**Email Classification API with PII/PCI Masking**

**Report**

**1. Introduction**

**Problem Statement:**  
This project addresses the challenge of automatically classifying emails into distinct categories while ensuring sensitive information (PII/PCI) is appropriately masked before processing. With the increasing volume of emails in business environments, manual classification is time-consuming and prone to human error. Additionally, handling emails often involves processing sensitive information that requires protection under various privacy regulations.

**Objectives:**

* Develop a robust model for classifying emails into four categories: Incident, Request, Problem, and Change
* Implement comprehensive PII/PCI masking to protect sensitive information
* Create a lightweight, accessible API for integration with existing systems
* Containerize the solution for easy deployment in various environments

**Business Use Case and Relevance:**  
Email classification systems are critical for organizations with high email volumes, particularly in IT service management, customer support, and technical assistance domains. Automatic classification helps route communications to appropriate teams, prioritize urgent matters, and maintain service level agreements. The addition of PII/PCI masking capabilities ensures compliance with privacy regulations such as GDPR, CCPA, and PCI-DSS while still enabling effective classification.

**2. Methodology**

**Data Collection and Preprocessing:**  
The project utilized a dataset of 24,000 emails distributed across four categories:

* Incident: 9,586 samples (40%)
* Request: 6,860 samples (29%)
* Problem: 5,037 samples (21%)
* Change: 2,517 samples (10%)

The preprocessing pipeline included:

1. Text normalization (lowercase conversion, punctuation handling)
2. PII/PCI data identification and masking
3. Removal of non-informative elements (email headers, signatures)
4. Tokenization and basic text cleaning

**PII/PCI Masking Techniques:**  
To ensure data privacy, the system implements comprehensive masking for sensitive information including:

* Full names
* Email addresses
* Phone numbers
* Date of birth information
* Aadhar card numbers
* Credit/debit card numbers
* CVV numbers
* Expiry dates

For each identified sensitive element, the system replaces the actual value with a generic placeholder that maintains the semantic category while removing the identifying information.

**Feature Extraction:**  
Text data was converted to numerical features using the Term Frequency-Inverse Document Frequency (TF-IDF) vectorization technique. This approach:

* Captures the importance of words in the corpus
* Reduces the impact of commonly occurring but less informative words
* Creates a sparse matrix representation suitable for machine learning algorithms

**Model Selection Approach:**  
After initial exploration of more complex models (BERT, RoBERTa), computational constraints led to adopting a Support Vector Machine (SVM) classifier. This decision balanced computational efficiency with performance requirements. The model selection process included:

1. Comparative analysis of baseline classifiers
2. Cross-validation to evaluate generalization performance
3. Grid search for hyperparameter optimization
4. Performance evaluation across multiple metrics

**3. Model Development**

**Description of the SVM Classifier:**  
A Support Vector Machine classifier was selected due to its effectiveness for text classification tasks, particularly when working with high-dimensional data produced by TF-IDF vectorization. SVMs are well-suited for this application because:

* They perform well on sparse feature spaces typical in text classification
* They handle the "curse of dimensionality" effectively
* They deliver good performance with limited training data
* They're computationally efficient compared to deep learning alternatives

**Hyperparameter Tuning Process:**  
Grid search cross-validation was used to optimize model parameters. The following hyperparameters were explored:

* Regularization parameter (C): [0.1, 1.0, 10.0]
* Max features for TF-IDF: [5000, 10000, 15000]
* N-gram range: [(1,1), (1,2)]

The best parameters identified through grid search were:

* C: 1.0
* max\_features: 10000
* ngram\_range: (1, 1)

The best cross-validation score achieved was 0.7587.

**Training Process:**  
The model was trained on the dataset of 24,000 emails with a standard train-test split. The training process involved:

1. Applying the optimized TF-IDF vectorization to the training data
2. Fitting the SVM classifier with the optimized hyperparameters
3. Regular evaluation on a validation set to monitor convergence
4. Model serialization for deployment

**4. Performance Analysis**

**Detailed Model Performance:**  
The final model achieved an overall accuracy of 0.77 on the test set, with the following per-class metrics:

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| Change | 0.93 | 0.85 | 0.89 | 504 |
| Incident | 0.72 | 0.82 | 0.76 | 1917 |
| Problem | 0.56 | 0.42 | 0.48 | 1007 |
| Request | 0.92 | 0.94 | 0.93 | 1372 |

**Class Imbalance Discussion:**  
The "Problem" category shows noticeably lower performance across all metrics (precision: 0.56, recall: 0.42, F1-score: 0.48). This underperformance can be attributed to:

1. Class imbalance in the training data (Problem category represents only 21% of samples)
2. Potential semantic overlap between "Problem" and "Incident" categories
3. Greater linguistic variability in how problems are described compared to other categories

The "Change" category, despite having the smallest number of samples, performs remarkably well (F1-score: 0.89), likely due to distinctive vocabulary and expressions used in change requests that make them easily distinguishable.

