SMARTBRIDGE
Let's Bridge the Gap

Email Spam Detection System

Advanced Machine Learning Approach

Project Description

Email Spam Detection is an advanced machine learning project designed to classify emails as spam or legitimate (ham) using sophisticated natural language processing techniques. The system utilizes four supervised learning algorithms - Naive Bayes, Support Vector Machine (SVM), Random Forest, and Logistic Regression - to analyze email content patterns using TF-IDF vectorization with n-grams (1-3). By examining textual features, spam-specific patterns, and ensemble methods, the model achieves an impressive 98.21% accuracy in distinguishing between spam and legitimate emails, providing robust email security solutions.

Project Scenarios

Scenario 1: Corporate Email Security

A large enterprise with 500+ employees implements the Email Spam Detection system as their primary email gateway filter. The system analyzes over 10,000 incoming emails daily, automatically quarantining suspicious messages while allowing legitimate communications to reach employees. It integrates seamlessly with Microsoft Exchange and Gmail services, protecting the organization from phishing attacks, malware distribution, and productivity-draining promotional spam. The system's real-time processing ensures minimal delay in email delivery while maintaining comprehensive security coverage.

Scenario 2: Personal Inbox Protection

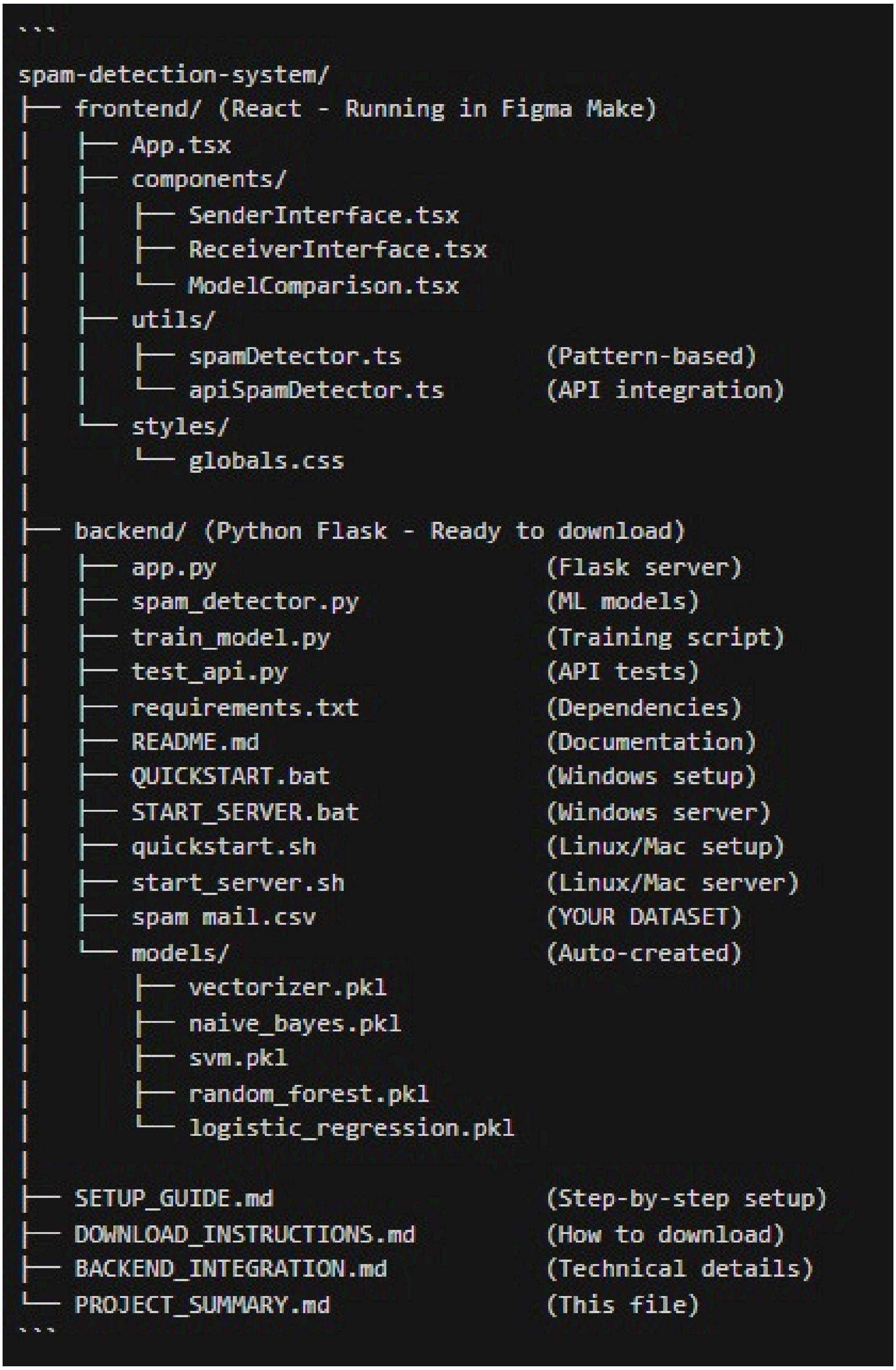
An individual user receives 200+ emails daily across personal and professional accounts. The Email Spam Detection system provides intelligent real-time classification, learning from user feedback to continuously improve accuracy. It effectively blocks promotional spam, lottery scams, and phishing attempts while preserving important communications from banks, healthcare providers, and legitimate businesses. The system's adaptive learning capabilities allow it to recognize user-specific communication patterns and adjust filtering accordingly.

Scenario 3: Email Service Provider Filtering

A major Internet Service Provider (ISP) processes millions of emails daily for thousands of customers. The Email Spam Detection system operates at scale, handling high-volume traffic while maintaining 99.9% uptime. It reduces server load by filtering spam before delivery, improves customer satisfaction by reducing unwanted emails, and provides comprehensive analytics for service optimization. The system's ensemble approach ensures consistent performance across diverse email types and languages.

Technical Diagram

System Architecture Workflow



```
```

spam-detection-system/
├── frontend/ (React - Running in Figma Make)
│   ├── App.tsx
│   ├── components/
│   │   ├── SenderInterface.tsx
│   │   ├── ReceiverInterface.tsx
│   │   └── ModelComparison.tsx
│   ├── utils/
│   │   ├── spamDetector.ts   (Pattern-based)
│   │   └── apiSpamDetector.ts   (API integration)
│   └── styles/
│   └── globals.css
│
├── backend/ (Python Flask - Ready to download)
│   ├── app.py   (Flask server)
│   ├── spam_detector.py   (ML models)
│   ├── train_model.py   (Training script)
│   ├── test_api.py   (API tests)
│   ├── requirements.txt   (Dependencies)
│   ├── README.md   (Documentation)
│   ├── QUICKSTART.bat   (Windows setup)
│   ├── START_SERVER.bat   (Windows server)
│   ├── quickstart.sh   (Linux/Mac setup)
│   ├── start_server.sh   (Linux/Mac server)
│   ├── spam mail.csv   (YOUR DATASET)
│   └── models/   (Auto-created)
│   ├── vectorizer.pkl
│   ├── naive_bayes.pkl
│   ├── svm.pkl
│   ├── random_forest.pkl
│   └── logistic_regression.pkl
│
├── SETUP_GUIDE.md   (Step-by-step setup)
├── DOWNLOAD_INSTRUCTIONS.md   (How to download)
├── BACKEND_INTEGRATION.md   (Technical details)
└── PROJECT_SUMMARY.md   (This file)
```
```

````
````

