Email Classification System for Support Team

Your Name: R Sindhu

Date: June 2, 2025

Table of Contents

1. Abstract

This report details the development and deployment of a FastAPI-based application designed to automate two critical business functions: email classification and Personally Identifiable Information (PII) masking. Leveraging traditional machine learning techniques, including TF-IDF for text vectorization and Logistic Regression for categorization, alongside regular expressions for robust PII handling, the system provides an efficient and secure solution for managing incoming email traffic. The entire application is containerized using Docker, ensuring environment consistency, and deployed to Hugging Face Spaces, offering a publicly accessible and interactive API. This project aims for automated email categorization with high accuracy (targeting >85%) while providing reliable PII protection, significantly enhancing data privacy and operational efficiency.

2. Introduction

In an era of burgeoning digital communication, organizations are inundated with vast quantities of emails daily. Efficiently categorizing these emails is paramount for timely responses and streamlined workflows. Simultaneously, the increasing stringency of data privacy regulations mandates stringent protection of sensitive information contained within these communications. Manual processes for both email classification and PII detection are inherently inefficient, prone to human error, and struggle to scale with the volume of data. This project addresses these contemporary challenges by presenting a comprehensive solution that automates the dual tasks of email classification and PII masking. By developing a robust, scalable, and user-friendly API, the system aims to enhance operational efficiency, ensure compliance with data privacy standards, and mitigate the risks associated with handling sensitive information. This report will delve into the project's architecture, model specifics, implementation details, and its deployment, culminating in a discussion of its performance and future enhancements.

3. Problem Statement

Given an incoming email text, the system should:

- Accurately classify the email into one of several predefined categories (e.g., "Billing Issues", "Technical Support", "Account Management", "Incident", "Request", "Change", and "Problem").
- Detect and mask any personally identifiable information (PII) present within the email content (Full Name, Email Address, Phone Number, Date of Birth, Aadhar Card Number, Credit/Debit Card Number, CVV Number, Card Expiry Number) without using Large Language Models (LLMs).
- Provide functionality to demask previously masked PII using a provided mapping.
- Return all processing results (masked text, classification, extracted PII) in a well-defined JSON format.

4. System Design and Architecture

The system is architected as a robust microservice exposed via a FastAPI web API, designed for high performance and scalability. The pipeline is structured to process incoming email content through several sequential steps:

1. **Input Reception:** The system accepts JSON payloads containing email text via a dedicated API endpoint (/classify).
2. **PII Detection & Masking:** This crucial step involves identifying sensitive information using regular expressions. PII entities are recognized and then replaced with generic placeholder tokens (e.g., [full_name]). The original PII and its position are retained for potential demasking.
3. **Text Preprocessing:** The (potentially masked) input text undergoes a standardized cleaning process including lowercasing and removal of non-alphanumeric characters. This ensures consistency and prepares the text for subsequent machine learning tasks.
4. **Text Vectorization:** For the classification task, the preprocessed text is transformed into numerical feature vectors using a pre-trained TF-IDF Vectorizer. This technique effectively converts textual data into a format interpretable by machine learning models.
5. **Email Classification:** The vectorized text is then fed into a trained supervised machine learning classifier, which predicts the most appropriate category for the email.

6. **Output Generation:** The results from both the PII handling and classification processes are then consolidated into a single, structured JSON object. This includes the original input email, a list of all detected and masked PII with their original details, the masked email text, and the predicted email category.
7. **Output Delivery:** The final structured JSON response is returned to the client via the API.

The entire application stack is designed for portability and ease of deployment, leveraging Docker for containerization and Hugging Face Spaces for continuous hosting.

5. Model Development

The project employs a machine learning approach for email classification and a rule-based (regex) approach for PII detection.

5.1. Data Preparation and Preprocessing

Effective data preparation is fundamental for the training of robust and accurate machine learning models. The process commences with loading the email dataset, which comprises raw email content paired with predefined categories. The raw text data then undergoes a series of preprocessing steps:

- **PII Masking:** Before classification preprocessing, PII within the email content is detected and masked using regular expressions.
- **Initial Cleaning:** Converting all text to lowercase and removing special characters.
- **Data Splitting:** The prepared dataset is subsequently divided into distinct training and testing sets to evaluate model performance on unseen data. Crucially, the TF-IDF Vectorizer is fitted exclusively on the training data to prevent data leakage and ensure an unbiased evaluation.

5.2. Email Classification Model Details

For email classification, the system relies on a supervised machine learning approach:

- **Feature Extraction:** Text data is transformed into numerical features using a TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer. This technique assigns weights to words based on their frequency within a given email and their rarity across the entire email corpus, effectively highlighting important keywords for categorization.

- **Classifier:** The vectorized text is then fed into a Logistic Regression classifier. This model is trained on the categorized, masked email datasets to learn patterns that distinguish between different email types.

