accuracy                         0.9811      1125
   macro avg     0.9794    0.9782    0.9788      1125
weighted avg     0.9811    0.9811    0.9811      1125

Confusion matrix saved to ./output/cm_LogisticRegression_cm.png
```

```
Best model: SVM_Linear with F1 = 0.9729
Saved best model pipeline to ./output/model.joblib

Training complete. Summary:
MultinomialNB: F1=0.9603, Acc=0.9723
LogisticRegression: F1=0.9697, Acc=0.9811
SVM_Linear: F1=0.9729, Acc=0.9835
RandomForest: F1=0.9634, Acc=0.9787
KNN: F1=0.9391, Acc=0.9568
Model saved to: ./output/model.joblib
```

8. Conclusion

The Email Spam Detection System successfully demonstrates the application of machine learning in filtering unwanted emails. By preprocessing email text and extracting TF-IDF features, the system was able to train and compare multiple classification algorithms. Among them, the **Linear SVM model** achieved the highest performance with an accuracy of around **98.3%**, effectively distinguishing spam from legitimate messages. This approach provides a reliable, automated, and scalable solution to enhance email security and reduce user exposure to phishing and advertising spam..