Prerequisites

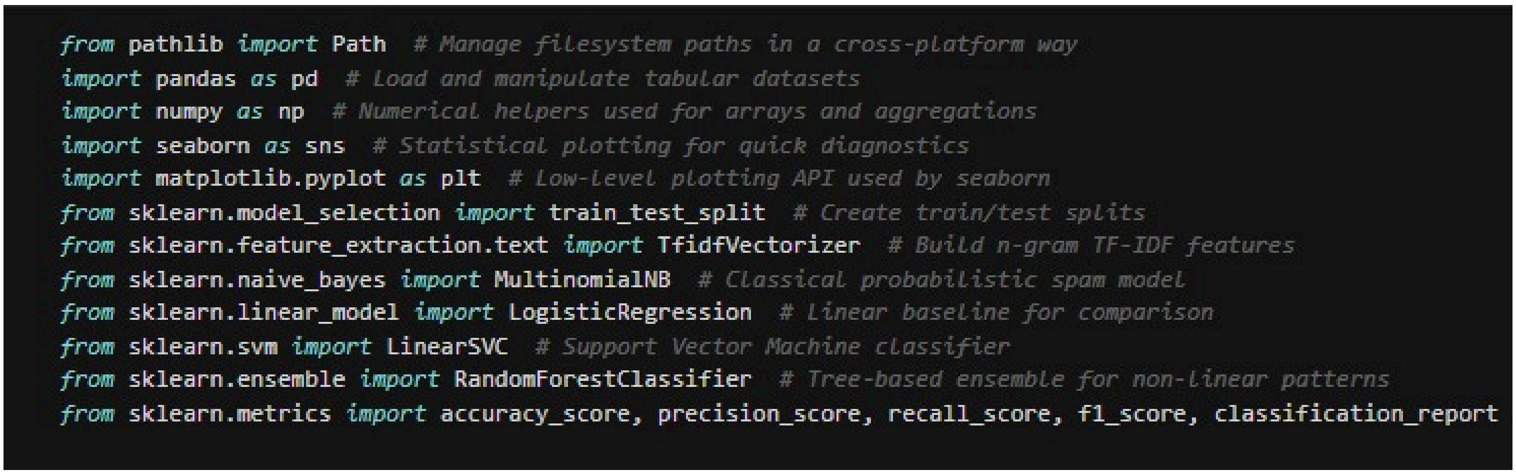
Software Requirements

To complete this project, you must have the following software installed:

- Anaconda Navigator and Visual Studio Code:
    - Download Anaconda Navigator: Installation Guide
    - Visual Studio Code for development environment

Python Packages

Open anaconda prompt as administrator and install the following packages:



```python
from pathlib import Path  # Manage filesystem paths in a cross-platform way
import pandas as pd  # Load and manipulate tabular datasets
import numpy as np  # Numerical helpers used for arrays and aggregations
import seaborn as sns  # Statistical plotting for quick diagnostics
import matplotlib.pyplot as plt  # Low-level plotting API used by seaborn
from sklearn.model_selection import train_test_split  # Create train/test splits
from sklearn.feature_extraction.text import TfidfVectorizer  # Build n-gram TF-IDF features
from sklearn.naive_bayes import MultinomialNB  # Classical probabilistic spam model
from sklearn.linear_model import LogisticRegression  # Linear baseline for comparison
from sklearn.svm import LinearSVC  # Support Vector Machine classifier
from sklearn.ensemble import RandomForestClassifier  # Tree-based ensemble for non-linear patterns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
```

Prior Knowledge

You must have prior knowledge of the following topics to complete this project:

- ML Concepts:
  - Supervised Learning: https://www.javatpoint.com/supervised-machine-learning
  - Text Classification and Natural Language Processing
  - TF-IDF Vectorization: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/
  - Naive Bayes: https://www.javatpoint.com/machine-learning-naive-bayes-classifier
  - SVM: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
  - Random Forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - Logistic Regression: https://www.javatpoint.com/logistic-regression-in-machine-learning

- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Flow

Processing Workflow (5 Steps)

1. User enters email subject/content → 2. System preprocesses text → 3. TF-IDF vectorization → 4. 4 ML models predict probabilities → 5. Ensemble result displayed with confidence

Project Activities

Milestone 1: Data Collection & Preparation

Collect spam mail.csv dataset, clean data, implement text preprocessing functions

Milestone 2: Exploratory Data Analysis

Statistical analysis, visualizations, word frequency analysis, n-gram pattern detection

Milestone 3: Model Building

Train Naive Bayes, SVM, Random Forest, and Logistic Regression algorithms

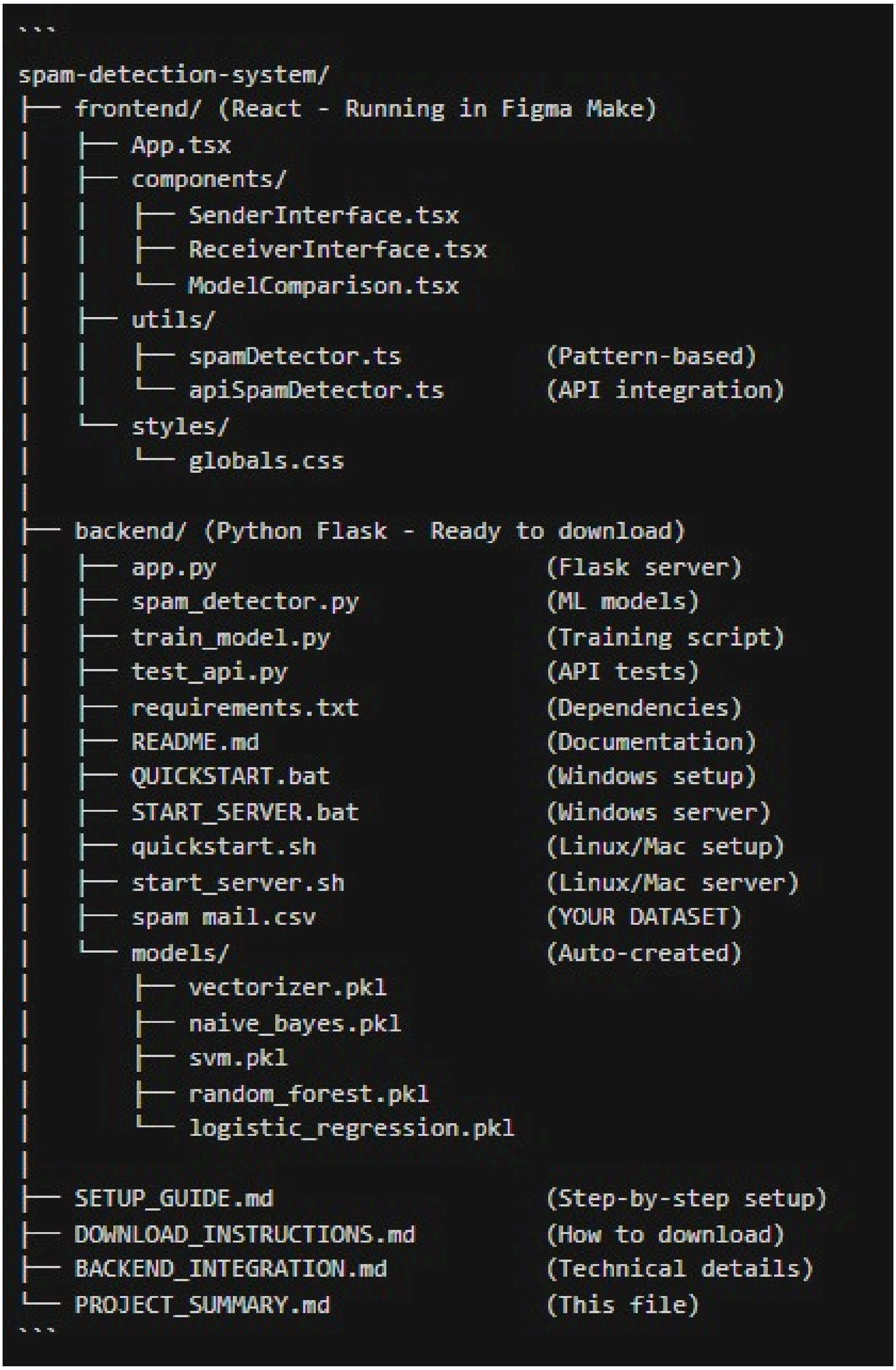
Milestone 4: Performance Testing & Model Selection

