USER GUIDE

Email Classifier System

David Adediji, Keith O’Halloran, Rahul Alam, Noel Fontugne, Didier Bosiwa

Table of Contents

Introduction

Overview

Our Email Classifier System is for our client, which is a multinational IT services company. The aim of the system is to automatically categorize emails for this company as this company receives an enormous amount of customer emails every day. The system addresses this problem of the high volume of customer emails which in turn will help reduce operational costs and improve response times, which will improve customer experience.

The main goal for our system is to classify emails into hierarchical categories based on the content of the email. For example, an email can be classified into broader categories like a “Compliant” and be narrowed down to more specific ones such as “Cloud Autoscaling.” With this automation implemented it addresses the problems created by the substantial number of customer emails received daily

Some of the features of our system are:

1. **Dynamic Model Selection:**

- Our system supports multiple machine learning algorithms such as Random Forest, SVM, AdaBoost, Naive Bayes and Logistic Regression
- Our system allows for seamless switching between models using a dynamic selection interface.

2. **Automated Classification:**

- Our system processes and categorizes emails. The system also does this with minimal intervention which would save time and reduce errors.

3. **Advanced Preprocessing and Vectorization:**

- Our system includes a number of preprocessing steps such as text cleaning, stopword removal, stemming, and TF-IDF vectorization which prepares the email data for classification.

4. **Interactive Workflow:**

- With our system we allow users to load datasets, select classifiers, tune hyperparameter and evaluate models which are all done in an interactive command line interface.

5. **Performance Evaluation:**

- Our system provides detailed metric such as accuracy, confusion matrix and ROC-AUC

6. **Observer Notifications:**

- Our system notifies observers with real-time updates on classification performance and results.

Target Audience

The main target audience for our system is as follows:

1. **Developer**

- **Purpose:** To understand the system architecture to make modifications or enhancements
- Key Focus Areas:
- Debugging and maintaining the system
- Extending functionality
- Examples:
- Software engineers develop tools for automated email handling.
- Backend developers working on integration with existing systems.

2. Business Analysts:

- **Purpose:** To analyze and extract useful information from the classification results
- Key Focus Areas:
- Understanding the categorization of the customer emails to prioritize responses
- Leveraging classification trends to improve customer service strategies
- Examples:
- Managers are seeking to streamline email handling processes.
- Analysts generating reports for operational efficiency.

3. **Customer Service Team:**

- **Purpose:** To use classification outputs to improve response times and quality
- Key Focus Areas:
- Navigating the user interface to load datasets and classify emails.
- Utilizing classification insights to prioritize customer queries
- Interpreting performance metrics to adjust workflows effectively.
- Examples:
- The customer support agent is overseeing the email complaints and queries.
- The team lead assigning tasks based on the classification levels.

4. **Data Scientists**

- **Purpose:** To optimize the machine learning models and analyze performance metrics
- Key Focus Areas:
- Fine tuning some classifiers using hyperparameter optimization
- Evaluating results with metrics like confusion matrices
- Experimenting with new algorithms to improve classification performance
- Examples:
- Data analysts exploring trends in classification results.
- Machine learning specialists researching improvements in automated text classification.

Assumptions:

Our Email Classification System assumes the following about the user's environment:

1. **Software and Hardware Requirements**

- Operating System:
- The system is not platform dependent so it can run on Windows, macOS or Linux.
- Python Version:
- The system requires Python 3.8 or later.
- Hardware:
- It is recommended that the user has a modern computer with at least 8GB of RAM and a multi-core processor.

2. **Required Libraries and Dependencies:**

The following Python libraries are needed:

- Core Libraries:
- pandas: For dataset loading and manipulation.
- numpy: For numerical operations
- scikit-learn: For machine learning models and utilities.
- nltk: For natural language preprocessing
- Additional Libraries:
- matplotlib or seaborn: For visualizing performance metrics
- Logging: For system event logging

3. **Data Requirements**

- Dataset Format:
- The input dataset must be in CSV format with clearly labelled columns.
- Directory Structure:
- A data/ directory is assumed to exist in the root project folder to store the datasets.

Getting Started

System Requirements: Hardware

Minimum Specifications

These specifications allow for the system to run basic operations.

