Akaike Email Classification Report

Author: Dhanush
Date: April 19, 2025
Project: Akaike Internship Assignment

1. Objective

The objective of this project was to develop a machine learning system to classify support emails into four categories—Incident, Request, Problem, and Change—while masking Personally Identifiable Information (PII) such as names, email addresses, phone numbers, and other sensitive data. The system aims to automate email processing for customer support teams, ensuring privacy compliance and efficient categorization.

2. Approach

The project was implemented using Python and deployed as a FastAPI application on Hugging Face Spaces with Docker. The approach involved the following steps:

- **PII Masking**: Utilized regular expressions (regex) to mask structured PII (e.g., emails, phone numbers, Aadhar numbers, credit card details) and Spacy's natural language processing (NLP) to identify and mask entities like names and dates. The mask_pii.py script combines these techniques for robust PII anonymization.
- **Data Preprocessing**: Loaded emails.csv with email and type columns. Used TfidfVectorizer to convert email text into numerical features, limiting to 5,000 features to manage computational complexity.
- **Model Training**: Trained a Random Forest Classifier (n_estimators=50) on a sampled dataset of 1,000 emails to handle class imbalance and reduce training time. The model and vectorizer were saved as rf_model.pkl and vectorizer.pkl using joblib.
- **Pipeline Integration**: Developed pipeline.py to integrate PII masking and classification, processing raw emails and outputting masked text and predicted categories.
- **API Deployment**: Built a FastAPI server (app.py) exposing a /classify endpoint to accept email inputs and return JSON responses with masked emails and categories. Deployed using Docker on Hugging Face Spaces.

3. Results

The Random Forest model achieved an accuracy of 0.74 on the test set, with the following class distribution in the sampled dataset:

- Incident: 403 emails
- Request: 297 emails
- Problem: 183 emails
- Change: 117 emails

The system successfully masked PII (e.g., replacing "john.doe@example.com" with "[email]") and classified emails, as demonstrated by local and Space API tests (e.g., classifying a test email as "Incident"). The FastAPI endpoint processed requests in under 1 second, suitable for real-time applications.

4. Challenges

Several challenges were encountered during development:

- **Git Errors**: Initial pathspec errors when committing rf_model.pkl were resolved by ensuring proper .gitignore and Git configuration.
- **Large CSV Files**: The emails.csv file was large, requiring sampling to 1,000 rows to manage memory and training time.
- **SDK Mismatch**: The Hugging Face Space initially used fastapi SDK, causing deployment issues. Switching to docker with sdk_version: 0.17.0 resolved this.
- **Column Mismatches**: Errors like KeyError: 'email_masked' and email_content were fixed by standardizing on the email column in models.py and pipeline.py.
- **Vectorization Errors**: pipeline.py failed to vectorize masked emails, causing a ValueError. Adding vectorizer.transform fixed this.
- **Streamlit in app.py**: An incorrect Streamlit implementation was replaced with FastAPI to match the project's API requirements.
- **scikit-learn Version Mismatch**: rf_model.pkl was pickled with scikit-learn 1.3.0, but the Space used 1.5.2, triggering warnings. Retraining with 1.5.2 resolved this.
- **Space 404 Error**: Persistent 404 errors on the /classify endpoint were addressed by rebuilding the Space and verifying configuration.

5. Outcome

The project successfully delivered a functional email classification system deployed on Hugging Face Spaces at https://huggingface.co/spaces/Parzivalbsdk/akaike-email-classification. The system masks PII and classifies emails with 0.74 accuracy, meeting the internship requirements.

The GitHub repository (https://github.com/dhanush14chowdary/akaike_email_classification) contains all source code, models, and documentation.

Future improvements include:

- Using BERT or other transformer models to improve classification accuracy.
- Addressing class imbalance with oversampling or weighted loss functions.
- Enhancing PII masking with more robust NLP models.
- Adding a front-end interface for user interaction.

6. Conclusion

This project demonstrated proficiency in machine learning, NLP, API development, and cloud deployment. Despite challenges, the system was completed and deployed within the deadline, showcasing problem-solving skills and technical expertise.

Word Count: ~350 (to be expanded to ~600–700 for 2 pages in Word)