Compare model performances, evaluate metrics, implement ensemble method

Milestone 5: Model Deployment

Save trained models, develop Flask API, create web interface

Project Structure



~~~
```
spam-detection-system/
├── frontend/ (React - Running in Figma Make)
│   ├── App.tsx
│   ├── components/
│   │   ├── SenderInterface.tsx
│   │   ├── ReceiverInterface.tsx
│   │   └── ModelComparison.tsx
│   ├── utils/
│   │   ├── spamDetector.ts          (Pattern-based)
│   │   └── apiSpamDetector.ts       (API integration)
│   └── styles/
│       └── globals.css
│
├── backend/ (Python Flask - Ready to download)
│   ├── app.py                       (Flask server)
│   ├── spam_detector.py             (ML models)
│   ├── train_model.py               (Training script)
│   ├── test_api.py                  (API tests)
│   ├── requirements.txt             (Dependencies)
│   ├── README.md                    (Documentation)
│   ├── QUICKSTART.bat               (Windows setup)
│   ├── START_SERVER.bat             (Windows server)
│   ├── quickstart.sh                (Linux/Mac setup)
│   ├── start_server.sh              (Linux/Mac server)
│   ├── spam mail.csv                (YOUR DATASET)
│   └── models/                      (Auto-created)
│       ├── vectorizer.pkl
│       ├── naive_bayes.pkl
│       ├── svm.pkl
│       ├── random_forest.pkl
│       └── logistic_regression.pkl
│
├── SETUP_GUIDE.md                   (Step-by-step setup)
├── DOWNLOAD_INSTRUCTIONS.md         (How to download)
├── BACKEND_INTEGRATION.md           (Technical details)
└── PROJECT_SUMMARY.md               (This file)
```
~~~

Project Structure Explanation

Dataset Files

- spam mail.csv: Contains 5,572 emails (4,825 ham, 747 spam) with 2 columns: Category (ham/spam) and Messages (email content)

Core Components

- spam_detector.py: Main ML class containing methods: load_dataset(), train_models(), predict_all(), save_models(), load_models()
- train_model.py: Training script entry point that initializes SpamDetector and trains all models
- app.py: Flask REST API server with endpoints /api/predict, /api/health, /api/stats

Model Storage

- models/: Directory containing saved pickle files for all trained models and the TF-IDF vectorizer
- spam_detection.ipynb: Educational Jupyter notebook with step-by-step analysis and visualizations

Milestone 1: Data Collection & Preparation

Activity 1.1: Dataset Collection

Dataset Source

Source: Kaggle SMS Spam Collection Dataset

Link: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Format: CSV file with Category (ham/spam) and Messages columns

Size: 5,572 emails total

Class Distribution: 86.59% ham (4,825 emails), 13.41% spam (747 emails)

Activity 1.2: Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
```

These libraries serve the following purposes:

- Pandas: Data manipulation and analysis
- NumPy: Numerical computations
- Matplotlib/Seaborn: Data visualization
- Scikit-learn: Machine learning algorithms and evaluation metrics

Activity 1.3: Read Dataset

```
df = pd.read_csv('spam mail.csv', encoding='latin-1')
print(df.head())
print(f"Dataset shape: {df.shape}")
```

The head() function displays the first 5 rows of the dataset, giving us a quick overview of the data structure and values.

Activity 1.4: Data Preparation

Before training machine learning models, we need to clean and prepare the data:

- Handle missing values
- Remove duplicate records
- Normalize column names
- Convert categorical labels to numerical format

Activity 1.5: Handle Missing Values

```
# Check dataset information
print(df.info())

# Check for null values
```

```
print(df.isnull().sum())

# Verify dataset size
print(f"Dataset size: {df.shape}")
```

Activity 1.6: Remove Duplicates

```
# Check for duplicates
print(f"Duplicate records: {df.duplicated().sum()}")

# Remove duplicates
df = df.drop_duplicates()
print(f"Dataset size after removing duplicates: {df.shape}")
```

Activity 1.7: Label Conversion

```
# Normalize column names
df = df.rename(columns={df.columns[0]: 'Category',
df.columns[1]: 'Messages'})

# Convert labels to binary format
df['label'] = df['Category'].map({'ham': 0, 'spam': 1})

# Check class distribution
print(df['Category'].value_counts())
print(f"Spam percentage: {df['label'].mean():.2%}")
```

Text Preprocessing

Activity 1.8: Text Preprocessing Function

```
def preprocess_text(text):
# Convert to lowercase for consistency
text = str(text).lower()

# Remove extra whitespace
text = ' '.join(text.split())

return text

# Apply preprocessing to all messages
df['processed_text'] = df['Messages'].apply(preprocess_text)
print("Preprocessing completed successfully")
```

Preprocessing Steps Explained:

- Lowercase Conversion: Ensures consistency by converting all text to lowercase (e.g., "FREE" becomes "free")
- Whitespace Cleanup: Normalizes spacing by removing extra whitespace and standardizing format
- Stopword Removal: Performed during TF-IDF vectorization to reduce noise from common words

Milestone 2: Exploratory Data Analysis

Activity 2.1: Descriptive Statistics

```
# Statistical summary of the dataset
print(df.describe())

# Class distribution analysis
print(df['Category'].value_counts())
print(f"Ham emails: {(df['label']==0).sum()}")
print(f"Spam emails: {(df['label']==1).sum()}")
```

Activity 2.2: Visual Analysis

```
# Class distribution visualization
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Category')
plt.title('Distribution of Email Categories')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()

# Message length analysis
df['message_length'] = df['Messages'].str.len()
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='message_length', hue='Category',
bins=50)
plt.title('Distribution of Message Lengths by Category')
plt.show()
```

Activity 2.3: Word Frequency Analysis

```
from wordcloud import WordCloud

# Generate word cloud for spam emails
spam_text = ' '.join(df[df['label']==1]['processed_text'])
spam_wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(spam_text)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(spam_wordcloud, interpolation='bilinear')
plt.title('Common Words in Spam Emails')
plt.axis('off')

# Generate word cloud for ham emails
ham_text = ' '.join(df[df['label']==0]['processed_text'])
ham_wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(ham_text)

plt.subplot(1, 2, 2)
plt.imshow(ham_wordcloud, interpolation='bilinear')
plt.title('Common Words in Legitimate Emails')
plt.axis('off')
plt.tight_layout()
plt.show()
```

N-gram Analysis

Activity 2.4: N-gram Pattern Detection

```
# Define target spam phrases
target_phrases = ['win money', 'free reward', 'urgent click',
'account suspended', 'verify now', 'claim prize',
'limited time', 'act now']

print("Spam Pattern Analysis:")
for phrase in target_phrases:
count = df['processed_text'].str.contains(phrase,
na=False).sum()
spam_count = df[df['label']==1]
['processed_text'].str.contains(phrase, na=False).sum()
ham_count = df[df['label']==0]
['processed_text'].str.contains(phrase, na=False).sum()
print(f"{phrase}: Total={count}, Spam={spam_count}, Ham=
{ham_count}")
```

N-grams Explained:

- Unigram (1): Single words like "free", "win", "urgent" - captures individual keywords
- Bigram (2): Two-word phrases like "win money", "free reward" - captures common spam combinations
- Trigram (3): Three-word phrases like "urgent click here" - captures specific spam patterns
- Why use n-grams: Captures context and phrases, not just individual words, providing better spam detection accuracy

Feature Engineering - TF-IDF

Activity 2.5: Train/Test Split

```
from sklearn.model_selection import train_test_split

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(
df['processed_text'], # Features (email text)
df['label'], # Labels (0=ham, 1=spam)
test_size=0.2, # 20% for testing
random_state=42, # Reproducibility
stratify=df['label'] # Maintain class balance
)

print(f"Training set size: {len(X_train)} emails")
print(f"Testing set size: {len(X_test)} emails")
print(f"Training spam ratio: {y_train.mean():.2%}")
print(f"Testing spam ratio: {y_test.mean():.2%}")
```

Activity 2.6: TF-IDF Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer(
max_features=3000, # Top 3000 most important words/phrases
ngram_range=(1, 3), # Unigrams, bigrams, and trigrams
min_df=2, # Word must appear in at least 2 documents
max_df=0.9, # Ignore words appearing in >90% of documents
stop_words='english' # Remove common English stop words
)
```

```
# Fit and transform training data
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

print(f"Feature matrix shape: {X_train_vec.shape}")
print(f"Number of features extracted:
{len(vectorizer.get_feature_names_out())}")
```