- Processor: Dual-core CPU
- RAM: 4 GB
- Storage: 10 GB of available disk space
- Graphics: No dedicated GPU required
- Operating System:
- Windows 10 (64-bit), macOS 10.13 (High Sierra), or Linux (Ubuntu 18.04 or equivalent)

Recommended Specifications

- Processor: Quad-core CPU
- RAM: 8 GB or higher
- Storage: 20 GB of available disk space for datasets, temporary files, and logs
- Graphics: No dedicated GPU required
- Operating System:
- Windows 11 (64-bit), macOS 11 (Big Sur) or later, or Linux (Ubuntu 20.04 or equivalent)

System Requirements: Software

1. **Python Version**

- Required Python 3.8 or later.

2. **Required Python Libraries**

- Core Libraries:
- pandas: For dataset loading and manipulation.

- numpy: For numerical operations
- scikit-learn: For machine learning models and utilities.
- nltk: For natural language preprocessing
- Utilities:
- Logging: For system event logging
- Re (Regular Expressions): For cleaning and preprocessing text data.
- Matplotlib or seaborn: For visualizing results

3. **Operating System**

- Windows: Windows 10 or later (64-bit)
- MacOS: macOS 10.13 (High Sierra) or later
- Linux: Ubuntu 18.04 or equivalent

4. **Optional Tools**

- IDE: PyCharm, Visual Studio Code or Jupyter Notebook for development
- Version Control: Git for managing the codebase.

Installation Instructions

1. **Prerequisites**

a. Python Installation:

- Ensure Python 3.8 or layer is installed on your system.
- Verify installation by running python –version.

b. Package Manager

- Install pip (Python package installer) if not already available:
- Python –m ensurepip -–upgrade

2. **Clone the Repository**

a. Use git to clone the project repository (if available):

git clone <repository_url>

cd <project_directory>

3. **Install Required Python Libraries**

- pip install pandas numpy scikit-learn nltk.
- pip install matplotlib seaborn.
- python –m nltk. downloader stopwords

4. **Prepare Dataset**

a. Place the dataset files in the data/ directory under the project root.

b. Ensure that all the datasets are in CSV format with proper column names.

5. **Run the System**

a. Navigate the project directory.

- cd <project_directory>

b. Start the classification system by running the main script.

- Python main.py

How to Run the Email Classifier System

1. **Navigate to the Project Directory**

- cd <project_directory>

2. **Run the System**

- Python main.py

3. **Interactive Workflow**

Once the program starts, follow the on-screen instructions to perform the following steps:

1. **Select a Dataset:**

- Choose from the available datasets:
- Press [1] for the "Gallery App Dataset."
- Press [2] for the "Purchasing Emails Dataset."

2. **Choose a Classifier:**

- Select one of the following machine learning models:
- [1]: Random Forest
- [2]: SVM
- [3]: AdaBoost
- [4]: Naive Bayes
- [5]: Logistic Regression

3. **View Results:**

- The model's accuracy and confusion matrix will be displayed.

4. **Next Steps:**

- Continue evaluating another model.
- Load a new dataset.
- Exit the system.

5. **Logs**

- The results and events that took place are stored in the file "app.log" which is in the project root.

System Workflow

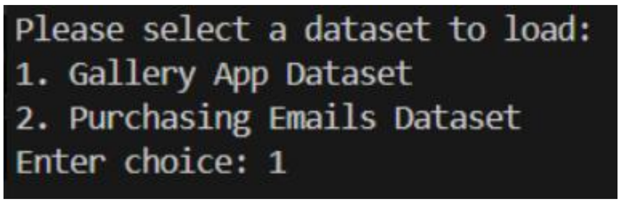
Launching the Application

- To start the application, the user just must run the min.py file from the command line.
- python main.py
- What happens After Launch:
- The system initializes and displays a menu for dataset selection.
- The user then interacts with the command line interface, who will follow the prompts to load a dataset and then select and classification model where it can view results.
- Logs are recorded in the “app.log” to track system events.

