YUM Email Categorizer - Complete User Guide

Table of Contents

Overview

YUM Email Categorizer is an intelligent email classification system that automatically categorizes emails into predefined categories. The system provides both a REST API backend and command-line interface for email processing, with built-in feedback mechanisms for continuous improvement.

Key Features

- **Automatic Email Classification**: Categorizes emails based on subject and body content
- **REST API**: Easy integration with existing systems via HTTP endpoints
- **CLI Interface**: Interactive command-line tools for testing and management
- **Feedback System**: User correction mechanism to improve classification accuracy
- **Realistic Email Generation**: Built-in tools for testing with synthetic data
- **Category Management**: Dynamic addition of new email categories
- **Secure Feedback Storage**: Email content is hashed for privacy protection

Current Categories

The system comes with these predefined categories: - **Personal**: Personal communications and private matters - **Work**: Business-related emails and professional communications - **Social**: Social media notifications and community updates - **Promotions**: Marketing emails, advertisements, and promotional content - **Updates**: System notifications, newsletters, and informational updates

---

System Architecture

Project Structure

```
yum/
├── README.md                    # Project documentation
├── generate_email.sh            # Random email generation script
├── generate_realistic_emails.sh # Realistic email generation script
├── backend/                     # Core backend components
│   ├── app.py                  # Flask REST API server
│   ├── categoriser.py          # Email classification logic
│   └── data/
│       ├── categories.json     # Available email categories
│       └── feedback.json       # User feedback storage
└── ui/
    └── cli_test.py             # Command-line testing interface
```

Data Flow Architecture

```
Email Input → API/CLI → Categoriser → Classification → Response
     ↓
User Feedback → Feedback Storage → Future Improvements
```

Technology Stack

- **Backend**: Python Flask framework
- **Data Storage**: JSON files for categories and feedback
- **API**: RESTful HTTP endpoints
- **Security**: SHA-256 hashing for email content
- **Testing**: Bash scripts and Python CLI tools

Installation & Setup

Prerequisites

- **Python 3.7+**: Required for running the Flask application
- **pip**: Python package manager for installing dependencies
- **curl**: Command-line tool for testing API endpoints
- **jq**: JSON processor for parsing API responses (optional but recommended)

Step 1: Environment Setup

```
# Navigate to project directory
cd /path/to/yum
```

Step 2: Install Python Dependencies

```
# Install required packages
pip install flask requests

# Verify installation
python -c "import flask, requests; print('Dependencies installed successfully')"
```

Step 3: Make Scripts Executable

```
chmod +x generate_email.sh
chmod +x generate_realistic_emails.sh
```

Step 4: Test Basic Functionality

```
# Test the categorizer module directly
cd backend
python -c "
import categoriser
categories = categoriser.get_categories()
print('Available categories:', categories)
result = categoriser.classify_email('Test subject', 'Test body')
print('Classification test:', result)
"
```

Core Components

1. Flask API Server (`backend/app.py`)

The main REST API server providing HTTP endpoints for email classification and management.

Key Features:

- **Request Logging**: All API requests are logged for debugging
- **Data Validation**: Input validation for required fields
- **Error Handling**: Comprehensive error responses with appropriate HTTP status codes
- **JSON Communication**: All requests and responses use JSON format

2. Email Categoriser (`backend/categoriser.py`)

The core classification engine responsible for analyzing email content and assigning categories.

Current Implementation:

- **Rule-Based Classification**: Simple keyword-based classification
- **Random Fallback**: Uses random selection when no specific rules match
- **Confidence Scoring**: Provides confidence levels for classifications

- **Feedback Storage**: Securely stores user corrections using SHA-256 hashing

Classification Logic:

1. **Keyword Analysis**: Searches for specific keywords in subject and body
2. **Priority Rules**: "Urgent" and "important" keywords trigger high-confidence classification
3. **Fallback**: Random category selection when no rules match
4. **Confidence Calculation**: Returns confidence score (0.0 to 1.0)

3. CLI Testing Interface (`ui/cli_test.py`)

Interactive command-line tool for testing and managing the email categorization system.

Usage Instructions

Starting the System

1. Start the Flask API Server

```
# Navigate to project root
cd /path/to/yum

# Start the server
python backend/app.py
```

The server will start on `http://127.0.0.1:5000`

2. Verify Server is Running

```
# Test server health (in a new terminal)
curl http://127.0.0.1:5000/categories
# Should return: ["Personal","Work","Social","Promotions","Updates"]
```

Interactive Testing with CLI

Start the CLI Interface

```
# In a new terminal (keep server running)
python ui/cli_test.py
```

Automated Email Generation and Testing

1. Generate Random Emails

```
# Generate one random email
./generate_email.sh
```

2. Generate Realistic Emails

```
# Generate 5 realistic emails (default)
./generate_realistic_emails.sh

# Generate 10 realistic emails
./generate_realistic_emails.sh 10
```

API Reference

Base URL

```
http://127.0.0.1:5000
```

Endpoints

1. Classify Email

POST `/classify`

Classifies an email into a category based on subject and body content.

Request Body:

```
{
  "subject": "Meeting tomorrow",
  "body": "Hi team, let's meet tomorrow at 2 PM to discuss the project"
}
```

Response:

```
{
  "category": "Work",
  "confidence": 0.75
}
```

2. Submit Feedback

POST `/feedback`

Submits user feedback to correct classification errors.

