Email Classification System Report

Name: Rituraj Sharma Platform: Hugging Face Spaces

Endpoint: https://rituraj18-email2.hf.space/classify

1. Introduction

This assignment is about building an email classification system for a support team. I needed to create a service that:

- Masks personal information (PII) in incoming emails.
- Classifies the masked email into categories like Incident, Problem, Request, or Change.
- Returns both the original email text and the masked version, along with the classification result.

The system is implemented as a Flask API deployed on Hugging Face Spaces. Users send a POST request to /classify with their email text, and the API responds with the required data in JSON format.

2. Approach for PII Masking and Classification

I used a two-step method to mask PII:

1. **NER-based masking:**
    - I used a lightweight Hugging Face NER model (dslim/bert-base-NER) to find names, email addresses, and dates.
    - Detected entities like PER, EMAIL, and DATE are mapped to our labels full_name, email, and dob.
    - A placeholder like [full_name] replaces each found entity.
2. **Regex fallback:**
    - For PII types not always found by NER (e.g., phone numbers, Aadhar, credit/debit card numbers, CVV, expiry dates), I wrote regular expressions.
    - Each regex finds its pattern in the text and replaces it with a matching placeholder (e.g., [phone_number]).
    - I also fixed issues where phone numbers like +971-50-123-4567 were partially masked as CVV or DOB by placing the phone regex first and making it more flexible.

After masking, the text is passed to the classification model.

1. Model Selection and Training

I tried three different models for classification: **LinearSVC**, **Multinomial Naive Bayes**, and **DistilBERT**. After comparing their performance, I chose **DistilBERT** as it gave the highest validation accuracy.

2. **Model**: TFDistilBertForSequenceClassification fine-tuned on a support-email dataset.
3. **Reason for choosing this model**:
    - DistilBERT is a lighter and faster version of BERT, while still effectively capturing contextual language representations.

- It is well-suited for deployment on lightweight environments such as Hugging Face Spaces.
- The model generalizes well even on smaller datasets and significantly reduces training time compared to full BERT.

4. **Training data**: The dataset included emails labeled as one of the following categories: **Incident**, **Request**, **Change**, or **Problem**.
5. **Preprocessing**:
    - All emails were masked for Personally Identifiable Information (PII) before training, so the model learned from anonymized placeholders instead of real names or numbers.
6. **Training steps**:

    - Tokenized the email texts using DistilBertTokenizerFast with truncation and padding to a fixed length of 128 tokens.
    - Trained the model using TensorFlow, experimenting with hyperparameters until a satisfactory validation accuracy was reached.
    - Saved the final fine-tuned model and a LabelEncoder (via joblib) to map model output IDs back to category names.
    - The model was trained for 3 epochs:
    - Adam with a learning rate of 5e-5
    - Loss Function: Sparse categorical cross-entropy (logits-based)
    - Metrics: Accuracy

7. **Validation Accuracy Achieved**: **76.13%**

4. System Integration and API Deployment

I built a Flask app (app.py) with one endpoint:

- **Endpoint:** /classify (POST)
- **Input format: {"input_email_body": "<email text>"}**
- **Output format:**

```
{
  "input_email_body": "<original text>",
  "list_of_masked_entities": [
    {"position": [start, end], "classification": "type", "en
  ],
  "masked_email": "<masked text>",
  "category_of_the_email": "<class>"
}
```

Key steps in the code:

- Load model, tokenizer, and label encoder at startup.
- In mask_pii(), apply NER first, then regex patterns.
- In **predict_category(),** tokenize and run the DistilBERT model to get output logits.
- Serve requests in Flask and return the JSON response. I used **json.dumps** and **Response()** instead of **jsonify()** to ensure the output key order matches exactly.

I deployed the app on Hugging Face Spaces with no frontend UI. The API is tested using a simple Python script (request.py) that sends sample emails and prints responses.

5. Challenges and Solutions

- **False positives in NER:** Short words like Dr or fragments like Na were sometimes detected as names. To reduce this, I filtered out entities shorter than three characters.
- **Trailing punctuation in regex matches:** Emails like **john@site.com..** included extra dots. I trimmed trailing dots from the matched string before masking.
- **Phone number masking issues:** Some international phone numbers were partially masked. I fixed this by improving the regex and making sure phone number detection happens before CVV or DOB masking.
- **Maintaining JSON output order:** Flask's jsonify method sorted keys, so I used json.dumps(..., sort_keys=False) with Flask Response to control the output format.
- **Balance between speed and accuracy:** I chose **DistilBERT** for its smaller size, which made the API faster while still providing excellent classification accuracy.

6. Conclusion

This project demonstrates a simple but effective email classification pipeline. It masks PII using a mix of NER and regex, classifies masked text with a DistilBERT model, and serves results through a Flask API. All requirements are met, and the endpoint can be tested programmatically.

7. API Testing Script

Below is the request.py script I used to test the deployed API:

```python
import requests

url = "https://rituraj18-email2.hf.space/classify"
email_text = (
    "My name is Rituraj, and I would like to request a change in the account type associated with the email address ritu@akaike.com"
)
payload = {"input_email_body": email_text}

response = requests.post(url, json=payload)

print("Status Code:", response.status_code)

print("Raw JSON response:")
print(response.text)
```

This script confirms that the API returns the correct fields and classification for sample requests.