**Project Report**

**Email Classification and PII Masking API**

**Date:** May 31, 2024

**Author:** Neethu J

1. **Introduction**

In today's digital landscape, organizations handle vast amounts of email data, which often contains sensitive information (Personally Identifiable Information - PII). Efficiently processing and categorizing these emails, while also ensuring the protection of sensitive data, is crucial for various applications, including customer support routing, compliance, and data privacy.This project addresses the problem of automating the classification of incoming emails into predefined categories (e.g., Incident, Request, Change, Problem) and the simultaneous identification and masking of PII within the email content. The solution is implemented as a web API using FastAPI, providing a flexible and scalable service that can be integrated into existing systems.

1. **Approach Taken**

The project adopts a modular approach, separating the concerns of PII handling, classification logic, and the web API interface.

* **PII Masking:** PII masking is performed using regular expressions and string matching techniques implemented in the utils.py module. This approach allows for the identification of common PII patterns such as email addresses, phone numbers, and names based on predefined patterns. Once identified, the sensitive information is replaced with masked placeholders (e.g., [EMAIL\_MASKED], [PHONE\_MASKED]) to protect privacy while retaining the overall structure and context of the email.
* **Email Classification:** For demonstration purposes, a DummyClassifier class was implemented in dummy\_classifier.py. This class simulates the output of a real classifier by randomly assigning one of the predefined categories (Incident, Request, Change, Problem) to an incoming email. This allows for the testing of the API's workflow and data handling without the need for a fully trained machine learning model during the initial development and deployment phases. In a production environment, this module would be replaced with a sophisticated text classification model.
* **API Implementation:** The web API is built using the FastAPI framework (app.py). FastAPI was chosen for its high performance, ease of use, and automatic generation of interactive API documentation (Swagger UI). The API exposes a /classify endpoint that accepts an email body in JSON format and returns the assigned classification, the masked email content, and the extracted PII entities. A root endpoint (/) was also added to provide a basic status check.

1. **Model Selection and Training Details (Production Considerations)**

While this project utilizes a dummy classifier, a production-ready system would require a trained machine learning model for accurate email classification. The selection of a model would depend on factors such as the complexity of the classification task, the volume of data, and computational resources. Potential models include:

* **Traditional Machine Learning Models:** Naive Bayes, Support Vector Machines (SVM), or Logistic Regression with TF-IDF or CountVectorizer for feature extraction.
* **Deep Learning Models:** Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), or pre-trained transformer models (e.g., BERT, RoBERTa) fine-tuned on a custom email dataset.

Training a production model would involve:

* **Data Collection and Labeling:** Gathering a diverse dataset of emails and manually labeling them with the appropriate categories.
* **Preprocessing:** Cleaning the text data, handling punctuation, stopwords, and potentially performing stemming or lemmatization.
* **Feature Extraction/Embedding:** Converting the text into numerical representations that the model can understand.
* **Model Training:** Training the selected model on the labeled dataset.
* **Evaluation:** Assessing the model's performance using metrics like accuracy, precision, recall, and F1-score on a separate test set.

1. **Challenges Faced and Solutions Implemented**

Several challenges were encountered during the project, particularly during the deployment phase to Hugging Face Spaces:

* **GitHub Push Issues:** Initial attempts to push the local repository to GitHub failed due to authentication issues. This was resolved by configuring Git to use a personal access token instead of password authentication.
* **Hugging Face Space Creation:** It was necessary to manually create the Hugging Face Space on the platform before pushing the code. Attempting to push to a non-existent space resulted in "Repository not found" errors.
* **Hugging Face Configuration:** Hugging Face Spaces requires specific metadata in the README.md file for proper configuration, including the SDK used (docker) and the main application file (app.py). Adding this configuration resolved build issues related to missing setup information.
* **Application Entry Point:** Hugging Face Spaces defaults to looking for app.py as the main application file. The initial structure used main.py. This was resolved by creating a simple app.py that imports the FastAPI application instance from main.py (or moving the app definition directly to app.py) and updating the Dockerfile accordingly.
* **Dummy Classifier Method Name:** The initial implementation in app.py called a classify method on the DummyClassifier instance, but the class only had a predict method. This runtime error was fixed by changing the method call to classifier.predict() and adjusting the input format to match the predict method's expectation (a list of inputs).

1. **Final Output**

The implemented API is deployed on Hugging Face Spaces and is accessible for testing.

* **Base URL:** https://neethu19-emailclassification.hf.space
* **API Documentation (Swagger UI):** https://neethu19-emailclassification.hf.space/docs

**Classification Endpoint:**

* **Method:** POST
* **Path:** /classify
* **Request Body:** application/json with the structure {"email\_content": "your email text here"}
* **Response Body:** application/json with the structure {"classification": "category", "masked\_content": "masked email text", "entities": {"emails": [...], "names": [...], "phone\_numbers": [...]}}

This API endpoint can be used to send email content and receive the classification and PII processed output. The Swagger UI provides an interactive way to test this endpoint directly from your browser.