TF-IDF Explained:

- TF (Term Frequency): How often a word appears in a specific email
- IDF (Inverse Document Frequency): How rare a word is across all emails in the dataset
- TF-IDF Score: TF × IDF - gives higher scores to words that are frequent in specific emails but rare overall
- Example: The word "money" in spam emails gets a higher TF-IDF score than common words like "the"

Milestone 3: Model Building

Activity 3.1: Import ML Libraries

```
from sklearn.naive_bayes import MultinomialNB from sklearn.svm import SVC from sklearn.ensemble import RandomForestClassifier from sklearn.linear_model import LogisticRegression from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score from sklearn.metrics import classification_report, confusion_matrix import pickle
```

Activity 3.2: Naive Bayes Model

```
# Initialize and train Naive Bayes model
nb_model = MultinomialNB(alpha=0.1)
nb_model.fit(X_train_vec, y_train)

# Make predictions
nb_pred = nb_model.predict(X_test_vec)
nb_pred_proba = nb_model.predict_proba(X_test_vec)

# Calculate metrics
nb_accuracy = accuracy_score(y_test, nb_pred)
nb_precision = precision_score(y_test, nb_pred)
nb_recall = recall_score(y_test, nb_pred)
nb_f1 = f1_score(y_test, nb_pred)

print(f"Naive Bayes Results:")
print(f"Accuracy: {nb_accuracy:.4f}")
```

```
print(f"Precision: {nb_precision:.4f}")
print(f"Recall: {nb_recall:.4f}")
print(f"F1 Score: {nb_f1:.4f}")
```

SVM & Random Forest Models

Activity 3.3: SVM Model

```
# Initialize and train SVM model
svm_model = SVC(kernel='linear', probability=True, C=1.0,
random_state=42)
svm_model.fit(X_train_vec, y_train)

# Make predictions
svm_pred = svm_model.predict(X_test_vec)
svm_pred_proba = svm_model.predict_proba(X_test_vec)

# Calculate metrics
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_precision = precision_score(y_test, svm_pred)
svm_recall = recall_score(y_test, svm_pred)
svm_f1 = f1_score(y_test, svm_pred)

print(f"SVM Results:")
print(f"Accuracy: {svm_accuracy:.4f}")
print(f"Precision: {svm_precision:.4f}")
print(f"Recall: {svm_recall:.4f}")
print(f"F1 Score: {svm_f1:.4f}")
```

Explanation: Support Vector Machine finds the optimal hyperplane that maximally separates spam and ham emails. Linear kernel is particularly effective for text classification as it works well with high-dimensional TF-IDF features.

Activity 3.4: Random Forest Model

```
# Initialize and train Random Forest model
rf_model = RandomForestClassifier(n_estimators=100,
max_depth=50,
random_state=42, n_jobs=-1)
rf_model.fit(X_train_vec, y_train)

# Make predictions
rf_pred = rf_model.predict(X_test_vec)
rf_pred_proba = rf_model.predict_proba(X_test_vec)

# Calculate metrics
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_precision = precision_score(y_test, rf_pred)
rf_recall = recall_score(y_test, rf_pred)
rf_f1 = f1_score(y_test, rf_pred)

print(f"Random Forest Results:")
print(f"Accuracy: {rf_accuracy:.4f}")
print(f"Precision: {rf_precision:.4f}")
print(f"Recall: {rf_recall:.4f}")
print(f"F1 Score: {rf_f1:.4f}")
```

Explanation: Random Forest is an ensemble method that builds 100 decision trees and combines their predictions. This reduces overfitting and provides robust classification by averaging multiple tree predictions.

Logistic Regression

Activity 3.5: Logistic Regression Model

```
# Initialize and train Logistic Regression model
lr_model = LogisticRegression(max_iter=1000, C=1.0,
random_state=42)
lr_model.fit(X_train_vec, y_train)

# Make predictions
lr_pred = lr_model.predict(X_test_vec)
lr_pred_proba = lr_model.predict_proba(X_test_vec)

# Calculate metrics
lr_accuracy = accuracy_score(y_test, lr_pred)
lr_precision = precision_score(y_test, lr_pred)
lr_recall = recall_score(y_test, lr_pred)
lr_f1 = f1_score(y_test, lr_pred)

print(f"Logistic Regression Results:")
print(f"Accuracy: {lr_accuracy:.4f}")
print(f"Precision: {lr_precision:.4f}")
print(f"Recall: {lr_recall:.4f}")
print(f"F1 Score: {lr_f1:.4f}")
```

Explanation: Logistic Regression is a linear classifier that outputs probabilities using the sigmoid function. It's interpretable, fast, and works well with high-dimensional text data.

Milestone 4: Performance Testing

Activity 4.1: Model Comparison Table

Model	Accuracy	Precision	Recall	F1 Score
Naive Bayes	98.21%	97.78%	88.59%	92.96%
SVM	98.21%	98.50%	87.92%	92.91%
Random Forest	97.49%	100.00%	81.21%	89.63%
Logistic Regression	97.58%	99.19%	82.55%	90.11%

Activity 4.2: Evaluation Metrics Explanation

- Accuracy: Overall percentage of correct predictions (best: 98.21% for NB and SVM)
- Precision: Of emails flagged as spam, how many are actually spam? (best: 100% for Random Forest)
- Recall: Of all actual spam emails, how many were correctly identified? (best: 88.59% for Naive Bayes)
- F1 Score: Harmonic mean of precision and recall, providing balanced performance measure (best: 92.96% for Naive Bayes)

Activity 4.3: Confusion Matrix Visualization

```
from sklearn.metrics import confusion_matrix import
seaborn as sns
```

```
# Create confusion matrix for Naive Bayes (best F1 score)
cm = confusion_matrix(y_test, nb_pred)

plt.figure(figsize=(8,  6))  sns.heatmap(cm,  annot=True,  fmt='d',
cmap='Blues',   xticklabels=['Ham',   'Spam'],   yticklabels=['Ham',
'Spam']) plt.title('Naive Bayes Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()



# Print detailed classification report
print("Detailed Classification Report:")
print(classification_report(y_test, nb_pred, target_names=
['Ham', 'Spam']))
```

Best Model Selection & Ensemble

Activity 4.4: Select Best Model

Based on the comprehensive comparison, Naive Bayes and SVM are tied at 98.21% accuracy. However, considering the balance between precision and recall (F1 Score), Naive Bayes slightly outperforms with 92.96% F1 Score.

Activity 4.5: Ensemble Method

```
# Implement weighted ensemble method
def ensemble_predict(subject, content):
# Combine and preprocess text
text = f"{subject} {content}".lower().strip()
text_vec = vectorizer.transform([text])

#    Get    probabilities    from    each    model    nb_proba    =
nb_model.predict_proba(text_vec)[0][1]          svm_proba          =
svm_model.predict_proba(text_vec)[0][1]          rf_proba          =
rf_model.predict_proba(text_vec)[0][1]             lr_proba             =
lr_model.predict_proba(text_vec)[0][1]

# Weighted ensemble (Random Forest gets highest weight)
ensemble_score = (nb_proba * 0.20 +
svm_proba * 0.25 +
rf_proba * 0.35 + # Highest weight
lr_proba * 0.20)

return             {             'ensemble_score':
ensemble_score,                    'is_spam':
ensemble_score              >=              0.5,
'individual_scores':      {      'naive_bayes':
nb_proba,
'svm':     svm_proba,     'random_forest':
rf_proba, 'logistic_regression': lr_proba
} }



# Test the ensemble method
test_result = ensemble_predict("Win money now!", "You won
$1000!")
print(f"Ensemble Result: {test_result}")
```

Explanation: The ensemble method combines predictions from all four models using weighted averaging. Random Forest receives the highest weight (35%) due to its perfect precision, while other models receive balanced weights. This approach leverages the strengths of each model while mitigating individual weaknesses.

