Project Report – Email PII Masking and Classification API

- **Introduction to the Problem Statement:**

As organizations receive more volumes of support and service-related emails, it is now important that user data is processed securely and issues are classified effectively. Such emails usually carry Personally Identifiable Information (PII) like names, email addresses, phone numbers, and organizational details, which need to be obfuscated for privacy and data protection regulations compliance (e.g., GDPR, HIPAA). Moreover, directing the email content to the relevant service category such as Incident, Request, Change, or Problem assists in quicker response and resolution.

- **Approach taken for PII masking and classification:**

Masking:

The implemented methodology focuses on detecting and masking Personally Identifiable Information (PII) from a dataset of emails written in multiple languages, specifically English and German. The process begins by importing necessary libraries such as pandas for data handling, spacy for Named Entity Recognition (NER), re for regular expressions, and langdetect for language identification. The system first loads the appropriate spaCy language models (en_core_web_sm and de_core_news_sm) to perform NER on the input texts. Each email is analyzed using langdetect to determine its language, and then passed through the respective spaCy pipeline for entity extraction.

The code focuses on identifying and masking sensitive entities such as names (using spaCy's NER for labels like PERSON and PER) and other forms of PII using carefully crafted regular expressions. These include email addresses, phone numbers, dates of birth, Aadhar numbers, credit/debit card numbers, CVV codes, and card expiry dates. Each of these is replaced with a standard placeholder such as [NAME], [EMAIL], [PHONE], etc. This hybrid approach, combining rule-based regex and machine learning-based NER, ensures a higher accuracy in identifying a wide range of PII.

Once all PII is masked, the cleaned data is stored in a new CSV file, maintaining the original format but ensuring privacy. The multilingual handling ensures adaptability for global datasets, while the modularity allows for easy extension to other languages or entity types. This method strikes a balance between performance and accuracy, making it suitable for preprocessing textual data before applying further analysis like classification or storage in compliance with data privacy regulations.

Classification:

This methodology involves training a Random Forest model to classify masked email texts. The dataset contains emails with PII already masked, where the first column holds the email content and the second contains the labels. Text data is converted into numerical form using TfidfVectorizer, focusing on the top 5000 important words and removing common stop words. The data is split into training and testing sets (80-20), and a RandomForestClassifier is trained. Model performance is evaluated using precision, recall, and F1-score. The trained model is then saved using joblib for future use.

- **Model selection and training details:**

Model selection:

The following link contains a comparison of other models with respect to Random Forest. Based on classification report, machine learning model was selected.

https://colab.research.google.com/drive/1aWfWFfiaaF-PZTTJRhtrYUuZEc1m8Ip9?usp=sharing

Model Training Details

The Random Forest Classifier was selected due to its robustness and ability to handle high-dimensional text data effectively. The model was trained on a dataset of email texts with personally identifiable information (PII) masked, and each email categorized into predefined classes.

1. Data Preprocessing:
    - Texts were vectorized using TF-IDF (Term Frequency-Inverse Document Frequency) with a maximum of 5000 features.
    - English stop words were removed to reduce noise in the input features.
2. Train-Test Split:

   - The dataset was divided into 80% training and 20% testing using train_test_split, with a fixed random state (random_state=42) for reproducibility.

3. Model Parameters:
   - A RandomForestClassifier from scikit-learn was used.
   - n_estimators=100, meaning the model built 100 decision trees.
   - Default parameters were used for other hyperparameters.

4. Evaluation:
   - The model was evaluated using classification metrics like precision, recall, f1-score, and accuracy via classification_report.

5. Model Saving:
   - The trained model was serialized using joblib and saved as random_forest_model.pkl for future use.

● Challenges Faced and Solutions Implemented

One of the major challenges encountered was integrating the project as an API on Hugging Face Spaces. Initially, deploying the FastAPI application required configuring the correct structure, setting up requirements.txt, Dockerfile, and ensuring the /classify endpoint accepted the required input format and returned the expected output. There were several errors related to routing (404 Not Found) and incorrect HTTP request formatting during Postman testing. These were resolved by carefully aligning the FastAPI route with the expected POST request schema and ensuring that the input and output strictly followed the specified JSON structure. Additionally, network protocol errors and incorrect usage of model file paths were debugged through logs, and necessary environment variables and file paths were corrected. Once the deployment was successful, testing the endpoint via Postman validated the entire flow—from email input to PII masking, classification, and JSON output.