DESIGN DOCUMENT

Email Classifier System

Authors

David Adediji-21311072, Keith O'Halloran-21334269, Rahul Alam-21317879, Noel Fontugne-21323372, Didier Bosiwa-20212267

The names are in order of who contributed the most into the entire project.

21311072@studentmail.ul.ie

Table of Contents

Introduction

Overview

By automatically categorising incoming emails into pre-established groups, our email classifier system assists users in managing their inboxes. Our system employs machine learning techniques to classify their emails into relevant categories like “Spam”, “Promotions”, “Work” and “Personal” along with others, thanks to this technology.

The main objective of our system is to offer a user-friendly, automated email classification solution that boosts productivity and decreases manual labour. Our system implements a variety of classification techniques, including random forest, Support Vector Machine (SVM), AdaBoost, Naïve Bayes, and Logistic Regression, allowing customers to choose the optimal classifier for their data and specific needs.

Problem Being Solved:

Sorting and managing emails can be very time consuming especially with the increasing volumes of emails. Email systems frequently offer simple filtering choices; however, these could not be precise or adaptable enough to accommodate various user preferences. Our Email Classification System allows users to save a huge amount of time and effort by automating the categorisation process.

- **Key Features:**
- **Dataset Selection**: Users can select different datasets to train and evaluate the classification models.
- **Model Selection**: Classification models like Random Forest, SVM, Naïve Bayes and Logistic Regression support our system to suit different use cases
- **Hyperparameter Tuning**: Our system has hyperparameter optimisation feature that uses GridSearchCV to fine-tune the models to improve the performance.
- **Evaluation**: Models are assessed according to their accuracy and confusion matrix, which gives a clear picture of how well they function.
- **Logging**: The system monitors performance by logging significant occurrences, such as classification results.

Scope

The Scope of our Email Classifier System is as the follow:

Types of Emails the System Can Classify:

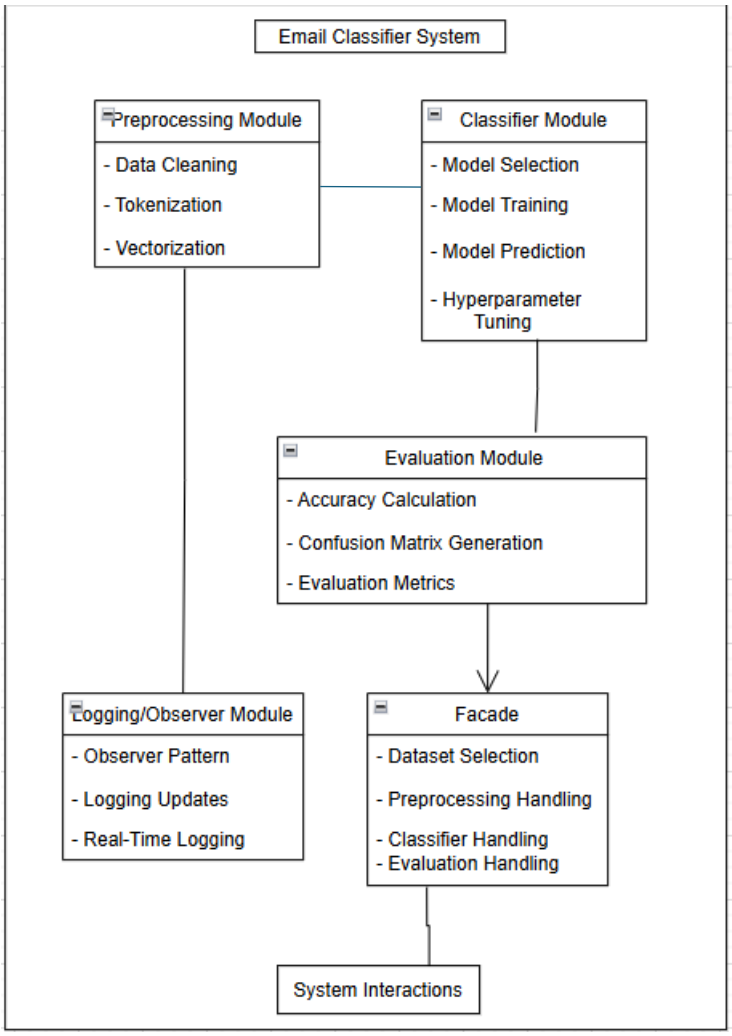
1. **Spam vs. Non-Spam**: The main classification task of our system is to be able to distinguish between span and valid emails. Our system will detect uninvited or irrelevant emails like ads or phishing attempts and classifies them as spam. Emails that are legitimate would be classified as “non-spam”.
2. **Purchase-Related vs. Other Emails**: Another classification task that our system handles are separating purchase-related emails such as receipts, order confirmation and shipment updates from all other emails. This allows users to access important transactions and prioritise their inboxes
3. **Work vs. Personal Emails**: To help users separate work and personal emails our system can classify those types of emails. This allows users to easily balance their work life and reply to emails more efficiently.
4. **Promotions vs. Other**: Promotion emails such as deals and offers from different suppliers can be classified as “Promotions”. The “Other” category would include general emails such as personal updates, newsletters and notifications.