Step-by-Step Instructions

1. Loading a Dataset

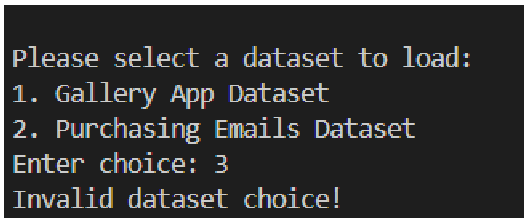
- User Action:
- The system prompts the user to select a dataset:



```
Please select a dataset to load:
1. Gallery App Dataset
2. Purchasing Emails Dataset
Enter choice: 1
```

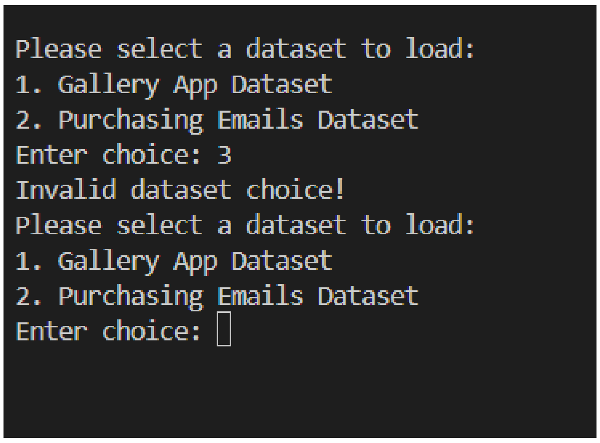
- The user inputs [1] for the Gallery App Dataset or [2] for the Purchasing Emails Dataset

- System Action:
- Based on the choice:
- The system loads the respective CSV file.
- If the choice is invalid, an error message will be displayed:

- 

```
Please select a dataset to load:
1. Gallery App Dataset
2. Purchasing Emails Dataset
Enter choice: 3
Invalid dataset choice!
```

- The user is prompted to reselect a valid option.

- 

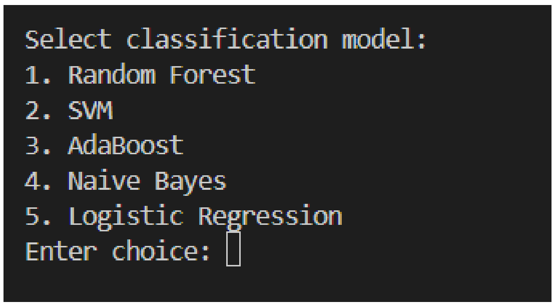
```
Please select a dataset to load:
1. Gallery App Dataset
2. Purchasing Emails Dataset
Enter choice: 3
Invalid dataset choice!
Please select a dataset to load:
1. Gallery App Dataset
2. Purchasing Emails Dataset
Enter choice: 
```

2. **Data Preprocessing**

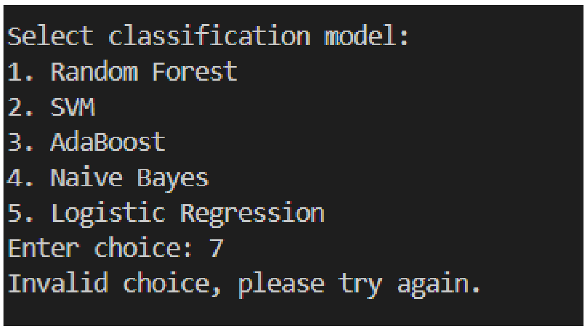
- What Happens:
- After the user loads the dataset, the system will preprocess the email data:
- Convert the email body to lowercase.
- Removes non alphabetic characters using regular expressions.
- Removes stopwords and applies stemming (if implemented)
- Drops rows with null values in the email body column.

3. **Choosing a Classification Model**

- User Action:
- The system prompts the user to select a classification model:

- 

```
Select classification model:
1. Random Forest
2. SVM
3. AdaBoost
4. Naive Bayes
5. Logistic Regression
Enter choice: 
```

- Choices Available:
- Random Forest: Ensemble based model suitable for general purpose classification.
- SVM: Effective for high-dimensional spaces
- AdaBoost: Combines weak classifiers to improve accuracy.
- Naive Bayes: Probabilistic model best for text classification
- Logistic Regression: Linear model for binary or multiclass classification
- Error Handling
- If an invalid choice is entered, the system will display an error:
- 

```
Select classification model:
1. Random Forest
2. SVM
3. AdaBoost
4. Naive Bayes
5. Logistic Regression
Enter choice: 7
Invalid choice, please try again.
```

- The user is prompted to be reelected.

