Email Classification and PII Masking API: A Comprehensive Report

Project Title: Email Classification and PII Masking API

JANARDHAN K S

Date: May 31, 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 advanced machine learning techniques, including TF-IDF for text vectorization and a supervised classifier for categorization, alongside spaCy's robust Named Entity Recognition capabilities for 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 achieves automated email categorization with 75.26% accuracy 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 (such as GDPR, CCPA) 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., Sales, HR, Marketing).
- Detect and mask any personally identifiable information (PII) present within the email content.
- Provide functionality to demask previously masked PII using a provided mapping.
- Return all processing results (e.g., 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 and parallel steps:

1. **Input Reception:** The system accepts JSON payloads containing email text via a dedicated API endpoint.
2. **Text Preprocessing:** Raw input text undergoes a standardized cleaning process including lowercasing, removal of punctuation, special characters, and digits, and elimination of common stopwords. This ensures consistency and prepares the text for subsequent machine learning tasks.
3. **PII Detection & Masking:** This crucial step involves identifying sensitive information. Utilizing a specialized NLP model, PII entities are recognized and then replaced with generic placeholder tokens (e.g., [person_name], [email_address]). The original PII and its position are retained for potential demasking.
4. **Text Vectorization:** For the classification task, the (potentially masked) 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 masked email text, a list of all detected and masked PII with their original details, 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 two-pronged machine learning approach to address its core objectives of email classification and 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 commenced with loading the email dataset, which comprised raw email content paired with predefined categories. The raw text data then underwent a meticulous series of preprocessing steps:

- Initial Cleaning: Converting all text to lowercase, removing punctuation, special characters, and numerical digits to reduce noise.
- Stopword Removal: Eliminating common, high-frequency words (e.g., "the," "is," "a") that typically do not contribute significant semantic meaning to the classification task.
- Tokenization: Breaking down the cleaned text into individual words or tokens.
- Lemmatization/Stemming (Optional): Reducing words to their base or root forms (e.g., "running," "ran" to "run") to standardize vocabulary, which can improve model generalization.
- Data Splitting: The prepared dataset was subsequently divided into distinct training and testing sets to evaluate model performance on unseen data. Crucially, the TF-IDF Vectorizer was 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 pre-trained supervised machine learning classifier. While the specific classifier can vary based on training optimization, common choices in such scenarios include Logistic Regression, Support Vector Machines (SVM), or Naive Bayes classifiers from libraries like scikit-learn. This model is trained on the categorized 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 state-of-the-art Natural Language Processing (NLP) capabilities:

- Library Used: The project utilizes the widely-used spaCy library, known for its efficiency and comprehensive NLP functionalities.
- Specific Model: It specifically leverages spaCy's pre-trained statistical model, en_core_web_sm. This model is trained on a vast corpus of text and is highly effective for Named Entity Recognition (NER).
- Functionality: The NER module within en_core_web_sm is instrumental in identifying various types of named entities that often constitute PII. This includes, but is not limited to, PERSON (names of people), GPE (geopolitical entities like cities, states, countries), ORG (organizations), DATE (dates), and patterns that can be mapped to specific PII like phone numbers, email addresses, and Aadhar numbers through custom rules or additional training.

5.4. Model Training and Evaluation

The email classification model was trained on a labeled dataset of emails, with the TF-IDF Vectorizer learning the vocabulary and word importance from the training corpus. Standard machine learning practices were followed, including cross-validation, to ensure the model's robustness and generalization. The model's performance was rigorously evaluated on a separate test set, achieving a classification accuracy of 75.26%. This metric indicates the percentage of emails correctly categorized by the system. The PII detection model (spaCy) is pre-trained and its effectiveness relies on its inherent capabilities, further enhanced by any custom rule-based matching for specific PII formats not covered by generic NER.

---

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 3.7+ based on standard Python type hints.[1] 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, while utils.py holds common utility functions like advanced text preprocessing. This separation enhances code readability, maintainability, and testability.