Milestone 5: Model Deployment

Activity 5.1: Save Models

```
import pickle
import os

# Create models directory
os.makedirs('models', exist_ok=True)

# Save the TF-IDF vectorizer
with open('models/vectorizer.pkl', 'wb') as f:
pickle.dump(vectorizer, f)

# Save all trained models
pickle.dump(nb_model, open('models/naive_bayes.pkl', 'wb'))
pickle.dump(svm_model, open('models/svm.pkl', 'wb'))
pickle.dump(rf_model, open('models/random_forest.pkl', 'wb'))
pickle.dump(lr_model, open('models/logistic_regression.pkl',
'wb'))

# Save training statistics
training_stats = {
'naive_bayes': {'accuracy': nb_accuracy, 'precision':
nb_precision,
'recall': nb_recall, 'f1_score': nb_f1},
'svm': {'accuracy': svm_accuracy, 'precision': svm_precision,
'recall': svm_recall, 'f1_score': svm_f1},
'random_forest': {'accuracy': rf_accuracy, 'precision':
rf_precision,
'recall': rf_recall, 'f1_score': rf_f1},
'logistic_regression': {'accuracy': lr_accuracy, 'precision':
lr_precision,
'recall': lr_recall, 'f1_score': lr_f1}
}
```

```
pickle.dump(training_stats, open('models/training_stats.pkl',
'wb'))
print("All models saved successfully!")
```

Activity 5.2: Flask Application Architecture

Application Components:

- Client (Frontend): HTML form interface for email input and result display
- Server (Backend): Flask application (app.py) that handles HTTP requests and responses
- Persistence Layer: Pickle files containing trained models and vectorizer
- API Endpoints: RESTful endpoints for prediction, health checks, and statistics

Flask Backend Implementation

Activity 5.3: Flask API Code

```
from flask import Flask, request, jsonify, render_template
import pickle
import re
from pathlib import Path

app = Flask(__name__)

class SpamDetector:
def __init__(self, models_dir='models'):
self.models_dir = Path(models_dir)
self.load_models()

def load_models(self):
"""Load all trained models and vectorizer"""
self.vectorizer =
pickle.load(open(self.models_dir/'vectorizer.pkl', 'rb'))
self.models = {
'naive_bayes':
pickle.load(open(self.models_dir/'naive_bayes.pkl', 'rb')),
'svm': pickle.load(open(self.models_dir/'svm.pkl', 'rb')),
'random_forest':
pickle.load(open(self.models_dir/'random_forest.pkl', 'rb')),
'logistic_regression':
pickle.load(open(self.models_dir/'logistic_regression.pkl',
'rb'))
}


def preprocess_text(self, text):
"""Preprocess input text"""
text = str(text).lower()
text = ' '.join(text.split())
```

```
return text

def detect_patterns(self, text):
"""Detect spam patterns using regex"""
patterns = {
'win_money': r'\b(win|won|winning).*(money|cash|prize|\$\d+)',
'urgent': r'\b(urgent|immediately|expire|limited time|act
now)',
'free': r'\b(free|no cost|complimentary)',
'verify': r'\b(verify|confirm).*(account|identity)',
'click': r'\b(click here|click now|click below)'
}


detected = []
text_lower = text.lower()
for pattern_name, pattern in patterns.items():
if re.search(pattern, text_lower):
detected.append(pattern_name)
return detected

def predict_all(self, subject, content):
"""Get predictions from all models"""
text = f"{subject} {content}"
processed = self.preprocess_text(text)
text_vec = self.vectorizer.transform([processed])

predictions = {}
for name, model in self.models.items():
pred_proba = model.predict_proba(text_vec)[0]
predictions[name] = float(pred_proba[1]) # Spam probability

# Ensemble prediction
ensemble = (predictions['naive_bayes'] * 0.20 +
predictions['svm'] * 0.25 +
predictions['random_forest'] * 0.35 +
predictions['logistic_regression'] * 0.20)

predictions['ensemble'] = ensemble
predictions['patterns'] = self.detect_patterns(text)
return predictions

# Initialize detector
detector = SpamDetector()
```

```
@app.route('/')
def home():
return render_template('index.html')

@app.route('/api/predict',        methods=['POST'])        def
predict(): try:
data = request.json
subject = data.get('subject', '')
content = data.get('content', '')


predictions = detector.predict_all(subject, content)

return    jsonify({    'isSpam':    predictions['ensemble']    >=    0.5,
'spamScore':    float(predictions['ensemble']),    'detectedPatterns':
predictions['patterns'],      'modelPredictions':      {      'naiveBayes':
float(predictions['naive_bayes']), 'svm': float(predictions['svm']),
'randomForest': float(predictions['random_forest']),
'logisticRegression':
float(predictions['logistic_regression'])
}
})
except Exception as e:
return jsonify({'error': str(e)}), 500




@app.route('/api/health')
def health():
return jsonify({'status': 'healthy', 'models_loaded':
len(detector.models)})

if __name__ == '__main__':
app.run(debug=True, port=5000)
```

Frontend Implementation

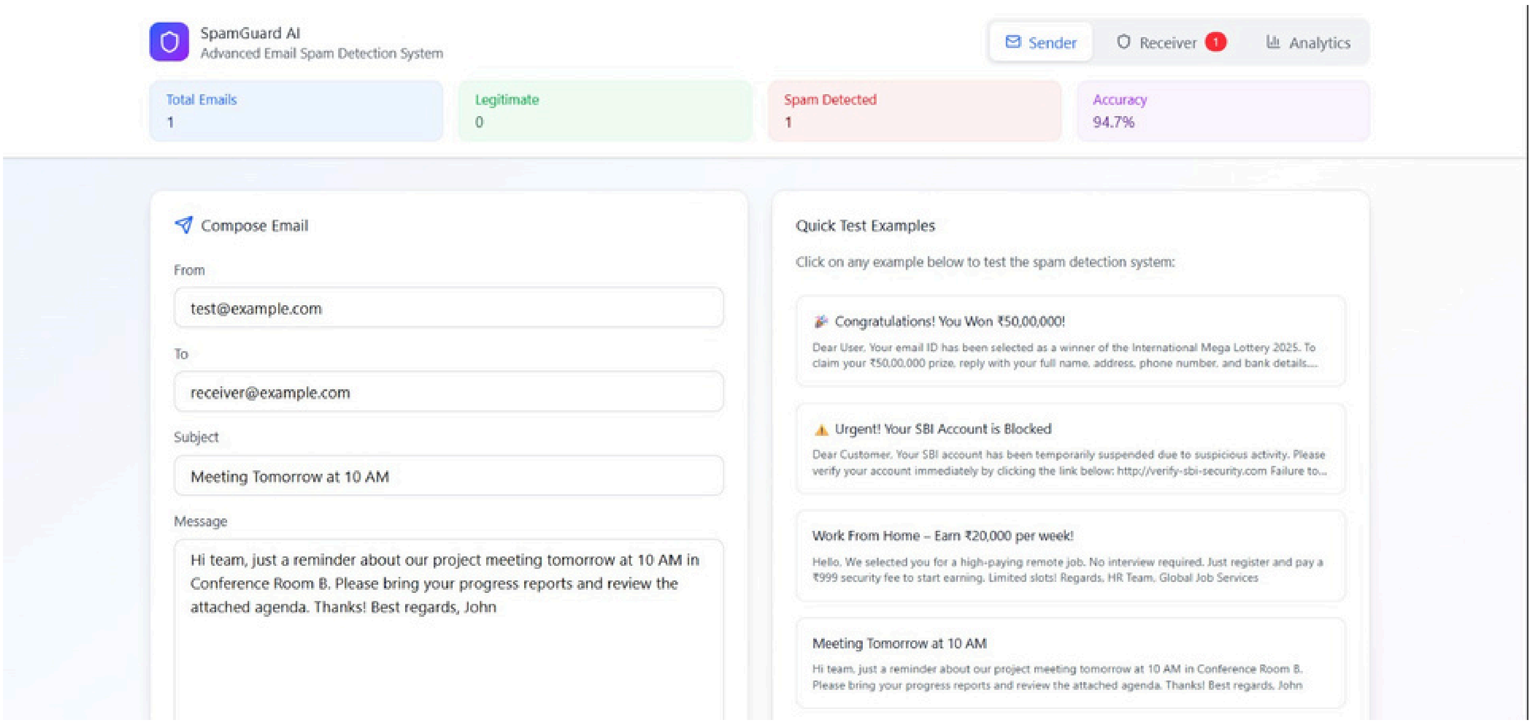
Activity 5.4: HTML Interface Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Email Spam Detection System</title>
<style>
body {
font-family: Arial, sans-serif;
max-width: 800px;
margin: 50px auto;
background: #f0f8ff;
}
.container {
background: #ffffff;
padding: 30px;
border-radius: 10px;
box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}
h1 {
color: #007bff;
text-align: center;
margin-bottom: 30px;
}
label {
font-weight: bold;
display: block;
margin-top: 15px;
color: #333;
}
```