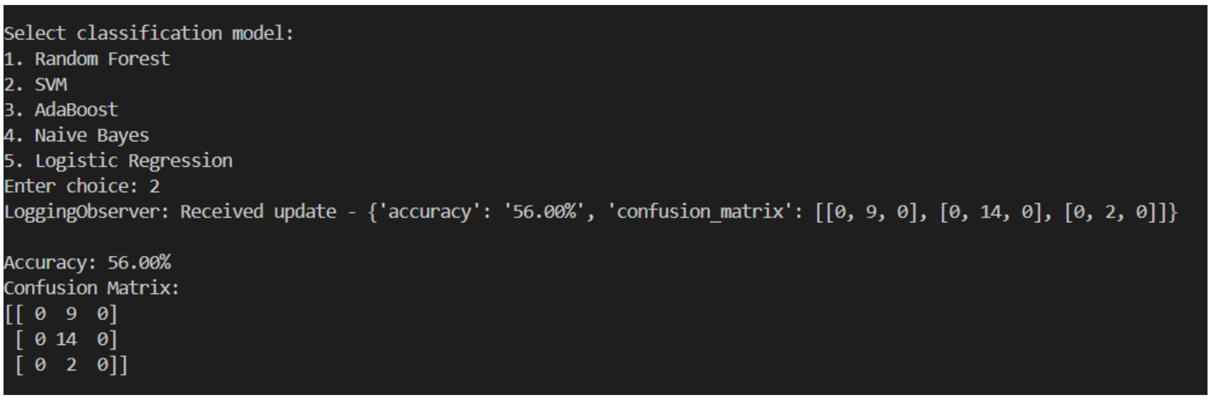
4. Model Training

- What happens:
- Once the user selects a model, the system will train it using the preprocessed and vectorized dataset.
- Training involves:
- Splitting the dataset into training (80%) and testing (20%) sets
- Then performing hyperparameter tuning with “GridSearchCV” to optimize the model

5. **Evaluating the Model**

- What Happens:
- After the system is done training, it will evaluate the model using the dataset.
- The metric calculated includes:
- Accuracy: The percentage of correct predictions
- Confusion Matrix: A table that summarizes true positives, true negatives, false positive and false negatives.

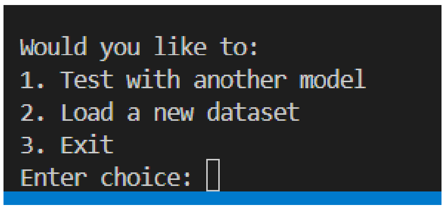
6. **Output and Results**

- What the user sees:
- The system displays the following metrics (for example with SVM):
- 

```
Select classification model:
1. Random Forest
2. SVM
3. AdaBoost
4. Naive Bayes
5. Logistic Regression
Enter choice: 2
LoggingObserver: Received update - {'accuracy': '56.00%', 'confusion_matrix': [[0, 9, 0], [0, 14, 0], [0, 2, 0]]}

Accuracy: 56.00%
Confusion Matrix:
[[ 0  9  0]
 [ 0 14  0]
 [ 0  2  0]]
```

- Next Steps:
- The user is prompted to choose one of the following:



```
Would you like to:
1. Test with another model
2. Load a new dataset
3. Exit
Enter choice:
```

- Based on the user's choice, the system allows:
- Re-selecting a model.
- Loading a different dataset
- Exiting the application

Configuring Various Classification Strategies

Overview Available Strategies

In our project, we implemented 3 strategies and concentrated on them. First of all, the strategy we implemented was the,

AdaBoost strategy:

The AdaBoost strategy is the short form of Adaptive Boosting Strategy, which is a technique to combine a lot of basic models such as decision stumps for developing a more powerful better classifier. It does this in steps, each step focusing on the mistakes made by earlier models by giving greater weight in the next round of training to those hard examples to classify. Each is a model dependent on the performance of the other: the final prediction weight votes across all models. This intelligent approach to improving the difficulty of getting things right makes AdaBoost the strong tool for classification and regression tasks.