System Architecture

Our Email Classifier System follows a system architecture where each key feature is included in separate components. These components preprocess data, classify emails, evaluate performance and manage system interactions all work together. Observer pattern is also integrated in our system to allow for real-time updates during the classification process. Below is an outline of the main components of our system architecture.

Component Diagram

Our email classifier system main modules and interactions are illustrated in the component diagram below:



So, each model of our email classifier system is focused on a certain task, making the system adaptable and simple to maintain, as you seen in the diagram above it places a great focus on the separation of concerns while defining the flow of actions between the models.

Component Breakdown and Data Flow

1. **Preprocessing Module**:
    - **Responsibilities**:
        - **Data Cleaning**: This converts text to lowercase and eliminates unnecessary characters and special symbols.
        - **Tokenization**: This breaks the email content into words(tokens)
        - **Vectorization**: **TF-IDF** is a method used to convert text into numerical feature vectors.
    - **Inputs**: Unprocessed email data
    - **Outputs**: Data that has been cleaned and vectorized for classification.
    - **Flow**: The classifier module gets the preprocessed data.

2. **Classifier Module**:
    - **Responsibilities**:
        - **Model Selection**: Depending on what user inputs the module chooses a classifier like AdaBoost, SVM, Logistic Regression etc.
        - **Model Training**: When the selected model is chosen, the model is trained using the training data.
        - **Model Prediction**: By using the trained model, the system makes predictions on the test data.
        - **Hyperparameter Tuning**: The GridSearchCV technique optimise the model.
    - **Inputs**: Data (X, y) that have been preprocessed by the Preprocessing Module.
    - **Outputs**: Results are predicted by the trained model.
    - **Flow**: After predictions are made, they flow to the Evaluation Module for examination.

3. **Evaluation Module**:
    - **Responsibilities**:
        - **Accuracy Calculation**: Determines the accuracy of the model's predictions.
        - **Confusion Matrix Generation**: Displays a confusion matrix to assess the model's performance. (e.g., true positives, false negatives)
        - **Evaluation Metrics**: Determines other metrics such as recall, precision and F1 score as needed.
    - **Inputs**: From the classifier module are predicated labels and true labels.
    - **Outputs**: Performance metrics such as the confusion matrix, accuracy etc.
    - **Flow**: Logging/Observer receive the evaluation metrics and are logged.

4. **Logging/Observer Module**:
    - **Responsibilities**:
        - **Observer Pattern**: This follows the observer design pattern to allow real-time logging of updates and notifications.
        - **Logging Updates**: Record key updates like classification results, model performance metrics and accuracy in the log.
        - **Real-Time Logging**: During the classification process it provides real-time feedback and logging.
    - **Inputs**: Performance metrics and updates from the Evaluation Module.
    - **Outputs**: Logs displayed in console output and written to a file
    - **Flow**: The logging data is saved in the log file, and it is also shown in real-time for the users

5. **Facade**:
    - **Responsibilities**:
        - **Dataset Selection**: User selects which dataset they want between the Gallery App and Purchasing Emails dataset.
        - **Preprocessing Handling**: Interacts with the Preprocessing module and manages preprocessing procedures.
        - **Classifier Handling**: Trains and selects classifier and communicates with classifier module
        - **Evaluation Handling**: Manages the assessment and provides the evaluation module with the results.
    - **Inputs**: Dataset choice, classifier choice etc.
    - **Outputs**: User friendly simplified interface with feedback on classification and evaluation results.
    - **Flow**: The Facade implemented in our system connects all the components together.

