Email Classification for Support Team

Objective:

The goal of this assignment is to design and implement an **email classification system** for a company's support team. The system should categorize incoming support emails into predefined categories while ensuring that **personal information (PII) is masked** before processing. After classification, the masked data should be restored to its original form.

Problem Statement:

You are required to develop an **email classification system** with the following key functionalities:

1. **Email Classification:**
    - Build a model to classify support emails into different categories (e.g., Billing Issues, Technical Support, Account Management, etc.).
    - You may use **traditional machine learning models, deep learning models, or Large Language Models (LLMs) for classification**.

2. **Personal Information Masking (Without LLMs):**
    - Before processing an email, all personally identifiable information (PII) and Payment Card Industry (PCI) must be masked.
      Below are the fields that should be detected, the Entity Field should be the same as what is mentioned in the brackets for the evaluation system to assess your solution:
        - Full Name (“full_name”)
        - Email Address (“email”)
        - Phone number (“phone_number”)
        - Date of birth (“dob”)
        - Aadhar card number (“aadhar_num”)
        - Credit/ Debit Card Number (“credit_debit_no”)
        - CVV number (“cvv_no”)
        - Card expiry number (“expiry_no”)
    - The masking process **must not** rely on Large Language Models (LLMs).

- You **can use any Named Entity Recognition (NER) method**, machine learning models (other than LLMs), **regular expressions (Regex), or any custom methods** for masking PII.
- Example:
  - Input: *"Hello, my name is John Doe, and my email is johndoe@example.com."*
  - Masked: *"Hello, my name is [full_name], and my email is [email]."*

3. **API Deployment:**
   - The final solution **must be exposed as an API**.
   - The API should allow users to send an email as input and receive the classified category along with the demasked email as output.
   - Your solution will be assessed on the deployed API.

Assignment Tasks:

1. Data Collection & Preprocessing:

- Use the provided dataset of emails containing different types of support requests.
- Identify and mask personal information using **NER-based approaches, machine learning models (excluding LLMs), Regex, or any custom text processing methods**.
- Store the original data securely for later demasking.

2. Model Selection & Training:

- Choose an appropriate classification model from the following options:
  - **Traditional ML models**: Naïve Bayes, SVM, Decision Trees, Random Forest, etc.
  - **Deep learning models**: LSTMs, Transformers, CNNs, etc.
  - **Large Language Models (LLMs)**: OpenAI GPT, BERT, RoBERTa, T5, etc.

- You **can use any libraries** you prefer for model training and deployment.
- Train the model on the **provided dataset** to classify emails into predefined support categories.

3. System Integration:

- Implement a pipeline where:
  - Emails are received and **masked**.
  - Masked emails are passed to the classification model.

- The classification result is obtained.

4. API Development & Deployment:

- Develop an API that exposes the solution. The API should:
  - Accept an email as input.
  - Mask the PII before processing.
  - Classify the email into a category.

- You can use **Flask, FastAPI, Django REST Framework, or any other API framework** of your choice.

Submission Guidelines

- Submit the **GitHub repository link** with a well-structured codebase.
- The repository should include:
  - `app.py` (or equivalent main script)
  - Requirements file (`requirements.txt` or `environment.yml`)
  - **README** with setup and usage instructions
  - `models.py` containing the model training and utility functions
  - `utils.py` containing the utility function and code
  - `api.py` to support the development of APIs.
- Deploy the application on Hugging Face Spaces and provide the **deployment link**.
- Ensure code is properly commented and follows best practices.
- Code will automatically be tested for quality. Please follow PEP8 guidelines.

5. Generate Output Using Provided Data:

- Use the given dataset to **process the emails**, apply **PII masking**, classify them, and **return the results**.

Deliverables:

1. **Code Implementation:**
   - Python scripts for PII masking, email classification.
   - API implementation and deployment instructions.
   - Proper documentation for each module.

2. **Report (2-3 Pages):**
   - Introduction to the problem statement.
   - Approach taken for PII masking and classification.
   - Model selection and training details.

    - Challenges faced and solutions implemented.

3. **Final Output:**
    - API endpoint details for testing.

Tools & Libraries Suggested:

- **Python:** Pandas, NumPy, Scikit-learn, Regex, NLTK, Spacy
- **ML/DL Models:** Scikit-learn, TensorFlow, PyTorch, Hugging Face Transformers
- **NER & Text Processing:** Spacy, NLTK, Regex, Custom NLP methods
- **LLMs for Classification:** OpenAI GPT, BERT, RoBERTa, T5
- **API Development:** Flask, FastAPI, Django REST Framework
- **Other:** Any additional libraries or tools that help achieve the objectives

Evaluation Criteria

Your submission will be assessed based on the following:

1. **API Deployment** – The API must be correctly deployed and accessible on hugging face spaces.

2. **Code Quality** – The implementation should adhere to **PEP8 guidelines** for readability and maintainability.

3. **Report Detail** – The report should clearly explain your approach, methodology, and implementation details.

4. **GitHub Submission** – A **GitHub repository link** must be included in your submission.

5. **Test Case Coverage** – The API must handle hidden test cases beyond the provided ones; failure to do so will result in rejection.

Strict API Format Requirements

- The API **must** be a `POST` request accepting an email body as input.

- The output **must** be in **JSON format** with the following structure:

```
{
  "input_email_body": "string containing the email",
  "list_of_masked_entities": [
    {
```

```
      "position": [start_index, end_index],
      "classification": "entity_type",
      "entity": "original_entity_value"
    }
  ],
  "masked_email": "string containing the masked email",
  "category_of_the_email": "string containing the class"
}
```

Note: If the output format does not strictly follow this structure, your submission will be **automatically rejected**, as we use an automated system for classification.