Chapter 2

Materials, Methods and Technology Background

In this thesis, we have explored the presence of various VOCs, gases and odors, generally considered as airborne hazards affecting human health indoors and outdoors during our daily living, maybe at smart homes or in industries or places of work.

Accordingly, we have proposed various prototypes and intelligent data processing approaches suitable for real-time air ambient detection and monitoring. In this section, we describe the details of the developed prototypes and the theoretical details of related data processing techniques as a ready reference.

We have developed five different types of prototypes, as described in Table 2.1.

Table 2.1 Description of the physical prototypes developed

S. No.	Prototypes	Gas sensors	PM Sensor	Temp. & Hum. sensor	Applications
1.	Standalone	MQ3, MQ4, MQ5, MQ7, MQ8, MQ135	×	DHT 22	Smart homes
2.	Standalone	MQ3, MQ4, MQ5, MQ7, MQ8, MQ135	PMS 5003	DHT 22	Fire Hazards
3.	LoRa Networked (SX 1278)	MQ2, MQ3, MQ5, MQ6, MQ7, MQ8, MQ135	×	DHT 22	Remote monitoring of Airborne Hazard
4.	Cloud Connected (AWS)	MQ3, MQ7, MQ135, MQ136, MQ137, MQ138	×	DHT 22	Disinfectants detection and monitoring
5.	Cloud Connected (AWS)	MQ2, MQ3, MQ5, MQ6, MQ7, MQ8, MQ135	×	×	Qualitative BGL estimation

Broadly, our developed prototypes include a tin-oxide-based metal-oxide (MOX) gas sensor array, a DHT22 sensor, and a PM sensor which generates real-time signature patterns of the smoke present in the ambient air. The proposed e-nose design's major component is a glass gas chamber. Inside this gas chamber, all the sensors are fitted on the sensor board, and wire connections are made with the microcontrollers. It comprises an electronic control and computer units for real-time data acquisition and processing. The electronic module contains two 32-bit microcontrollers, one for the PM Sensor operations while the other interfaces with the rest of the sensors. Using an integrated development environment (IDE), an essential communication protocol was set up between the microcontrollers and the computer to send the data generated during the experiment and to synchronise with the beginning and end of the data capturing.

2.1 General Experimental Setup

Post fabrication, the chamber prototype has a dimension of 29 cm x 21 cm x 12 cm providing a total interior volume of 7.308 L (7308 cm2). Further details of the gas chamber have been given in section 2.1.1.9. The general experimental setup is shown in Figure 2.1.