Classification Strategy:

The next one we applied was classification strategy is a strategy for sorting data into predefined categories or classes. How classification works is that the model gets training on labelled examples where it already knows the correct category. It may learn patterns and features that distinguish the classes. After training, when it analyses new unseen data, the model will be able to predict the most likely category based on what it learned. This type of approach is useful in tasks like spam detection, image recognition, and sentiment analysis, where one needs to sort out data into accurate groups.

The Random Forest Strategy:

The third strategy we have implemented in our project is The Random Forest Strategy creates many decision trees and integrates the predictions of every one of them to come up with a robust and precise model. Every tree is trained on a random subset of the data, considering only a random selection of features added to the trees at each split. The final prediction is then averaged for regression or taken by the majority vote for classification. This is where diversity among these trees comes in handy, helps Random Forest to handle overfitting and makes it robust and versatile for many kinds of tasks.

Naive Bayes strategy:

Next, we implemented the Naive Bayes strategy that performs classification based on a simplistic yet powerful application of Bayes Theorem, assuming all features are independent of each other. This somewhat simple assumption often works very well in practice. The algorithm calculates probabilities for each class given feature likelihoods and then combines them to make predictions. Its most useful for things like spam filtering or sentiment analysis that requires high speed and efficiency.

Logistic regression:

Finally, we implemented the Logistic regression is a simple and widely used strategy, which by receiving the probabilities, predicts and categorizes data in classes. It models the relationship between input features and the likelihood of the particular outcome. It utilizes a logistic function that outputs values in the range between 0 and 1. This algorithm, throughout the training, changes the model parameters to minimize the error; typically, an approach like gradient descent is followed. Nevertheless simple, logistic regression performs outstandingly well for tasks like binary classification and can be easily extended to multiple classes or work with regularization to prevent overfitting.

Choosing a Strategy:

This system allows you to choose from and then test different classification strategies through a menu. You may choose, depending on data and purposes, between Random Forest, SVM, AdaBoost, Naive Bayes, or Logistic Regression. Then, each model has been fine-tuned using grid search such that the best setting for it was selected automatically. This will then train and evaluate your model by showing the accuracy and confusion matrix, logging the results for references in future comparisons. You can then choose to try another model, load a new dataset, or exit. It becomes easy to test strategies and find out what works best.

Adding New Strategies:

In order to add a new classification strategy to the system, one needs to create a new class for the classifier inside the folder we named 'classifiers'. There, in that folder, one can add their classifier file and implement all required methods. Then, one should change the Classifier Factory file, which is inside the 'factories' folder, to include their new classifier, which would be set dynamically. Finally, make sure the new classifier is added to the Classifier Façade file within the 'utils' folder where options that are available are listed so that it's easy to use in the system.

Understanding the Output

Accuracy:

Accuracy is a way to measure how well an email classification system works by looking at the percentage of emails it correctly identifies as either spam or not spam, ham. It gives the number of correct predictions divided by total number of emails processed. E.g., if the system gets 90 out of 100 emails right, its accuracy is 90%. But if a dataset has many more ham emails than spam, then accuracy all by itself might not provide the complete picture, so it's very often accompanied by other metrics like precision and recall for clarity.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Confusion Matrix:

A confusion matrix helps understand the general performance of the email classification model as it contrasts the predicted values against the actual labels, whether spam or not spam. It reflects the number of true positives-those that were actual spam, correctly identified; true negatives-those which are not spam, correctly classified; false positives-non spam emails which have been marked as spam, and false negatives-spam emails that have failed identification. From these values, one is able to get a glimpse of the strengths and weaknesses of the model, like how it mistakes spam for non-spam or vice versa, or how many spam emails it fails to catch. This could be very instrumental in refining the performance of the classifier in filtering out unwanted e-mails.

Logging Output:

The Observer pattern is implemented through the Observer and Subject classes. The Observer class defines a method to receive updates, whereas the Subject class manages a list of observers, which can be attached or detached and informs them with any changes in data. Precisely, LoggingObserver extends the Observer and is responsible for logging updates. Each time there is a classification update, LoggingObserver receives the data and prints it out, thus giving a neat log of the operations performed by the system. This logging mechanism is very important for performance tracking of models, debugging issues, and monitoring its behaviour over time, such as identifying misclassifications or errors during processing.

