Email Classification API – Final Report

# 1. Introduction

In today’s enterprise environments, companies often receive large volumes of customer emails ranging from technical complaints to billing queries. Many of these emails contain personally identifiable information (PII) such as phone numbers, email IDs, Aadhar numbers, and card details.  
  
This project addresses two key problems:  
1. PII Masking: Before processing or storing sensitive emails, PII must be masked to comply with data privacy regulations.  
2. Email Classification: After masking, emails must be automatically classified into categories such as Billing Issue, Technical Support, Incident, etc., to streamline downstream processing.  
  
The goal was to implement a solution that solves both problems using traditional machine learning (without large language models) and expose it as a deployed API on Hugging Face Spaces.

# 2. Approach

## A. PII Masking Logic

We used custom regular expressions (regex) to identify various PII types such as email IDs, Aadhar numbers, credit card numbers, phone numbers, names, CVV, and expiry dates.  
  
To prevent overlapping matches (e.g., an Aadhar being misclassified as a phone number), each entity was assigned a priority. Lower-numbered priorities (like card number) overwrite overlapping entities with higher ones (like full name).  
  
Masked output replaces entities with tags such as [aadhar\_num], [email], [phone\_number], etc.

## B. Email Classification Logic

Once masked, emails are passed into a machine learning pipeline built with scikit-learn. The steps include:  
  
1. Text Vectorization: Using TfidfVectorizer to convert masked email text into numerical features.  
2. Model Selection: Initially used Naive Bayes, but it performed inconsistently and was sensitive to class imbalance.  
3. Final Model: Switched to RandomForestClassifier, which performed better with sparse vectors and unbalanced classes.  
4. Pipeline: Combined vectorizer and classifier into a Pipeline for clean training and inference.

# 3. Model Training & Evaluation

Dataset:  
- CSV file containing email samples labeled with categories.  
- Applied masking before vectorization.  
- Used 80/20 train-test split.  
  
Performance:  
- Random Forest showed improved accuracy and robustness.  
- The model was serialized using joblib into email\_classifier.pkl.

# 4. Deployment

Deployed using:  
- FastAPI for building the API  
- Uvicorn as the ASGI server  
- Dockerfile for containerization  
- Hugging Face Spaces for live deployment  
  
Endpoints:  
- POST /classify/: Accepts raw email and returns masked version, detected PII, and classification.  
- GET /: Returns welcome message.  
  
Live URL: https://shardaseque1206-email-classifier.hf.space

# 5. Challenges & Solutions

Challenge 1: Overlapping Regex Matches  
Issue: Overlapping patterns like phone vs. Aadhar caused mislabeling.  
Solution: Introduced priority-based entity resolution.  
  
Challenge 2: Naive Bayes Classifier Performed Poorly  
Issue: Naive Bayes yielded inconsistent predictions.  
Solution: Switched to Random Forest for better class balance handling.  
  
Challenge 3: Compatibility Issues with OpenAI  
Issue: OpenAI API was slow and resource-heavy on local machine.  
Solution: Used traditional ML (TF-IDF + Random Forest) for speed and simplicity.  
  
Challenge 4: PEP8 Violations and Code Quality  
Issue: Many initial code style violations.  
Solution: Used flake8 and black to clean and format code.  
  
Challenge 5: FastAPI Port Conflicts  
Issue: Port 8000 was occupied by another app.  
Solution: Changed to port 8001 or killed the conflicting process.  
  
Challenge 6: Deployment Errors on Hugging Face  
Issue: Circular imports and missing model files.  
Solution: Refactored project structure and compressed model to avoid size limit issues.

**Live API:**  
<https://shardaseque1206-email-classifier.hf.space>

GitHub Repository:  
<https://github.com/Sharda-jj/email_classifier>