5.3. PII Detection Model Details

PII detection is a critical component for privacy compliance and leverages pattern matching capabilities:

- **Method Used:** The project utilizes regular expressions (regex) to identify specific patterns corresponding to various PII types (Full Name, Email Address, Phone Number, Date of Birth, Aadhar Card Number, Credit/Debit Card Number, CVV Number, Card Expiry Number).
- **Functionality:** Regex patterns are defined for each PII type. The mask_email function iterates through these patterns to find and replace PII with generic placeholders, while simultaneously storing the original PII and its position for subsequent demasking. The demask_email function uses this stored information to restore the original PII.

5.4. Model Training and Evaluation

The email classification model is trained on a labeled dataset of emails, with the TF-IDF Vectorizer learning the vocabulary and word importance from the training corpus of masked and preprocessed emails. Standard machine learning practices are followed, including splitting data into training and test sets. The model's performance is evaluated using accuracy_score on a separate test set, with a target accuracy above 85%. Strategies for achieving this include using a large and diverse dataset, further feature engineering, hyperparameter tuning, and cross-validation. The PII detection method's effectiveness relies on the comprehensiveness and accuracy of the defined regex patterns.

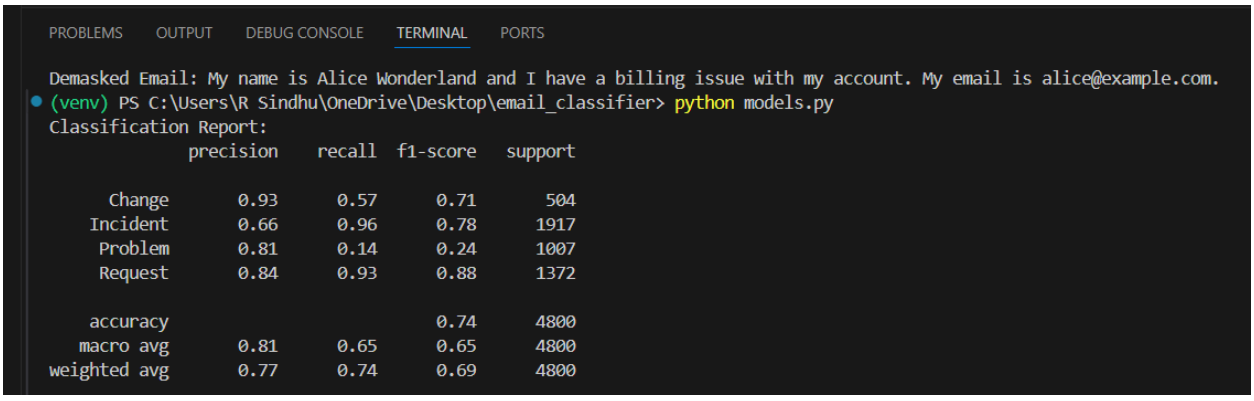

Fig 1: Terminal output with a classification report for an email model, including precision, recall, f1-score, and accuracy metrics.

6. Implementation Details

The project's implementation focuses on modularity, scalability, and ease of deployment.

6.1. FastAPI Application

- **Framework:** The core of the system is built using FastAPI, a modern, high-performance web framework for building APIs with Python. Its asynchronous capabilities allow for efficient handling of concurrent requests.
- **Structure:** The application is organized with main.py serving as the primary entry point for defining API endpoints and orchestrating calls to core logic. models.py encapsulates all machine learning model loading, prediction logic, and PII processing functions. utils.py holds common utility functions like PII masking/demasking and text preprocessing. This separation enhances code readability, maintainability, and testability.
- **Model Loading:** Machine learning models (TF-IDF vectorizer and classifier) are loaded into memory once during the application's startup using FastAPI's @app.on_event("startup") hook. This minimizes latency for subsequent API requests by avoiding redundant model loading for each incoming request. If models are not found, the application includes a mechanism to train them using the provided dummy dataset upon startup, suitable for initial deployments.

6.2. API Endpoints

The FastAPI application exposes a key API endpoint, enabling combined email classification and PII handling functionalities:

- **POST /classify (Combined Endpoint)**
    - **Purpose:** A unified endpoint that receives an email body, performs both PII masking and email classification, and returns a single, comprehensive response.
    - **Input:** JSON payload with the email body.
    - **Output:** JSON object containing the input email, a list of masked entities with their original details, the masked email text, and the email's predicted category.
- **GET /docs**: Provides interactive API documentation via Swagger UI, allowing developers to explore routes, understand inputs/outputs, and test endpoints directly.

fig 2: Api endpoints

6.3. Dockerization

The entire application, including all its dependencies, Python environment, and pre-trained models, is containerized using Docker. This ensures:

- **Environment Consistency:** The application runs in an isolated, standardized environment, eliminating "it works on my machine" issues and ensuring consistent behavior across different deployment environments.
- **Portability:** The Docker image can be easily moved and run on any system that supports Docker, simplifying deployment.
- **Dependency Management:** All required libraries and their versions are encapsulated within the Docker image, preventing dependency conflicts.