Testing the Design Patterns

Testing Singleton Pattern (ConfigurationManager):

How to Test:

The user should confirm that just one instance of the ConfigurationManager class is created while the program is running in order to confirm that the ConfigurationManager is appropriately implemented as a Singleton. By regularly getting the settings and making sure the instance stays the same, this can be tested.

Run the program several times, retrieving the ConfigurationManager object after each operation (e.g., selecting a classifier, loading a dataset).

Use Python's id () function to compare the instance memory locations to make sure the instance references are always the same.

Expected Out Behaviour:

To guarantee that the same configuration is used throughout the application's lifecycle, the ConfigurationManager should supply consistent configuration variables, such as

dataset paths or logging files, across various system components.

2. Testing Factory Pattern (ClassifierFactory):

How to Test:

To test the ClassifierFactory, the user can simulate multiple classifier selections within the application and validate that the proper classifier object is instantiated and utilised for training. A RandomForest classifier should be instantiated, for instance, if "Random Forest" is selected.

Steps:

1. When asked to choose a classification model during system operation, select various classifiers (e.g., "1. Random Forest").
2. Verify that the system trains the appropriate classifier after choosing one. Examining the best_model object that the facade returns will confirm this. Make sure the choose_classifier () method matches the appropriate type (for example, RandomForestClassifier when "Random Forest" is chosen).

Expected Out Behaviour:

The system should instantiate and apply the appropriate classifier for model training and evaluation when choosing different classifiers. For instance:

The RandomForestClassifier should be used when "Random Forest" is selected.

The SVC class should be used when "SVM" is selected, and so on.

3. Testing Strategy Pattern (ClassificationStrategy)

How to Test:

By checking to see if the appropriate classification strategy is being used based on user input (for example, choosing "SVM" should activate the SVMStrategy), the user can confirm the Strategy pattern.

Steps:

Select a classification model (like "SVM") from the list of alternatives.

Keep an eye on the system's behaviour to make sure the right strategy class—such as SVMStrategy—is called.

Verify that the outcomes of the model assessment align with the categorisation approach that was selected (e.g., a particular model evaluation metric or output format relating to the chosen strategy).

Expected Out Behaviour:

The system must employ a different model for training and assessment whenever the categorisation technique is altered (for example, selecting "SVM"). The chosen approach affects how the model is trained, and the outcomes are generated, thus the behaviour should align with it.

4. Testing Observer Pattern (LoggingObserver)

How to Test:

The user can connect a LoggingObserver to the system and confirm that log entries are generated when important events take place (for example, following model evaluation) in order to test the Observer pattern.

Steps.

Verify that the system is recording events; this is configured using logging. basicConfig() in run_email_classification_system().

Examine the contents of the log file (app.log) following model evaluation and training.

Important actions, such the trained model's accuracy or the confusion matrix following classification, should be noted in the log.

Expected Out Behaviour:

Entries in the log should inform the user of significant occurrences. For instance:

Accuracy and confusion matrix results, among other log entries, need to show up in the log file following model evaluation and training.

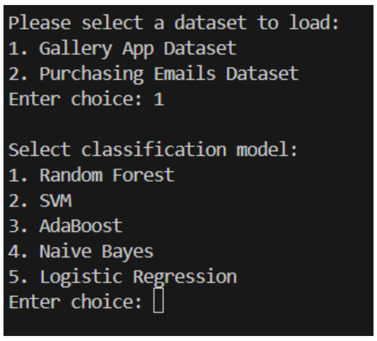
For later analysis, every important step in the classification process—such as loading the dataset, choosing a model, and evaluating it—should be recorded.

5. Testing Facade Pattern (ClassifierFacade)

The Facade shouldn't need to be tested because it has already been applied.