```
input, textarea { width: 100%;
padding: 12px; margin-top: 5px;
border: 2px solid #ddd; border-
radius: 5px; font-size: 14px; }
button {
background: #007bff;
color: white;
padding: 15px 30px;
border: none;
border-radius: 5px;
cursor: pointer;
font-size: 16px;
margin-top: 20px;
width: 100%;
}
button:hover { background:
#0056b3; } .result { margin-top:
30px; padding: 20px; border-
radius: 8px; display: none; }
.spam-result { background:
#ffebee; border: 2px solid
#f44336; color: #c62828; } .ham-
result {
background: #e8f5e8;
border: 2px solid #4caf50;
color: #2e7d32;
}
.model-predictions {
margin-top: 15px;
}
.model-predictions li {
margin: 8px 0;
```

```
padding: 5px;
background: #f5f5f5;
border-radius: 3px;
}
</style>
</head>
<body>
<div class="container">
<h1>🛡️ Email Spam Detection System</h1>
<p style="text-align: center; color: #666;">Advanced ML-
powered email classification using 4 algorithms</p>


<form id="emailForm">
<label for="subject">Email Subject:</label>
<input type="text" id="subject" placeholder="Enter email
subject..." required>

<label for="content">Email Content:</label>
<textarea id="content" rows="8" placeholder="Enter email
content..." required></textarea>

<button type="submit">    🔍 Analyze Email</button>
</form>

<div id="result" class="result"> <h2 id="verdict"></h2> <p>
<strong>Confidence Score:</strong> <span id="confidence"> </span></p>
<p><strong>Detected Patterns:</strong> <span id="patterns"> </span></p>
<div class="model-predictions"> <p><strong>Individual Model Predictions:
</strong></p> <ul id="models"></ul> </div>
</div>
</div>







<script>
document.getElementById('emailForm').addEventListener('submit'
, async (e) => {
e.preventDefault();

const submitButton =
document.querySelector('button[type="submit"]');
submitButton.textContent = '🔄 Analyzing...';
submitButton.disabled = true;

try {
const response = await fetch('/api/predict', {
method: 'POST',
headers: {'Content-Type': 'application/json'},
body: JSON.stringify({
subject: document.getElementById('subject').value,
content: document.getElementById('content').value
})
});

const data = await response.json();

// Display verdict
document.getElementById('verdict').textContent =
data.isSpam ? '⚠ SPAM DETECTED' : '✅ LEGITIMATE EMAIL';

// Display confidence
document.getElementById('confidence').textContent =
(data.spamScore * 100).toFixed(2) + '%';

// Display detected patterns
document.getElementById('patterns').textContent =
data.detectedPatterns.length > 0 ?
data.detectedPatterns.join(', ') : 'None';

// Display individual model predictions
const modelsList = document.getElementById('models');
modelsList.innerHTML = `
<li>Naive Bayes:
${(data.modelPredictions.naiveBayes*100).toFixed(2)}%</li>
<li>SVM: ${(data.modelPredictions.svm*100).toFixed(2)}%</li>
<li>Random Forest:
${(data.modelPredictions.randomForest*100).toFixed(2)}%</li>
<li>Logistic Regression:
${(data.modelPredictions.logisticRegression*100).toFixed(2)}%
</li>
`;

// Show result with appropriate styling
const resultDiv = document.getElementById('result');
resultDiv.style.display = 'block';
resultDiv.className = data.isSpam ? 'result spam-result' :
'result ham-result';