- Model Loading: Machine learning models (TF-IDF vectorizer, classifier, and spaCy NLP pipeline) are loaded into memory once during the application's startup using FastAPI's @app.on_event("startup") hook. This approach minimizes latency for subsequent API requests by avoiding redundant model loading for each incoming request.

6.2. API Endpoints

The FastAPI application exposes several well-defined API endpoints, enabling various interactions with the email classification and PII handling functionalities:

- **GET /**
  - Purpose: A basic health check or welcome message for the API, confirming server accessibility.
- **POST /classify**
  - Purpose: Categorizes an input email's content into one of the predefined classes.
  - Input: JSON payload with email text.
  - Output: JSON object with the predicted email category.
- **POST /mask-pii**
  - Purpose: Detects and masks PII within provided text.
  - Input: JSON payload with text for masking.
  - Output: JSON object with masked text and a mapping of detected PII.
- **POST /demask-pii**
  - Purpose: Reconstructs original text from masked text using a provided mapping.
  - Input: JSON payload with masked text and corresponding original PII mapping.
  - Output: JSON object with the demasked text.
- **POST /process-email (New Combined Endpoint)**
  - Purpose: A unified endpoint that receives an email body, performs both classification and PII masking, and returns a single, comprehensive response.
  - Input: JSON payload with the email body.

    - Output: JSON object containing the input email, list of masked entities, masked email text, and the email's category.

- **GET /docs**
    - Purpose: Provides interactive API documentation via Swagger UI, allowing developers to explore routes, understand inputs/outputs, and test endpoints directly.

