REPORT: Machine Learning Process - Gas Leak Detection with Traditional Classification ML Techniques

Background

Many gaseous emissions are colorless, odorless, and tasteless, which makes them challenging to detect using human senses alone. Moreover, relying on a single sensor for gas detection may not provide accurate results, as numerous real-world applications require robust and reliable detection methods. Consequently, there is a need to develop efficient gas detection systems that improve accuracy and dependability by incorporating classification-based machine learning techniques. To enable prompt implementation in emergency systems, it is essential that these machine learning algorithms can predict and retrain quickly. Therefore, we employ straightforward traditional machine learning approaches, such as k-Nearest Neighbors (kNN) and RandomForest, to address this requirement.

A non-ML solution approach may possible to solve this issue by setting threshold values for each gas sensor value based on observed patterns in the sample data (using Data Analysis). However, this approach may not work well for more complex patterns or when the differences between gas types are subtle. Besides that, manually defining thresholds can be time-consuming and may require regular updates as conditions change. Given the explanation above, using machine learning techniques to solve this classification issue can be more effective and efficient approach.

Objectives

By utilizing machine learning techniques for gas detection and classification, we can expect several benefits:

1. Improved accuracy: ML models can capture complex patterns and relationship between sensor readings
2. Adaptability: ML models can be retrained with new data as condition change, making the system more dynamic and robust.
3. Automation: ML can automate the process of classifying gas types, reducing the need for manual analysis, which can save time and resources.
4. Scalability: ML models can be scaled to handle large amounts of data. Even as the volume of data increases, ML can leverage this additional information to generate more accurate and robust classifications, enhancing the overall effectiveness of the gas detection system.

In this project, our objective is to design a machine learning model that can efficiently and accurately identify the type of gas present in the environment in real-time.

Business Metrics

In this project, we chose accuracy as the primary evaluation metric because it quantifies the proportion of correctly classified instances out of the total instances. In scenarios where the classes are roughly balanced, as is the case with our gas type classification problem, accuracy provides a reasonable measure of overall model performance. Our system should be capable of accurately classifying the type of gas currently present in the environment, enabling personnel monitoring the system to take appropriate and timely actions.

However, accuracy alone might not provide a comprehensive view of the model's performance across different classes, especially when considering the cost of misclassification varies among the classes.

Therefore, we also considered the Classification Report, which includes precision, recall, and F1-score. These metrics provide insights into the model's ability to correctly identify each class while minimizing false positives and false negatives.

Lastly, we used the ROC AUC curve to assess the model's discriminative ability in a one-vs-rest (OvR) setup. This metric provides insights into the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) for each class. The ROC AUC curve enables us to visualize the overall performance of the model across a range of classification thresholds, making it a valuable tool for understanding how well the model distinguishes between different gas types in various operating conditions.

Machine Learning Solutions

In this section, we conducted experiments using various algorithms (Logistic Regression, Decision Tree, Random Forest, k-Nearest Neighbors, XGBoost) with various hyperparameters, used cross-validation to estimate model performance, and evaluated the models using several metrics on sklearn classification report. This comprehensive approach allows us to select the best model for gas detection and identification using the given dataset.

The best ML will be chosen for further experiment. We will try various hyperparameters to find the best combination for this gas detection and identification case. Hyperparameters play a crucial role in determining the performance of the models, and finding the optimal set of hyperparameters is essential for achieving high classification accuracy.