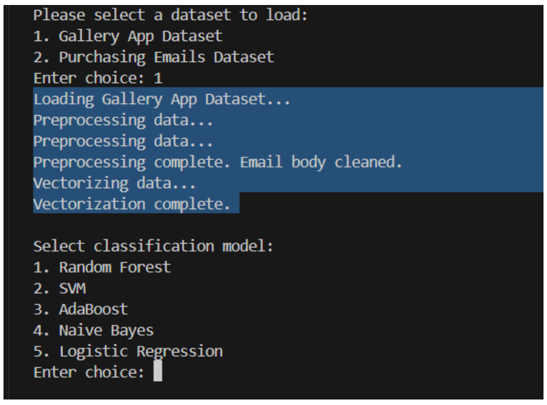
Additional Notes:

Code Snippet: This is a code snippet of the use of the ClassifierFacade in our email classifier system:



```
Please select a dataset to load:
1. Gallery App Dataset
2. Purchasing Emails Dataset
Enter choice: 1

Select classification model:
1. Random Forest
2. SVM
3. AdaBoost
4. Naive Bayes
5. Logistic Regression
Enter choice: 
```



```
Please select a dataset to load:
1. Gallery App Dataset
2. Purchasing Emails Dataset
Enter choice: 1
Loading Gallery App Dataset...
Preprocessing data...
Preprocessing data...
Preprocessing complete. Email body cleaned.
Vectorizing data...
Vectorization complete.

Select classification model:
1. Random Forest
2. SVM
3. AdaBoost
4. Naive Bayes
5. Logistic Regression
Enter choice: 
```

As you can see the difference in the 2 terminal the Façade gives the terminal a cleaner look by hiding the preprocessing stuff in the system.

Troubleshooting and FAQ

Common Errors

1. **Invalid Dataset Selection**
   Problem: Invalid option for dataset.
   Solution:

   - This means you has selected an invalid dataset option, so make sure you select a valid option, e.g. 1 for Gallery App Dataset and 2 for Purchasing Emails Dataset.
   - You should also check if the dataset files exist in the data/directory with the correct filenames (gallery_app.csv, purchasing_data.csv).
   - If you find out that files are missing, add them to the right directory or update the ConfigurationManager paths.

2. **Missing Files**
   Problem: System cannot find required files like datasets or log files.
   Solution:

   - Check if all required files are in the right location.
   - Confirm the file paths in ConfigurationManager.
   - Make sure you have permission to read the datasets and logs.

3. **Incorrect Model Configuration**
   Problem: During Model or Evaluation system fails due to incorrect hyperparameter settings.
   Solution:

   - Go and check the hyperparameter set in the ClassifierFacade.
   - Validate parameters or use default settings before applying them in grid search.
   - Make sure the training dataset is not too small.

Performance Issues

1. **Slow Training Times**
   Advice:

- Make the size of the dataset smaller by using smaller sample for testing purposes.
- Reduce grid search ranges or fewer cross-validation splits to optimize hyperparameter tuning.
- Try using a cloud-based environment with GPU acceleration or Upgrade hardware resources e.g., RAM or CPU.

2. **Low Classification Accuracy**
   Advice:

- Make sure dataset is preprocessed correctly.
- Try different classifiers to find the best performing model for your dataset.
- For better or for a more balance dataset, tune the hyperparameters.

3. Memory or Resource Constraints
   Advice:

- Try working with smaller batches of data during preprocessing training.
- Run a high-performance environment by increasing system resources

Frequently Asked Questions

1. **Q: What should I do if the system crashes during training?**
   A: First ensure that the dataset is correctly formatted, check if you have enough system resources and confirm that hyperparameters are within reasonable ranges.
2. **Q: Can I add a new classifier to the system?**
   A: Of course, to implement a new strategy you have to go to the ClassificationStrategy interface and integrate it into the ClassifierFactory
3. **Q: How can I improve the model's performance on my custom dataset?**
   A: Tune hyperparameters through grid search or trial-and-error, preprocess your dataset or use balanced classes.
4. **Q: What logs can I refer to for debugging?**
   A: You can look at the app.log file for detailed logs of system events, errors, and performance metrics.

Conclusion

Final Thoughts

Our Email Classifier System provides a nice flexible and user-friendly solution to sorting out emails effectively. And with the help of our user guide which provides a clear guidance on setup, operation, and troubleshooting, it will allow users to be able to use our system without major issues and allows users to customise it to their specific requirement. By carefully following the user guide, users can maximise the systems effectiveness and scalability.