Data Flow Explanation

- **User Input**: Throught the Façade design implemented, the user selects a dataset and a classification model.
- **Data Preprocessing**: The **Facade** passes the data to the **Preprocessing Module** for cleaning, tokenization, and vectorization. The Preprocessing module receives the data from the Façade for cleaning, vectorization and tokenization.
- **Model Training and Prediction**: When the classifier module receives the processed data from the preprocessing module, it trains a machine learning model and uses it to make predictions.
- **Evaluation**: The evaluation module calculates the metrics once predictions are made, so that it can assess the model's performance.
- **Logging**: The results are log and displayed in real-time through the Logging/Observer module.

Design Patterns Used

Below are the design patterns that we employed for our email classification system:

Singleton Pattern

The first design pattern employed is a Singleton Pattern which basically ensures that only one instance of the ConfigurationManager can exit across the classifier system. By doing this it will help manage global settings such as file paths and log configurations. This works as the ConfigurationManager will create a single shared instance and any part of the system would be able to access this if they wanted to do such things like retrieve or update configuration settings. There are many benefits that come with this pattern, one of which is that it provides consistent settings throughout the entire system. Another benefit is that it is great at not duplicating configuration logic across multiple components. The final benefit would be that it allows for easy updates and debugging because it centralizes configuration management.

Factory Pattern

The second design pattern is the Factory Pattern which is used in the ClassifierFactory class. This pattern basically centralizes the creation of machine learning classifiers. By doing this it means that the user will then be able to choose from various options such as Random Forest, SVM, AdaBoost, Naive Bayes and Logistic Regression. This pattern works as the ClassifierFactory has a method take a user's choice (For example "Random Forest") and then it will return the appropriate classifier object. A benefit of this is that greatly simplifies the addition of new classifiers. It also keeps the code clean and focused on the system's core functionality. Another benefit is that it decouples classifier creation logic form other parts of the system.

Strategy Pattern

The third pattern used was the Strategy Pattern which is in the ClassificationStrategy class, and it basically allows the system to switch dynamically between different classification algorithms without changing the core logic. This strategy involves having classification algorithms implement the ClassificationStrategy interface and so by doing this it will define the methods for training, predicting and evaluating models. The system then at a runtime uses the selected strategy to perform the classification tasks. A benefit to this is that it will allow for the ability to be able to change the classification algorithms on the flight. It also helps maintain a clean separation of algorithm logic from the main system