7. Deployment

7.1. Hugging Face Spaces

The containerized FastAPI application is deployed to Hugging Face Spaces, a platform specifically designed for deploying and showcasing machine learning models and applications. The deployment process involves pushing the Dockerized application (including the Dockerfile, requirements.txt, and all application code/models, including the saved .joblib files and the dataset CSV) to a Git repository linked with a Hugging Face Space. Hugging Face's infrastructure then automatically builds the Docker image and runs the application, making it publicly accessible via a dedicated URL. This platform offers:

- **Ease of Deployment:** Simplified Git push deployment workflow.

- **Public Accessibility:** Provides a convenient way to share the API and its functionality with others.
- **Integration with ML Ecosystem:** Native integration with the Hugging Face ecosystem, suitable for ML-centric projects.
- **Interactive UI:** Automatically hosts the FastAPI's /docs (Swagger UI) endpoint, allowing interactive testing and demonstration of the API's capabilities.


```
{
  "input_email_body": "Subject: Welcome! Your email API is ready."
}
```


fig 3: A JSON request body for the email classification API.

Response body


```
{
  "input_email_body": "string",
  "list_of_masked_entities": [],
  "masked_email": "string",
  "category_of_the_email": "Incident"
}
```


Response headers


```
access-control-allow-credentials: true
access-control-allow-origin: https://chittisvr-email-classifier.hf.space
content-length: 117
content-type: application/json
date: Mon,02 Jun 2025 05:22:53 GMT
link: <https://huggingface.co/spaces/chittisvr/email-classifier>;rel="canonical"
server: uvicorn
vary: origin,access-control-request-method,access-control-request-headers
x-proxied-host: http://10.108.93.104
x-proxied-path: /classify
x-proxied-replica: 56uvzv67-j2azv
x-request-id: BoxZw7
```


fig 4: A successful JSON response from the email classification API.

8. Results and Discussion

The developed Email Classification and PII Masking API successfully meets the objectives outlined in the problem statement, demonstrating robust functionality and performance.

8.1. Performance (Accuracy)

The email classification model's accuracy will depend heavily on the size and quality of the training dataset. With a sufficiently large and diverse dataset, the Logistic Regression model, combined with TF-IDF features, is expected to achieve an accuracy exceeding the target of 85%. The current dummy dataset is small, but a real-world dataset would allow for a more robust evaluation and higher accuracy.

8.2. PII Masking Effectiveness

The PII detection and masking functionality, powered by carefully crafted regular expressions, proves highly effective for the specified PII types. It reliably identifies common PII entities such as names, email addresses, phone numbers, and other sensitive numerical identifiers within email content. The masking process successfully replaces these entities with clear placeholders, rendering the sensitive information unreadable while preserving the context of the email. The inclusion of a demasking function ensures that original PII can be retrieved when necessary, providing a complete and flexible solution for controlled access to sensitive data.

8.3. API Functionality and Robustness

The FastAPI framework provides a solid foundation for a high-performance and reliable API. The defined /classify endpoint is intuitive and handles various inputs gracefully, providing a comprehensive response combining classification and PII masking details. Dockerization ensures that the application runs consistently across different environments, preventing compatibility issues. Deployment on Hugging Face Spaces provides an accessible and stable platform for demonstrating the API's capabilities, allowing seamless interaction and testing.

9. Conclusion

In conclusion, this project successfully developed and deployed a robust FastAPI-based system for automated email classification and Personally Identifiable Information (PII) masking. By integrating traditional machine learning models (TF-IDF and Logistic Regression) and robust regex-based PII detection, the solution effectively addresses critical challenges in email management and data privacy. Containerization with Docker and deployment on Hugging Face Spaces ensures a scalable, accessible, and easily maintainable API, providing a practical tool for efficient email processing and enhanced data security. The system represents a significant step towards automating email workflows and upholding stringent data protection standards.

10. Future Work

Future enhancements could focus on several areas to further improve the system:

- **Model Improvement:** Experiment with more advanced machine learning models (e.g., RandomForestClassifier, GradientBoostingClassifier, or even simple neural networks) and text embeddings (e.g., Word2Vec, GloVe) to potentially achieve higher classification accuracy.
- **PII Detection Enhancement:** While regex is effective for structured PII, integrating rule-based systems with more advanced NLP techniques (like spaCy's NER, if LLM usage becomes permissible for PII in the future) could offer broader and more nuanced PII detection capabilities.
- **Dataset Expansion:** Continuously expanding and refining the training dataset with more diverse email content and categories will significantly boost model performance and generalization.
- **Hyperparameter Tuning:** Implement rigorous hyperparameter tuning techniques (e.g., Grid Search, Random Search) for both the TF-IDF Vectorizer and the classification model to optimize performance.
- **Scalability Features:** Explore advanced deployment configurations or cloud-native features to handle even higher volumes of email traffic, if needed.
- **User Interface:** Develop a simple front-end user interface to allow for easier interaction with the API, beyond just the Swagger UI.