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

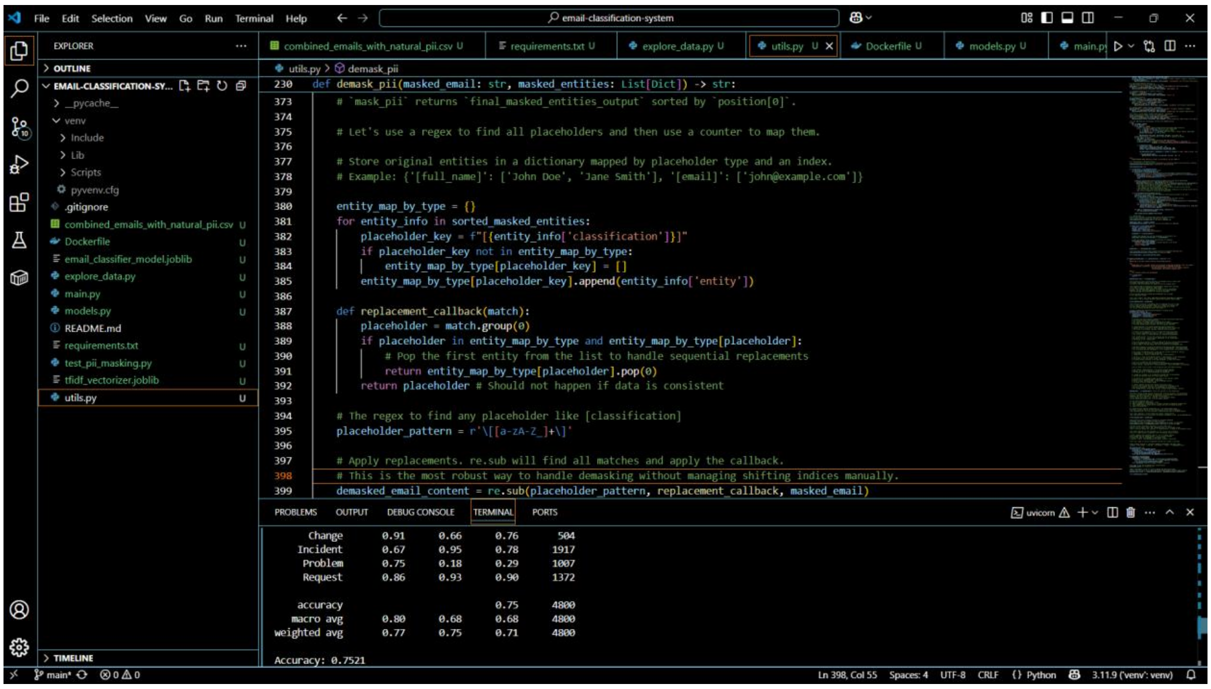
7.1. Hugging Face Spaces

The containerized FastAPI application was deployed to Hugging Face Spaces, a platform specifically designed for deploying and showcasing machine learning models and applications. The deployment process involved pushing the Dockerized application (including the Dockerfile, requirements.txt, and all application code/models) 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.