Observer Pattern

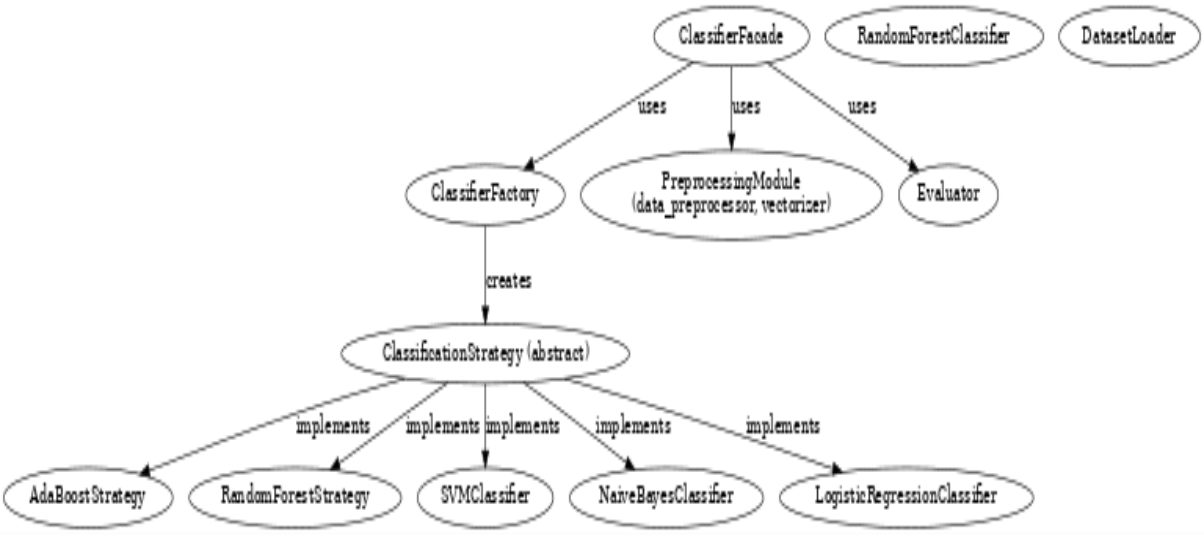
The fourth design pattern used was the Observer Pattern in the Subject and LoggingObserver classes. This design pattern basically will notify the observers when a major event occurs, for example if the model evaluation was completed. It works as the Subject class keeps a list of observers and when an event occurs it then notifies all observers. The LoggingObserver then logs classification results such as accuracy and confusion matrix to the system log. A key benefit to this is that it keeps logs automatically updated and provides real-time feedback. Another benefit is that it decouples the classification logic from ancillary tasks such as logging.

Facade Pattern

The final design pattern used was the Facade Pattern in the ClassifierFacade class. This pattern basically helps simplify the user interaction providing the user with a unified interface to complex subsystems such as data loading, preprocessing, model selection, training and evaluation. It works as the ClassifierFacade integrates calls to different components (Example: DatasetLoader, DataPreprocessor, ModelTrainer) all into a single, easy to use interface. The users will only be interacting with the facade which handles all the underlying complexity. A key benefit to this system is that it hides the complexity of the system and because of this it means that it will be easier for users to work with. Another benefit is that it centralizes the management of subsystems which then simplifies maintenance and updates. Another benefit is that it provides a single point of entry for system interactions which means that usability would be improved

UML Diagrams

Class Diagram:



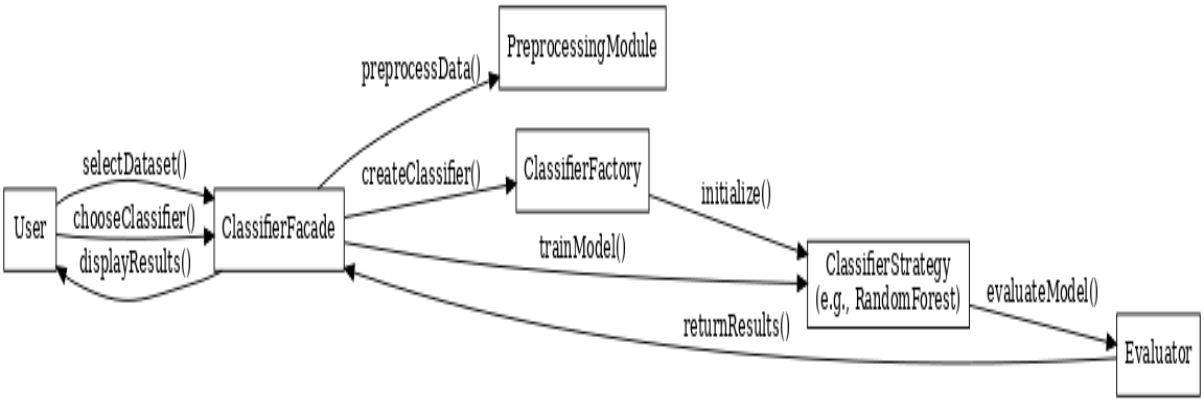
The Class Diagram illustrates the structure of the system by showing its key classes, their relationships, and how they interact.

It demonstrates the inheritance hierarchy for the classification strategies (e.g., RandomForestStrategy and SVMClassifier inherit from ClassificationStrategy).

Shows how the ClassifierFacade acts as the central interface, coordinating actions with the factory, preprocessing module, and evaluator.

