Email Classification and PII Masking System for Customer Support

# Overview

This project focuses on building a smart email handling system for a company’s support team. The goal is to automatically figure out what kind of help a customer needs — billing issue, technical support, or account help — and do it safely by first masking any personal info (like names, emails, and card details). Everything is wrapped up into a clean API that anyone on the team can use.

# Step 1: Detecting & Masking Personal Info (PII)

Before we touch the email content, we run it through a masking engine that hides sensitive info like:

- Full names

- Emails

- Phone numbers

- Aadhaar numbers

- Credit/Debit cards (plus CVV and expiry)

- Date of birth

We used:

- Regex to spot predictable patterns like email formats and card numbers.

- spaCy’s NER (Named Entity Recognition) to catch names and dates.

Example:

Input: “Hi, I’m John Doe. My email is john@example.com.”

Masked: “Hi, I’m [full\_name]. My email is [email].”

# Step 2: Classifying Emails by Topic

Once the sensitive stuff is masked, we move on to figuring out what the email is actually about. Is it a billing issue? Technical trouble? A request about their account?

To do this, we vectorized the email text using TF-IDF, then trained and compared two machine learning models:

- Naive Bayes (Multinomial) – Simple, fast, and works well for text.

- SVM (Support Vector Machine) – A bit heavier, but more accurate.

Both were trained on real support emails from the dataset. The SVM model gave better accuracy (~85%), so we used it in the final API.

# The Pipeline: How It All Fits Together

1. Input: Raw email text comes in through a POST request.

2. PII Masking: Sensitive info is detected, masked, and saved.

3. Classification: The cleaned email is passed to the SVM model.

4. Output: The system responds with the masked email, detected PII (with positions), and the predicted category.

Sample Output:

{

"input\_email\_body": "Hi, my name is John Doe...",

"list\_of\_masked\_entities": [

{

"position": [17, 25],

"classification": "full\_name",

"entity": "John Doe"

}

],

"masked\_email": "Hi, my name is [full\_name]...",

"category\_of\_the\_email": "Technical Support"

}

# Challenges We Faced (And Solved)

- PII Overlap: Sometimes regex and spaCy caught the same thing. We added checks to avoid duplicate masking.

- Maintaining Character Positions: As the text changes (via masking), it’s easy to lose track of where things were. We carefully handled replacements to track positions.

- Data Cleanup: The raw dataset had some missing or inconsistent labels, so we cleaned that up before training.

# What's Next?

You’re almost ready to ship. Here’s what’s left:

GitHub Setup:

- app.py – Main FastAPI entrypoint

- models.py – ML logic

- utils.py – Masking utilities

- api.py – Pydantic schemas

- requirements.txt – Dependencies

- README.md – Setup and usage

Hugging Face Spaces:

- Push the repo to GitHub

- Deploy it using the FastAPI template

- Share the public link