In this study, we utilize the dataset obtained from Narkhede P., et al.'s paper [(1)]. The dataset consists of 6400 samples, which are divided into training, testing, and validation sets using a ratio of 0.6:0.2:0.2, resulting in 3840 samples for training, and 1280 samples each for testing and validation. A sample of the data is shown below:

MQ2	MQ3	MQ5	MQ6	MQ7	MQ8	MQ135	Gas	Image Name
555	515	377	338	666	451	416	NoGas	0_NoGas
555	516	377	339	666	451	416	NoGas	1_NoGas
556	517	376	337	666	451	416	NoGas	2_NoGas
556	516	376	336	665	451	416	NoGas	3_NoGas
556	516	376	337	665	451	416	NoGas	4_NoGas

The dataset comprises values from seven Metal Oxide Semiconductor (MQ) gas sensors (MQ2, MQ3, MQ5, MQ6, MQ7, MQ8, and MQ135), with each sensor being sensitive to different gases and thermal images of each samples. For simplicity purpose, we will not use the image data. Each sensor's output is an integer value, with the minimum and maximum values ranging from 275 to 824 across all sensors. The target variable is the *Gas* column, which consists of four classes representing different types of gases. Here is the lists the gas sensors used in the dataset and their corresponding sensitive gases:

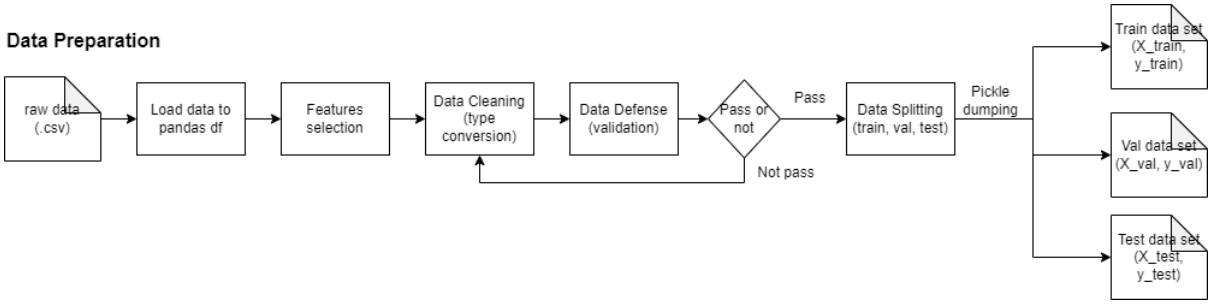
Sensor	Sensitive Gas
MQ2	LPG, Butane, Methane, Smoke
MQ3	Smoke, Ethanol, Alcohol
MQ5	LPG, Natural Gas
MQ6	LPG, Butane

MQ7	Carbon Monoxide
MQ8	Hydrogen
MQ135	Air Quality (Smoke, Benzene)

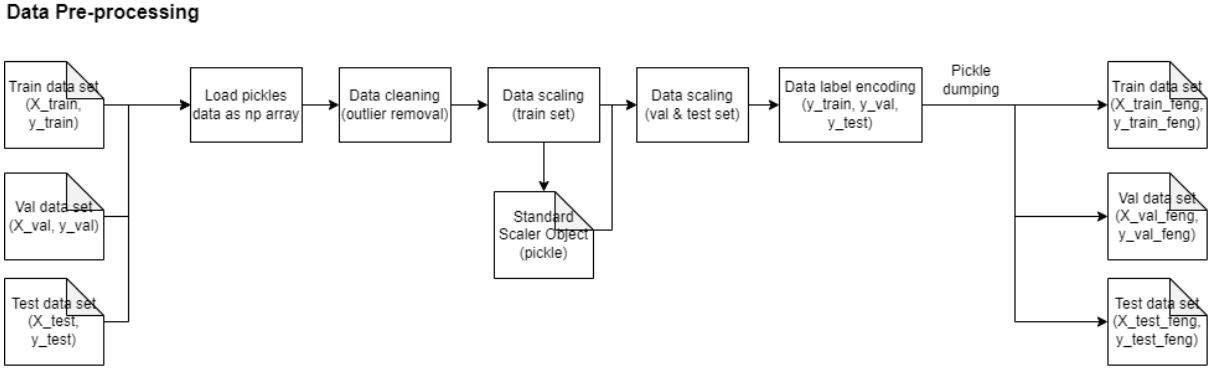
Methodology

Our project consists of several typical machine learning research process, which are:

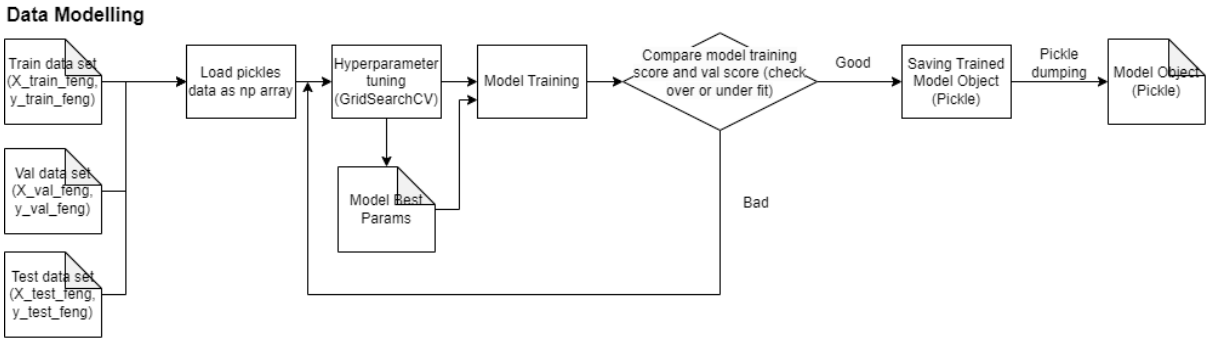
1. Data Preparation: Involves loading and cleaning the data.



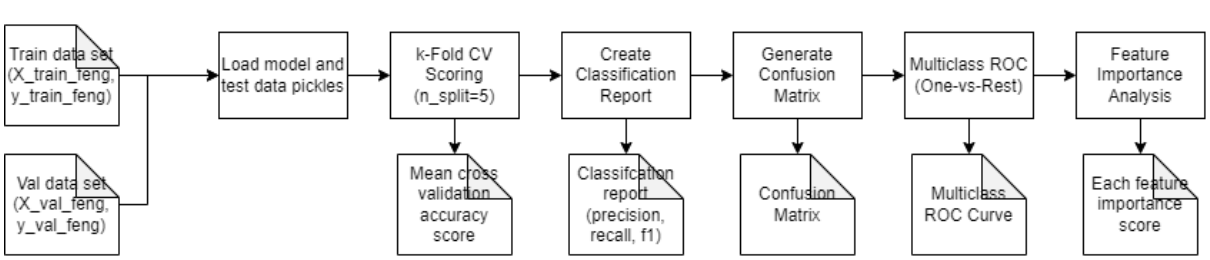
2. Data Pre-processing: Consists of feature engineering to enhance model performance.



3. Model Training: Includes hyperparameter tuning and training the model using the best parameters.



4. Model Evaluation: Assesses the best models' performance based on accuracy, precision, recall, ROC, and AUC metrics.



Initially, we will focus on these machine learning research processes to identify the most suitable model with optimal training time and accuracy. Once the best model is selected, we will proceed to the next phase, which involves deploying the machine learning model. We will utilize FastAPI and Streamlit for deploying and testing our trained ML model in real-world scenarios.

Deployment Process

The ML deployment process starts with importing required modules and loading the trained machine learning model (the best ML model). The FastAPI application is initialized, and a Pydantic model is made for input validation (data type and range). A */predict_gas* endpoint is created to accept sensor data as input and make predictions with the ML model. The input data is transformed using a pre-loaded scaler (fitted on training data) before making predictions. The endpoint gives back the predicted gas type and its probability. To run the FastAPI application, the script uses Uvicorn when run directly, serving the app on a local IP address and port. This deployment process allows users to interact with the trained machine learning model through a RESTful API.