Highlights design patterns such as the Strategy Pattern is used for classification strategies and Facade Pattern which would simplify interaction with subsystems.

Sequence Diagram:



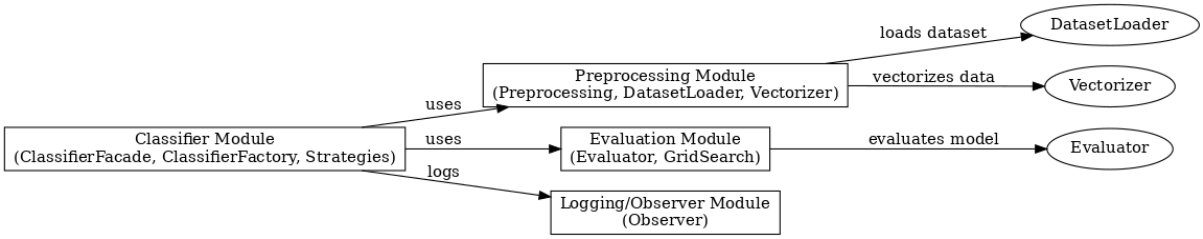
This diagram shows how the system processes an email classification task, highlighting the sequence of interactions between objects over time.

It starts with the user selecting a dataset and ends with the results being displayed.

Key modules like the **PreprocessingModule**, **ClassifierFactory**, and **EvaluationModule** are shown.

Demonstrates the logical flow of method calls: data preprocessing → classifier creation → model training → evaluation.

Component Diagram:



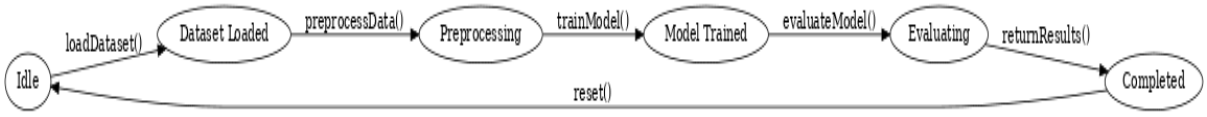
The Component Diagram provides a high-level view of the system's architecture, showing its major components and their dependencies.

Depicts modular design with distinct modules like Classifier Module, Preprocessing Module, and Evaluation Module.

Illustrates how components interact, such as the Classifier Module depending on the Preprocessing Module for clean data and the Evaluation Module for performance feedback.

Emphasizes the importance of the Logging/Observer Module in tracking system events.

State Diagram:



The State Diagram visualizes the different states the system could be in during the classification process and the transitions between them.

Includes states such as Idle, DatasetLoaded, Preprocessing, ModelTrained, and Completed.

It shows how user actions or system processes trigger transitions between these states, providing clarity on the system lifecycle.

Evaluation and Performance Metrics

In this section we will discuss the evaluation of the email classifier systems performance, we will go over used measures like accuracy and confusion matrix, and we will also discuss how the system is built to grow more data or how classifiers are added.

1. Evaluation of Model Performance

To really show how effectively the classifier is performing, the model's performance must be evaluated. Theses are the following metrics used:

Accuracy

When it comes to categorisation jobs, accuracy is the most obvious metric. The ratio of accurate predictions to all predictions is what is known as this:

Accuracy= Number of correct predictions/Total number of predictions

- **Use case**: Accuracy is useful metric when classes are spread or when the dataset is balanced. In situations where the classes are unbalanced such as when the quantity of spam emails is significantly lower than that of non-spam emails, it might not give a clear picture.
- **Implementation**: After testing the data, a percentage score shows the percentage of correctly classified emails produced by the system.

Confusion Matrix

The confusion matrix is a more thorough assessment of classification performance, particularly when it comes to binary categorisation (such as spam vs non spam). It helps comprehend the many kinds of mistakes the model makes. Theses are the four parts of the matrix:

- **True Positives (TP)**: The number of positive cases that are appropriately classified (e.g., spam emails correctly labelled as spam).
- **True Negatives (TN)**: The number of negative cases that were accurately classified (e.g., non-spam emails correctly labelled as non-spam).
- **False Positives (FP)**: The number of negative cases that are mistakenly categorised as positive (e.g., emails that are not spam that are mistakenly classed as spam).
- **False Negatives (FN)**: The number of positive cases that are mistakenly categorised as negative (such as spam emails that are mistakenly categorised as non-spam).