Request Body:

```
{
```

```
  "email": {
    "subject": "Meeting tomorrow",
    "body": "Hi team, let's meet tomorrow at 2 PM"
  },
  "correct_category": "Work"
}
```

Response:

```
{
  "status": "success"
}
```

3. Get Categories

GET `/categories`

Retrieves all available email categories.

Response:

```
["Personal", "Work", "Social", "Promotions", "Updates"]
```

4. Add Category

POST `/category`

Adds a new category to the system.

Request Body:

```
{
  "name": "Finance"
}
```

Response:

```
{
  "status": "success"
}
```

Configuration

Categories Configuration (`backend/data/categories.json`)

The system's categories are stored in a JSON array format:

```
[
  "Personal",
  "Work",
  "Social",
  "Promotions",
  "Updates"
]
```

Feedback Storage (`backend/data/feedback.json`)

User feedback is stored with hashed email content for privacy:

```
[
  {
    "email_hash": "sha256_hash_of_email_content",
    "correct_category": "Work"
  }
]
```

Testing & Development

Manual Testing with curl

Test Classification

```
curl -X POST -H "Content-Type: application/json" \
  -d '{"subject":"Meeting request","body":"Can we schedule a meeting?"}' \
  http://127.0.0.1:5000/classify
```

Test Feedback Submission

```
curl -X POST -H "Content-Type: application/json" \
  -d '{"email":{"subject":"Test","body":"Test"},"correct_category":"Work"}' \
  http://127.0.0.1:5000/feedback
```

Test Category Addition

```
curl -X POST -H "Content-Type: application/json" \
  -d '{"name":"Finance"}' \
  http://127.0.0.1:5000/category
```

Automated Testing Scripts

The project includes two bash scripts for automated testing:

1. `generate_email.sh`: Generates random email content for stress testing
2. `generate_realistic_emails.sh`: Creates realistic email scenarios for functional testing

---

Troubleshooting

Common Issues

1. Server Won't Start

Problem: `python backend/app.py` fails to start

Solutions: - Check Python version: `python --version` (requires 3.7+) - Install Flask: `pip install flask` - Check port availability: `netstat -an | grep 5000`

2. API Connection Errors

Problem: CLI or scripts can't connect to API

Solutions: - Verify server is running: `curl http://127.0.0.1:5000/categories` - Check firewall settings - Ensure correct URL in scripts and CLI

3. Permission Denied on Scripts

Problem: `./generate_email.sh` shows permission denied

Solution:

```
chmod +x generate_email.sh
chmod +x generate_realistic_emails.sh
```

4. JSON Parsing Errors

Problem: Invalid JSON responses or parsing failures

Solutions: - Install jq: `sudo apt-get install jq` (Linux) or `brew install jq` (Mac) - Check API response format - Verify request JSON syntax

Debug Mode

To enable debug mode for more detailed error information:

```
# In backend/app.py, change:
app.run(debug=False)
# To:
app.run(debug=True)
```

Advanced Features

1. Custom Classification Rules

The current implementation uses simple keyword matching. To add custom rules, modify the `classify_email` function in `backend/categoriser.py`:

```
def classify_email(subject, body):
    text = (subject + " " + body).lower()

    # Custom rules
    if any(word in text for word in ["meeting", "schedule", "appointment"]):
        return "Work", 0.9
    elif any(word in text for word in ["sale", "discount", "offer"]):
        return "Promotions", 0.8
    # ... add more rules
```

2. Feedback Analysis

To analyze stored feedback for improving the system:

```
import json
import hashlib
from collections import Counter

def analyze_feedback():
    with open('backend/data/feedback.json', 'r') as f:
        feedback = json.load(f)

    categories = [item['correct_category'] for item in feedback]
    return Counter(categories)
```

3. Batch Processing

For processing multiple emails programmatically:

```
import requests

def batch_classify(emails):
    results = []
    for email in emails:
        response = requests.post(
            'http://127.0.0.1:5000/classify',
            json=email
        )
        results.append(response.json())
    return results
```

4. Integration Examples

Web Application Integration

```
// JavaScript example for web integration
async function classifyEmail(subject, body) {
    const response = await fetch('http://127.0.0.1:5000/classify', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ subject, body })
    });
    return await response.json();
}
```

Python Application Integration

```
import requests

class EmailClassifier:
    def __init__(self, api_url="http://127.0.0.1:5000"):
        self.api_url = api_url

    def classify(self, subject, body):
        response = requests.post(
            f"{self.api_url}/classify",
            json={"subject": subject, "body": body}
        )
        return response.json()

    def submit_feedback(self, email, correct_category):
        response = requests.post(
            f"{self.api_url}/feedback",
            json={"email": email, "correct_category": correct_category}
        )
        return response.json()
```

Future Enhancements

Planned Features

1. **Machine Learning Integration**: Replace rule-based classification with ML models
2. **Database Storage**: Move from JSON files to proper database
3. **User Authentication**: Add user management and authentication
4. **Web Interface**: Create a web-based UI for easier interaction
5. **Email Integration**: Direct integration with email providers
6. **Analytics Dashboard**: Visualization of classification statistics
7. **Export/Import**: Backup and restore functionality for categories and feedback

Contributing

To contribute to the project: 1. Fork the repository 2. Create a feature branch 3. Make your changes 4. Add tests for new functionality 5. Submit a pull request

---

Conclusion

The YUM Email Categorizer provides a solid foundation for email classification with room for enhancement and customization. The modular architecture allows for easy extension and integration into existing systems.

For support or questions, refer to the troubleshooting section or check the project documentation.

Last updated: December 2024