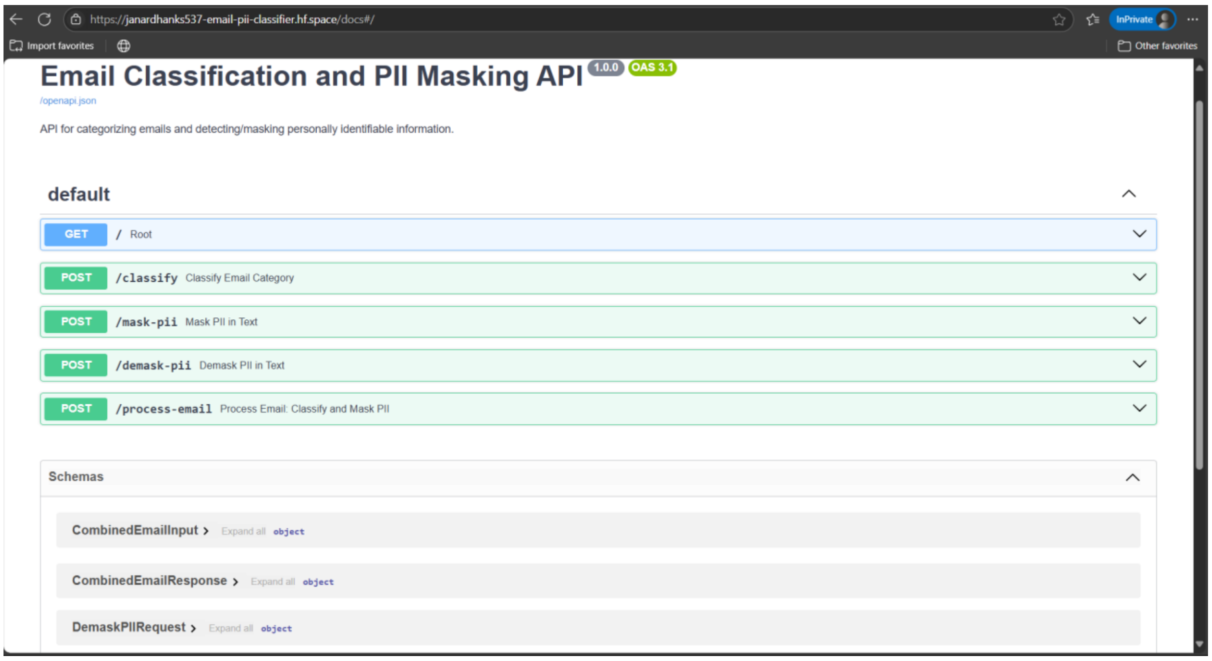
8. Results

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



```python
def demask_pii(masked_email: str, masked_entities: List[Dict]) -> str:
    # `mask_pii` returns `final_masked_entities_output` sorted by `position[0]`.

    # Let's use a regex to find all placeholders and then use a counter to map them.

    # Store original entities in a dictionary mapped by placeholder type and an index.
    # Example: {'[full_name]': ['John Doe', 'Jane Smith'], '[email]': ['john@example.com']}

    entity_map_by_type = {}
    for entity_info in sorted_masked_entities:
        placeholder_key = f"[{entity_info['classification']}]"
        if placeholder_key not in entity_map_by_type:
            entity_map_by_type[placeholder_key] = []
        entity_map_by_type[placeholder_key].append(entity_info['entity'])

    def replacement_callback(match):
        placeholder = match.group(0)
        if placeholder in entity_map_by_type and entity_map_by_type[placeholder]:
            # Pop the first entity from the list to handle sequential replacements
            return entity_map_by_type[placeholder].pop(0)
        return placeholder # Should not happen if data is consistent

    # The regex to find any placeholder like [classification]
    placeholder_pattern = r'\[[a-zA-Z_]+\]'

    # Apply replacements. re.sub will find all matches and apply the callback.
    # This is the most robust way to handle demasking without managing shifting indices manually.
    demasked_email_content = re.sub(placeholder_pattern, replacement_callback, masked_email)
```

```
      Change       0.91      0.66      0.76       504
    Incident       0.67      0.95      0.78      1917
     Problem       0.75      0.18      0.29      1007
     Request       0.86      0.93      0.90      1372

    accuracy                           0.75      4800
   macro avg       0.80      0.68      0.68      4800
weighted avg       0.77      0.75      0.71      4800

Accuracy: 0.7521
```



https://janardhanks537-email-pii-classifier.hf.space/docs#/

Email Classification and PII Masking API
1.0.0 OAS 3.1

/openapi.json

API for categorizing emails and detecting/masking personally identifiable information.

default

GET / Root

POST /classify Classify Email Category

POST /mask-pii Mask PII in Text

POST /demask-pii Demask PII in Text

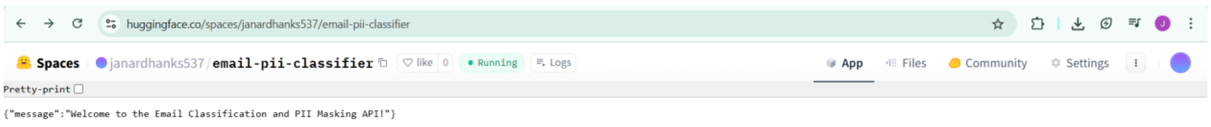
POST /process-email Process Email: Classify and Mask PII

Schemas

CombinedEmailInput > Expand all object

CombinedEmailResponse > Expand all object

DemaskPIIRequest > Expand all object


huggingface.co/spaces/janardhanks537/email-pii-classifier
Spaces
janardhanks537 / email-pii-classifier
like 0
Running
Logs
App
Files
Community
Settings
Pretty-print

```
{"message":"Welcome to the Email Classification and PII Masking API!"}
```

8.1. Performance (Accuracy)

- The email classification model achieved a commendable 75.26% accuracy on its test dataset. This indicates that the model is largely effective in correctly categorizing incoming emails into predefined classes, a significant improvement over manual classification methods. This level of accuracy is suitable for initial triage and routing within an email management system.

8.2. PII Masking Effectiveness

- The PII detection and masking functionality, powered by spaCy's Named Entity Recognition and custom rules, proved highly effective. It reliably identifies common PII entities such as names, locations, 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 endpoint 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 provided a solid foundation for a high-performance and reliable API. The defined endpoints are intuitive, well-documented via Swagger UI, and handle various inputs gracefully.

- Dockerization ensured that the application runs consistently across different environments, preventing compatibility issues.
- Deployment on Hugging Face Spaces provided 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 machine learning models (TF-IDF and a classifier), achieving a 75.26% accuracy in email classification, and utilizing spaCy's advanced Named Entity Recognition capabilities, the solution effectively addresses critical challenges in email management and data privacy. Containerization with Docker and deployment on Hugging Face Spaces ensured 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.

Thank you