From We can obtain the following from the confusion matrix:

- **Precision**: Out of all occurrences anticipated as positive, the percentage of accurately predicted positive instances is as follows:

$$\text{Precision} = TP/TP+FP$$

- **Recall (Sensitivity)**: Out of all actual positive events, the percentage of accurately anticipated positive instances is:

$$\text{Recall} = TP/TP+FN$$

- **F1-Score**: The precision-recall harmonic mean, which balances the trade-off between the two:

$$F1 = 2\times(\text{Precision}\times\text{Recall})/(\text{Precision}+\text{Recall})$$

Evaluation in the System

The following steps are followed in our email classifier system to assess the model following training using a chosen classifier (e.g., Logistic Regression, SVM):

1. The test sets true labels are compared with the models' predictions.
2. Based on the accurate predictions, the accuracy is calculated.
3. The confusion matrix is calculated by evaluating the kinds of classification errors the model makes

These evaluation metrics are displayed and logged through the LoggingObserver in real time, ensuring that stakeholders can access the performance insights immediately.

2. Scalability Considerations

Our email classification systems efficiency and performance must be maintained without sacrificing the design integrity as it expands in terms of both dataset size and classifier count. The following is addressed in the systems scalable design:

Handling Increasing Data Volume

- **Efficient Data Handling**: Scikit-learns is used in the system to optimise data structures when handling big datasets without encountering memory or performance bottlenecks.
- **Data Preprocessing**: Two examples of preprocessing techniques that are intended to be effective are vectorisation and tokenisation. Emails are converted into numeric

vectors using Term Frequency -Inverse Document Frequency (TF-IDF) vectorisation which ensure that even massive text datasets are managed well.

- **Incremental Learning**: Models such as Naïve Bayes or Stochastic Gradient Descents (SGD) classifiers might be incorporated into the system to handle even larger datasets. This will engage the system to learn from batches of data instead of loading the full complete dataset into the memory all at once.

Adding New Classifiers

- **Modular Classifier Design**: Factory and Strategy patterns allow new classifiers to be added with ease. A developer can add a new machine learning model by updating the factory method and implementing a new classifier strategy without changing the core system.
- **Separation of Concerns**: To guarantee that adding or changing a classifier won't necessitate major adjustment to other system components data preprocessing, classification, evaluation and logging are all divided into independent components by the system design
- **Dynamic Classifier Selection**: Dynamically selecting several classifiers (such as SVM, AdaBoost etc,) during runtime, users can easily experiment and scale the classification process.

Distributed Processing and Parallelization

- **Parallelization**: Pythons multiprocessing or joblib modules can be used to parallelise data preprocessing such as vectorisation and tokenisation and model training. Our system can be modified to do some operations foe huge datasets.
- **Cloud or Distributed Environments**: For our system to be able to grow horizontally and analyse massive amounts of data across numerous nodes our system would need to be expanded to operate in distributed systems like Apache Spark or cloud environments like AWS and Google Cloud.

Performance Monitoring and Optimization

- **Real-Time Monitoring**: As the systems data volume and complexity increases the model performance would need to be monitored regularly. So, our Observer pattern can be expanded to monitor memory use and execution times, which helps in the identification of bottlenecks
- **Model Tuning and Hyperparameter Optimization**: Because of our simple architecture system it makes it easy to apply hyperparameter optimisation strategies, like random search or grid search, to optimise the classifiers performance even when handling big datasets

Conclusion

Our email classifier system provides and efficient solution for categorizing emails into meaningful categories, such as spam vs non-spam or purchase related emails. Our systems guarantee flexibility, scalability and maintainability by using key designs such as singleton, factory, strategy, observer and façade. Performance metrics like confusion matrix and accuracy allows for comprehensive evaluation of the classification models. Our system has been designed to support the addition of new classifiers and handling increasing data volumes with ease. Overall, our system presents an adaptable and robust framework for classifying emails effectively, making it suitable for real world applications.