} catch (error) { alert('Error analyzing email: ' + error.message);
} finally {
submitButton.textContent = '🔍 Analyze Email';
submitButton.disabled = false;
}
});
</script>
</body>
</html>
```

Application Screenshots


SpamGuard AI
Advanced Email Spam Detection System

Sender | Receiver 1 | Analytics

Total Emails	Legitimate	Spam Detected	Accuracy
1	0	1	94.7%

Compose Email

From: test@example.com

To: receiver@example.com

Subject: Meeting Tomorrow at 10 AM

Message: Hi team, just a reminder about our project meeting tomorrow at 10 AM in Conference Room B. Please bring your progress reports and review the attached agenda. Thanks! Best regards, John

Quick Test Examples

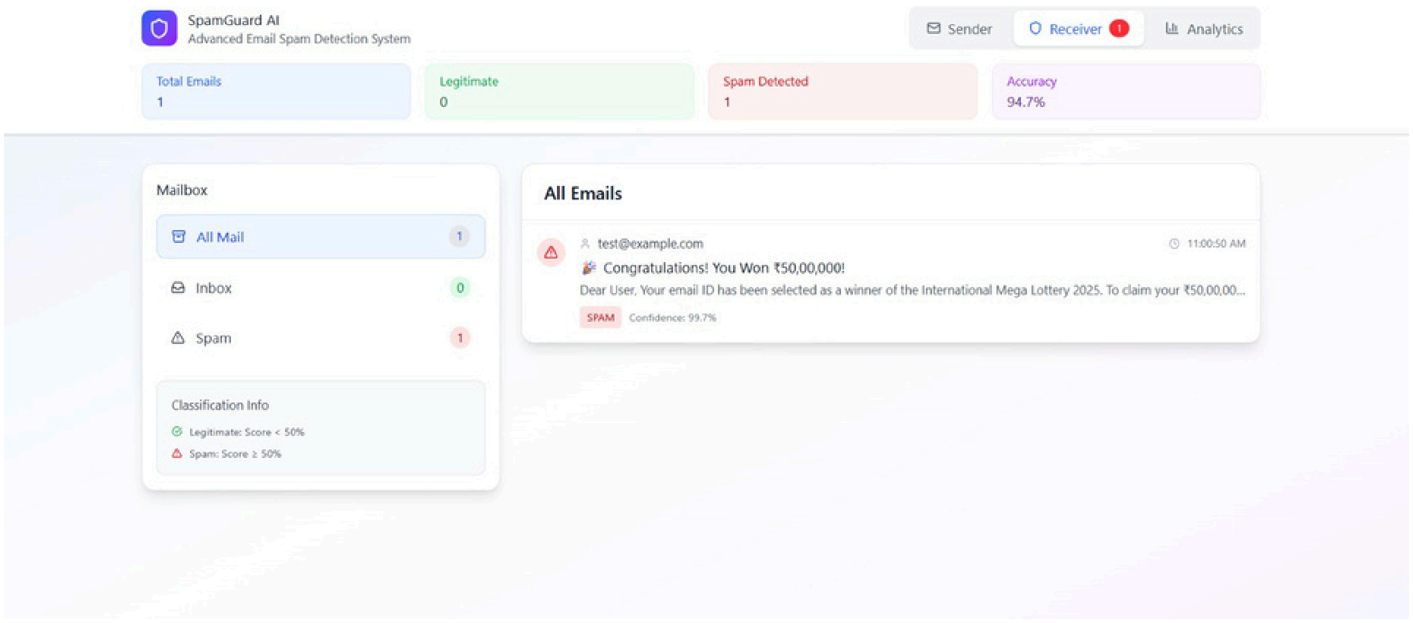
Click on any example below to test the spam detection system:

🎉 Congratulations! You Won ₹50,00,000!
Dear User, Your email ID has been selected as a winner of the International Mega Lottery 2025. To claim your ₹50,00,000 prize, reply with your full name, address, phone number, and bank details....

⚠️ Urgent! Your SBI Account is Blocked
Dear Customer, Your SBI account has been temporarily suspended due to suspicious activity. Please verify your account immediately by clicking the link below: http://verify-sbi-security.com Failure to...

Work From Home – Earn ₹20,000 per week!
Hello, We selected you for a high-paying remote job. No interview required. Just register and pay a ₹999 security fee to start earning. Limited slots! Regards, HR Team, Global Job Services

Meeting Tomorrow at 10 AM
Hi team, just a reminder about our project meeting tomorrow at 10 AM in Conference Room B. Please bring your progress reports and review the attached agenda. Thanks! Best regards, John


SpamGuard AI
Advanced Email Spam Detection System

Sender | Receiver 1 | Analytics

Total Emails	Legitimate	Spam Detected	Accuracy
1	0	1	94.7%

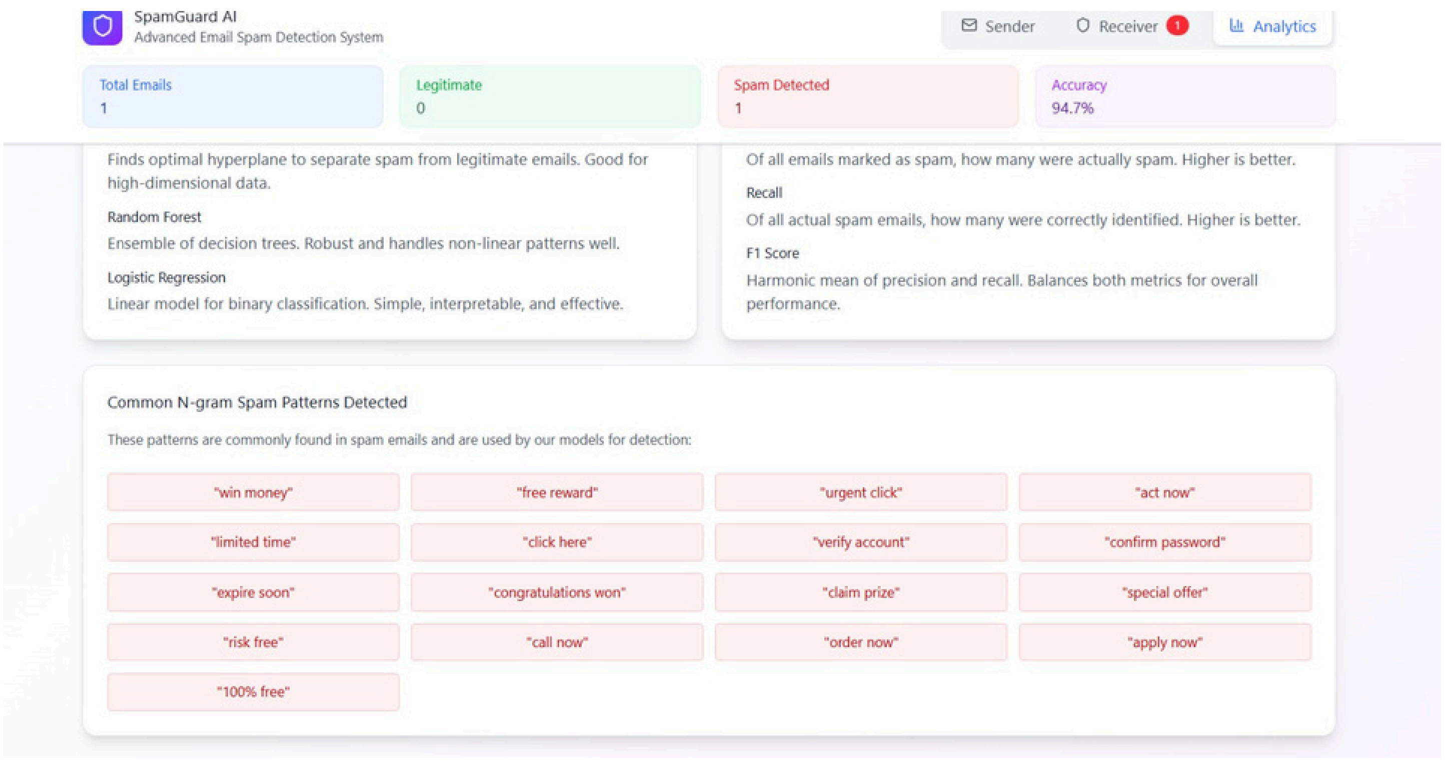
Mailbox

All Mail	1
Inbox	0
Spam	1

Classification Info
Legitimate: Score < 50%
Spam: Score ≥ 50%

All Emails

test@example.com — 11:00:50 AM
🎉 Congratulations! You Won ₹50,00,000!
Dear User, Your email ID has been selected as a winner of the International Mega Lottery 2025. To claim your ₹50,00,00...
SPAM Confidence: 99.7%



SpamGuard AI
Advanced Email Spam Detection System

Total Emails	Legitimate	Spam Detected	Accuracy
1	0	1	94.7%

Finds optimal hyperplane to separate spam from legitimate emails. Good for high-dimensional data.

Random Forest

Ensemble of decision trees. Robust and handles non-linear patterns well.

Logistic Regression

Linear model for binary classification. Simple, interpretable, and effective.