**Suggestions for Improvement:**  
To address the performance disparities, especially for the "Problem" category:

1. Implement class weighting or balanced sampling techniques to address class imbalance
2. Explore feature engineering specific to the underperforming class
3. Consider ensemble methods that might better capture the nuances between similar categories
4. Increase the representation of the "Problem" category in the training dataset
5. Investigate additional contextual features beyond simple word presence

**5. API Implementation**

**Architecture Overview:**  
The API implementation follows a modular design with three main components:

1. **Preprocessing Module**: Handles text cleaning and PII/PCI masking
2. **Classification Module**: Manages model loading, feature extraction, and prediction
3. **API Layer**: Exposes endpoints and handles request/response formatting

The architecture ensures separation of concerns and enables independent testing and improvement of each component.

**Endpoints Description:**  
The API offers two primary endpoints:

1. **Classification Endpoint** (POST /classify):
   * Accepts raw email text
   * Returns the predicted category, confidence score, and masked version of the text
   * Example request:
   * { "email\_text": "Your email content here..."}
   * Example response:
   * { "category": "Incident", "confidence": 0.85, "masked\_text": "Your email with masked PII..."}
2. **PII Masking Endpoint** (POST /mask):
   * Accepts any text input
   * Returns the text with sensitive information masked
   * Example request:
   * { "text": "Text with PII to mask"}
   * Example response:
   * { "masked\_text": "Text with masked PII"}

**Dockerization Process:**  
The application was containerized using Docker to ensure consistent behavior across different environments:

1. A lightweight Python base image was selected (python:3.9-slim)
2. Required dependencies were installed through requirements.txt
3. Model artifacts and code were copied into the container
4. The FastAPI application was configured to run using Uvicorn
5. The container was optimized for size and startup speed
6. The image was published to Docker Hub (justtmansss/email-classification-api)

**Deployment on Hugging Face Spaces:**  
The API was successfully deployed on Hugging Face Spaces, providing:

* Public accessibility for demonstration purposes
* Built-in documentation through FastAPI's Swagger UI
* Scalable infrastructure for handling requests
* Easy sharing and collaboration capabilities

The deployed API is available at: https://huggingface.co/spaces/justtmansss/email-classification-api

**6. Future Work**

**Potential Improvements to Model Accuracy:**  
Several avenues for improving the model's performance have been identified:

1. **Advanced NLP Techniques**: Revisit the initial plan to use transformer-based models like BERT or RoBERTa with adequate computational resources
2. **Domain Adaptation**: Fine-tune the model on domain-specific email datasets
3. **Ensemble Methods**: Combine multiple classifiers to improve robustness and accuracy
4. **Active Learning**: Implement a feedback loop to continuously improve on misclassified instances

**Additional Features:**  
Future development could incorporate:

1. **Sentiment Analysis**: Add sentiment detection to gauge urgency or customer satisfaction
2. **Priority Scoring**: Automated urgency assessment based on content analysis
3. **Response Generation**: Suggest template responses based on classification
4. **Multi-language Support**: Expand beyond English to support global operations
5. **Advanced Analytics**: Provide insights on email volume, response times, and classification trends

**Scaling Considerations:**  
For enterprise-level deployment, the following scaling strategies should be considered:

1. **Horizontal Scaling**: Deploy multiple instances behind a load balancer
2. **Asynchronous Processing**: Implement queue-based processing for high-volume scenarios
3. **Model Quantization**: Reduce model size and inference time through quantization techniques
4. **Caching Strategies**: Implement intelligent caching for improved response times
5. **Database Integration**: Move from file-based to database storage for scalable persistence

**7. Conclusion**

**Summary of Achievements:**  
This project successfully delivered:

1. A functional email classification system with 77% overall accuracy
2. Robust PII/PCI masking capabilities for privacy protection
3. A containerized API solution ready for deployment
4. A public demonstration on Hugging Face Spaces
5. A foundation for future improvements and extensions

The system effectively balances performance, privacy, and accessibility, providing a practical solution for email classification needs.

**Lessons Learned:**  
Several valuable insights emerged during this project:

1. **Start Simple, Then Iterate**: The initial ambitious approach using transformer models proved computationally challenging. Pivoting to a simpler SVM-based approach allowed for faster development and established a working baseline that can be enhanced incrementally.
2. **Data Quality Matters**: The class imbalance significantly impacted model performance, highlighting the importance of balanced training data or appropriate compensation techniques.
3. **Privacy by Design**: Incorporating PII/PCI masking early in the pipeline ensured privacy considerations were embedded throughout the system rather than added as an afterthought.
4. **Practical Deployment Considerations**: Containerization and API development revealed the importance of considering the entire solution lifecycle, not just model performance.

The project demonstrates how machine learning can be effectively applied to real-world business problems while maintaining privacy standards and providing accessible interfaces for integration.