The Streamlit application provides an interactive user interface for testing the machine learning model API in real-time. The app starts by importing required modules (Streamlit and Requests) and setting the FastAPI URL. Users can input values for each sensor. By clicking the "Predict" button, the input data is packaged into a dictionary and sent as a POST request to the FastAPI endpoint we've created before. If the request is successful, the app extracts the predicted gas type and probabilities, then displays the results. In case of an error, the app shows an error message, prompting the user to check the FastAPI server. This Streamlit app enables users to interact with the deployed machine learning model conveniently and obtain real-time gas type predictions based on the provided sensor data.

Result Insights

We trained multiple ML models using various algorithms, including logistic regression, decision trees, random forests, k-Nearest Neighbors, and XGBoost. After evaluating these models on a validation set, we found that the top two performers, Random Forest and kNN, had similar F1 scores. Logistic Regression didn't perform well due to our dataset complexity (asymmetric distribution). We then tuned their hyperparameters for optimal performance. Below is the result of the hyperparameter tuning for each model:

Random Forest	k-Nearest Neighbors (kNN)
• **n_estimators**: The number of trees in the forest -> 50 • **max_depth**: The maximum depth of the tree, with values -> None • **min_samples_split**: The minimum number of samples required to split an internal node -> 2 • **min_samples_leaf**: The minimum number of samples required to be at a leaf node, with values -> 1	• **n_neighbors**: Number of neighbors to consider -> 5 • **weights**: The weight function used in prediction -> distance-based • **algorithm**: The algorithm to compute nearest neighbors -> ball_tree • **leaf_size**: The leaf size passed to the BallTree or KDTree -> 10

Comparing the models, kNN had a slightly higher accuracy (5-fold mean cross-val accuracy 0.96959 and test set accuracy 0.97344) and a significantly faster training time (0.004025 seconds) compared to Random Forest (5-fold mean cross-val accuracy 0.96906 and test set accuracy 0.97109) with 0.263005 seconds training time.

Both models performed well in distinguishing the four gas types, but with some error on classifying class 1 and 2 (NoGas and Perfume). kNN having marginally slightly better ROC AUC scores for classes 1 and 2. But despite minor differences in classifying classes 1 and 2 (NoGas and Perfume), both models performed well in classifying each class. Here are the details of classification report of Random Forest model (left) and kNN model (right) on test set:

Gas Type	precision	recall	f1-score
Mixture	1.00	1.00	1.00
NoGas	0.94	0.95	0.94
Perfume	0.95	0.94	0.94
Smoke	1.00	1.00	1.00

Gas Type	precision	recall	f1-score
Mixture	1.00	1.00	1.00
NoGas	0.96	0.93	0.95
Perfume	0.94	0.96	0.95
Smoke	1.00	1.00	1.00

Due to its faster training time (65 times faster) and slightly better accuracy, we selected kNN for deployment so we can have a faster re-training time in the future.

Based on our ML model deployment tests using FastAPI, we evaluated our API with three scenarios: sending ideal input values, sending out-of-range input values (0-1000), and sending mistyped input values (string instead of integer). The API server successfully handles all scenarios. With the help of Pydantic BaseModel, our input is validated, and the server returns a 422 Unprocessable Entity error whenever we send invalid input (mistyped or out-of-range). Our tests also show that the API's latency is approximately 0.01 - 0.015 seconds, or 10 - 15 ms (run 10 times). This indicates that our model is suitable for real-time implementation.

Streamlit deployment tests also demonstrate that the API works seamlessly with front-end frameworks. In our sample Streamlit dashboard, we use a slider input with a range of 0 - 1000. Any errors and prediction request results are clearly displayed in the prediction result section.

References

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. doi:10.1109/MCSE.2007.55

Narkhede, P.; Walambe, R.; Chandel, P.; Mandaokar, S.; Kotecha, K. MultimodalGasData: Multimodal Dataset for Gas Detection and Classification. Data 2022, 7, 112. https://doi.org/10.3390/data7080112

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.