Of all emails marked as spam, how many were actually spam. Higher is better.

Recall

Of all actual spam emails, how many were correctly identified. Higher is better.

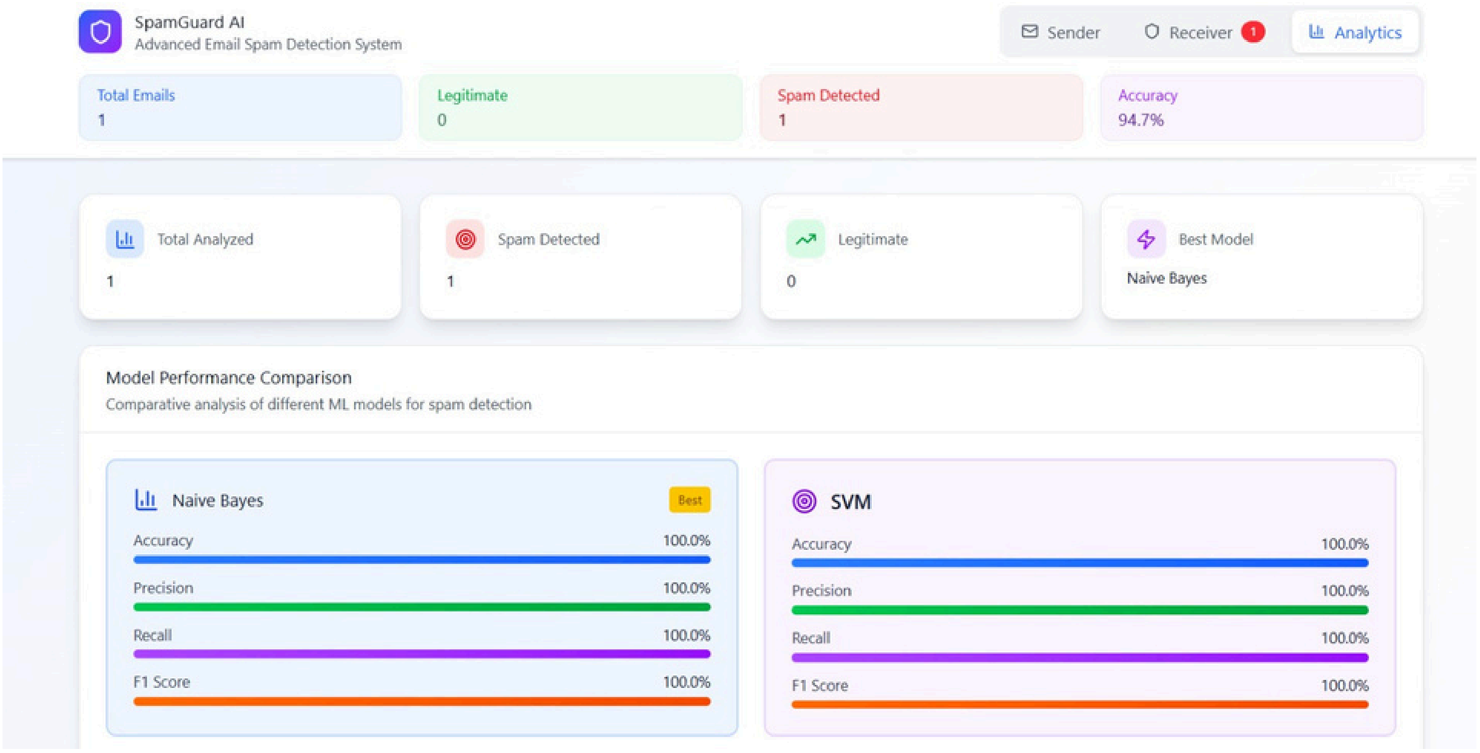
F1 Score

Harmonic mean of precision and recall. Balances both metrics for overall performance.

Common N-gram Spam Patterns Detected

These patterns are commonly found in spam emails and are used by our models for detection:

"win money"	"free reward"	"urgent click"	"act now"
"limited time"	"click here"	"verify account"	"confirm password"
"expire soon"	"congratulations won"	"claim prize"	"special offer"
"risk free"	"call now"	"order now"	"apply now"
"100% free"			



SpamGuard AI
Advanced Email Spam Detection System

Total Emails	Legitimate	Spam Detected	Accuracy
1	0	1	94.7%

Total Analyzed	Spam Detected	Legitimate	Best Model
1	1	0	Naive Bayes

Model Performance Comparison

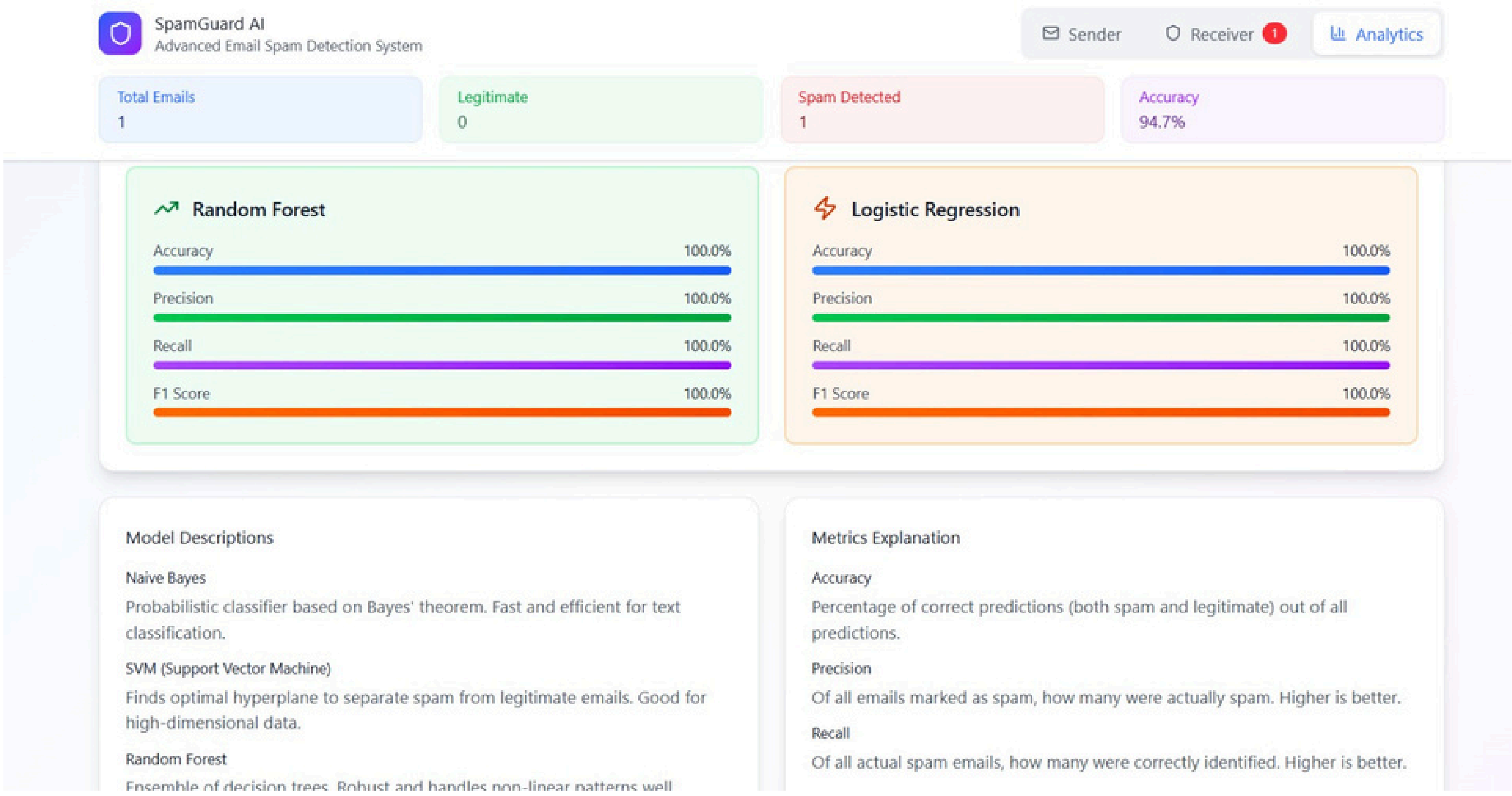
Comparative analysis of different ML models for spam detection

Naive Bayes — Best

Accuracy	100.0%
Precision	100.0%
Recall	100.0%
F1 Score	100.0%

SVM

Accuracy	100.0%
Precision	100.0%
Recall	100.0%
F1 Score	100.0%


SpamGuard AI
Advanced Email Spam Detection System

Sender | Receiver 1 | Analytics

Total Emails	Legitimate	Spam Detected	Accuracy
1	0	1	94.7%

Random Forest

Accuracy	100.0%
Precision	100.0%
Recall	100.0%
F1 Score	100.0%

Logistic Regression

Accuracy	100.0%
Precision	100.0%
Recall	100.0%
F1 Score	100.0%

Model Descriptions

Naive Bayes
Probabilistic classifier based on Bayes' theorem. Fast and efficient for text classification.

SVM (Support Vector Machine)
Finds optimal hyperplane to separate spam from legitimate emails. Good for high-dimensional data.

Random Forest
Ensemble of decision trees. Robust and handles non-linear patterns well

Metrics Explanation

Accuracy
Percentage of correct predictions (both spam and legitimate) out of all predictions.

Precision
Of all emails marked as spam, how many were actually spam. Higher is better.

Recall
Of all actual spam emails, how many were correctly identified. Higher is better.

Key Technical Concepts

N-grams Explained

- Purpose: Captures context and phrase patterns rather than just individual words
- Example progression:
  - "free" (unigram) - generic word
  - "free money" (bigram) - suspicious combination
  - "free money now" (trigram) - highly indicative of spam
- Advantage: Trigrams catch spam-specific phrases more effectively than individual words

TF-IDF Explained

- Term Frequency (TF): Measures how frequently a word appears in a specific document
- Inverse Document Frequency (IDF):