Email Classification with PII Masking

◆ 1. Introduction to the Problem Statement

Customer support teams often receive large volumes of emails, many of which contain sensitive personal data and require categorization for appropriate resolution. These emails vary from billing queries and technical issues to account management requests.

Manual classification not only introduces delays but also poses risks of mishandling personal information. To streamline this process and ensure data privacy, this project aims to:

1. Automatically **detect and mask sensitive PII** (Personally Identifiable Information) using non-LLM methods.
2. **Classify the masked email content** into predefined support categories.
3. Restore the original content securely when needed and **provide an API interface** for real-time usage.

◆ 2. Approach Taken for PII Masking and Classification

✓ *PII Masking*

Since the use of Large Language Models (LLMs) was restricted for PII masking, we used **Regex-based Named Entity Recognition (NER)**. The following patterns were defined:

- **Full Name**: Detected using capitalized first and last names.
- **Email Address**: Detected using RFC-like patterns.
- **Phone Number**: Recognized as 10-digit numeric strings.
- **Date of Birth (DOB)**: Patterns like `dd-mm-yyyy` or `dd/mm/yyyy`.
- **Aadhar Number**: Formatted as `XXXX XXXX XXXX`.
- **Credit/Debit Card Number**: `XXXX-XXXX-XXXX-XXXX` or similar.
- **CVV**: Three-digit numbers.
- **Expiry Date**: Recognized as `MM/YY` or `MM/YYYY`.

Each matched entity is replaced with a placeholder tag like `[email]`, `[phone_number]`, etc., and its position and original value are saved in a structured format.

✓ *Email Classification*

Masked emails are then fed into a **machine learning classification pipeline**, which classifies them into categories such as:

- Billing Issues
- Technical Support
- Account Management

The classification is performed only on masked data to ensure **privacy-preserving model training**.

◆ 3. Model Selection and Training Details

✓ *Data Preprocessing*

- All emails were cleaned and masked using the `mask_pii()` function.
- The masked emails were stored as a separate column and used for training.

✓ *Model Used*

- We selected a **Naive Bayes classifier** (`MultinomialNB`) for its simplicity and effectiveness on text classification tasks.
- A **TfidfVectorizer** was used to convert masked emails into numerical feature vectors.

✓ *Pipeline & Training*

- We built a `scikit-learn` pipeline that included TF-IDF transformation followed by the classifier.
- The dataset was split using an 80-20 train-test ratio.
- The model was trained using `fit()` and evaluated using `classification_report()`.

✓ *Performance*

The classifier achieved high precision and recall on most categories, demonstrating that masking does not degrade classification accuracy significantly.

◆ 4. Challenges Faced and Solutions Implemented

Challenge 1: PII Detection Without LLMs

- **Problem:** Detecting personal entities like names and card numbers without pre-trained LLMs is difficult.
- **Solution:** We created highly optimized **Regex patterns** for each PII type and fine-tuned them with real examples from the dataset.

Challenge 2: Model Accuracy with Masked Inputs

- **Problem:** Masking replaced contextual keywords, which could reduce classification accuracy.
- **Solution:** We trained the model entirely on masked data so that it learned from structure and intent, not PII context.

Challenge 3: Maintaining Output Format for API

- **Problem:** The required JSON response format had strict guidelines.
- **Solution:** We designed a helper function to wrap classification results and masked entity metadata in the exact format expected by the automated grading system.

Challenge 4: Deployment Readiness

- **Problem:** Integrating model prediction, masking logic, and API response in one flow.
- **Solution:** We structured the project with reusable modules (`utils.py`, `app.py`, etc.) and deployed it using **FastAPI** on **Hugging Face Spaces**.