Email Classifier Project Report

Introduction

In modern enterprise environments, service teams often receive large volumes of internal support emails. These emails typically relate to various categories such as incidents, problems, change requests, or general reasons behind operational anomalies. Manually categorizing such emails is time-consuming and error-prone, especially when timely triage is crucial.

This project addresses that challenge by building an intelligent system that can automatically classify incoming emails into one of four operational categories: incident, problem, change, or reason. The ultimate goal is to streamline internal support processes, reduce resolution time, and ensure appropriate teams handle the right type of requests efficiently.

Approach

PII Masking

Given the sensitive nature of internal communication, we incorporated a robust PII (Personally Identifiable Information) masking module as a pre-processing step. This step ensures privacy by identifying and replacing entities such as names, phone numbers, email addresses, dates, and ticket IDs with generic tokens (e.g., , ,).

We used a custom function called apply_masking that leverages regular expressions and optionally, spaCy's Named Entity Recognition (NER), to detect and anonymize sensitive data. This protects user privacy while preserving the semantic structure of the text for model training.

Text Preprocessing

Post-masking, we cleaned the text using a text_preprocessing function. This involved:

- Lowercasing
- Removing stopwords, special characters, and URLs
- Optional stemming/lemmatization for linguistic normalization

The processed text was then passed through a SentenceTransformer model ("all-MiniLM-L6-v2") to generate dense semantic embeddings for each email.

Model Selection and Training

Initially, we experimented with classical ML models like Logistic Regression, which provided a solid baseline. However, to better capture contextual dependencies in sentence structure, we transitioned to deep learning models — specifically, LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) networks.

Final Architecture

The final model architecture includes:

- Input: (1, 384)-dimensional sentence embeddings
- Reshaped to (1, 1, 384) for sequential modeling
- A Bidirectional GRU layer with dropout for generalization
- Dense softmax output with 4 units corresponding to the 4 classes

The model was compiled with:

- Loss: Sparse Categorical Crossentropy
- Optimizer: Adam
- Metrics: Accuracy

Label Encoding

We used scikit-learn's LabelEncoder to transform the categorical type labels into numeric form (e.g., incident → 0). During inference, we apply the inverse_transform method to convert predictions back to the original category names. To ensure compatibility with Hugging Face's restrictions, we replaced the pickle-based label encoder with a JSON-serialized list of class labels.

Prediction Pipeline

The prediction flow for a single email involves:

1. Preprocessing and masking
2. Embedding using a SentenceTransformer
3. Reshaping input for the GRU model
4. Predicting the class probabilities
5. Converting the argmax class back to the type label

This was implemented in a clean and modular function: predict_single_email().

Challenges and Solutions

1. **PII Identification**
   Challenge: PII formats vary across organizations.
   Solution: Designed modular regex-based patterns and allowed easy expansion with additional patterns. Used optional NER tagging for robustness.

2. **Class Imbalance**
   Challenge: "Incident" and "Problem" emails were overrepresented.
   Solution: Used stratified sampling during train-test split and monitored class-wise metrics during evaluation.

3. **Embedding Shape Compatibility**
   Challenge: Sentence embeddings needed reshaping for GRU input.
   Solution: Reshaped input to (batch, time_steps, features) to align with GRU expectations.

4. **Model Interpretation**
   Challenge: Neural models are often black boxes.
   Solution: Stored confidence scores and provided clear mappings from numeric predictions to label names using LabelEncoder.

5. **Output Format**
   Challenge: TensorFlow outputs class indices; inverse transformation sometimes failed due to array type mismatch.
   Solution: Explicitly reshaped and cast predictions to NumPy arrays before using inverse_transform.

6. **Environment Compatibility for Deployment (Current Issue)**
   Challenge: The model was trained in a newer environment on Google Colab (TensorFlow 2.13+, SentenceTransformer 2.2.2), but the Hugging Face deployment runtime uses older versions by default. This leads to runtime errors due to version mismatches.
   Solution: We replaced all pickle-based objects with JSON (for tokenizer/label mapping) and defined a requirements.txt file specifying exact dependency versions. Despite this, the Hugging Face Space currently fails to run due to unresolved runtime compatibility issues. We are actively working on aligning the deployment environment with our local training setup by customizing dependencies and investigating Hugging Face's runtime logs.

Conclusion

The Email Classifier project brings together advanced natural language processing (NLP) methods, deep learning, and strong privacy protection to create a smart, automated system for classifying internal support emails. By combining these modern technologies, the system can understand the meaning behind each message and accurately sort it into one of several categories—such as incident, problem, change request, or operational reason.

A key feature of this solution is its built-in protection for sensitive data. The classifier automatically masks personally identifiable information (PII) like names, email addresses, phone numbers, and ticket IDs before processing any email content. This ensures the system remains privacy-compliant while still learning from the important parts of the message.

The system is also designed to be modular and flexible. This means it can easily adapt to new categories or types of emails in the future without needing to be rebuilt from scratch. It can scale across teams and departments, and it's built in a way that supports continuous improvement with new data.

In real-world use, this tool can dramatically reduce the amount of time spent manually reviewing and sorting support emails. By automatically routing each message to the right team, it speeds up response times, minimizes human error, and helps support teams work more efficiently—while ensuring that sensitive data stays protected throughout the process.