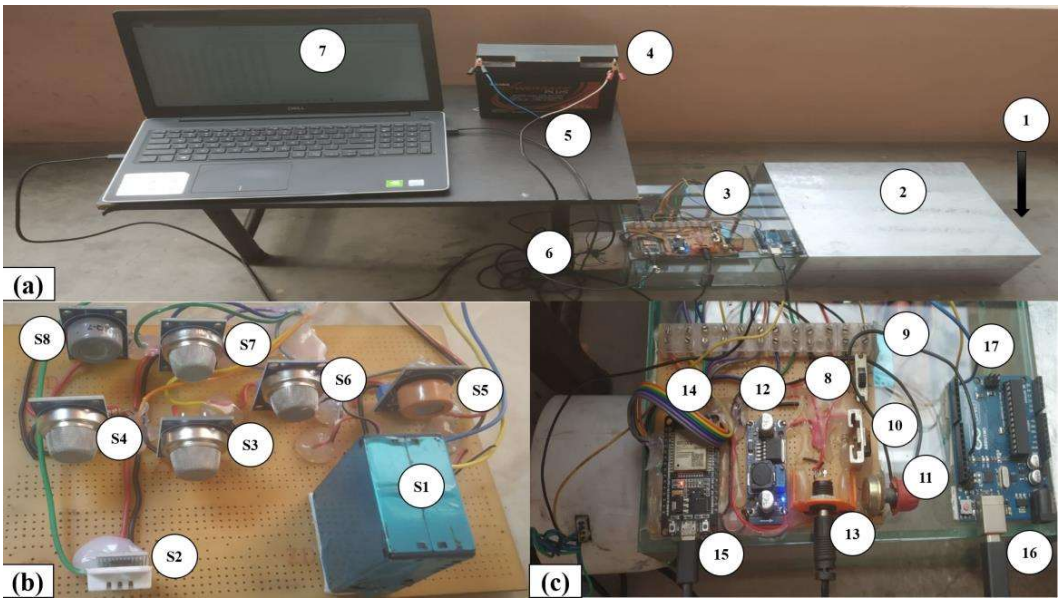

Figure 2.1. (a), (b), (c) General experimental setup

1: Gases/odors inlet; 2: Air Duct; 3: Sensor Chamber; 4: Power Supply (12V DC); 5: Power Cable; 6: Exhaust fan; 7: Laptop for sensor response capturing into text format;

S1: Sensor 1, S2: Sensor 2, S3: Sensor 3, S4: Sensor 4, S5: Sensor 5, S6: Sensor 6, S7: Sensor 7, S8: Sensor 8, 8: On-Off Switch; 9: Internal-External wire connecting point; 10: Heat sink; 11: Voltage Regulator; 12: Buck-converter; 13: Power distribution point; 14: ESP 32; 15: UART Cable-I; 16: UART Cable-II; 17: Arduino Uno

Details of the components and their basic electrical characteristics as used in the sensor node prototype development of the proposed advanced e-noses have been given in Table 2.2.

Table 2.2. Ratings of the Components as Used in Prototype

Components	Input Voltage	Power Ratings
PMS 5003	5V	100 mA
Arduino UNO	5V	50 mA
Arduino TX/RX pins	3.3V	40 mA
ESP32	5V	130 mA
ESP32 GPIO pins	3.3V	40 mA
DC-DC Buck converter	5V	2.5A
DHT22	3-5V	2.5 mA
MQ sensor	5V	150 mA

2.1.1 Sensors, Devices and Microcontrollers as Used

2.1.1.1 Tin-oxide MOX-based Gas Sensors

Tin-oxide (SnO2) metal oxide (MOX) gas sensors are widely used for gas detection and monitoring in various industrial and domestic applications due to their high sensitivity, selectivity, and low cost. The MOX-based gas sensors operate on the principle of detecting changes in the electrical conductivity of the sensing material when it comes into contact with a target gas.

In a typical SnO_2 MOX-based gas sensor, the sensing element is made of a thin film of tin oxide deposited on a ceramic substrate, which serves as the electrodes. The tin oxide film is usually doped with other metal oxides to enhance its gas-sensing properties. The sensor is operated at high temperatures (around 300 - 400^{o}C) to increase its sensitivity and selectivity to target gases.

When a target gas is present, it reacts with the tin oxide surface, causing a change in the material's electrical conductivity. This change is determined by measuring the resistance of the sensing element using an external circuit. The magnitude of the resistance change is proportional to the concentration of the target gas in the surrounding atmosphere.

The SnO_2 MOX-based gas sensors are commonly used to detect carbon monoxide, nitrogen dioxide, methane, propane, and ethanol. However, the sensitivity and selectivity of the sensor can be influenced by several factors, such as the operating temperature, humidity, and the presence of interfering gases. Depending on the sensing material characteristics and their dopants, different sensor elements are available in the commercial market. Some of the popular MQ series-based gas sensor elements and the gases for which they have been designed are listed in Table 2.3.

Table 2.3 MQ series-based gas sensors and their salient properties

	Sensor Name	**Target Gas/Odor**	**Detection Ranges (PPM)**
1	MQ 2	LPG, Smoke, Alcohol	200 - 10000
2	MQ 3	Alcohol, Ethanol, Smoke	25 – 500
3	MQ 4	Methane, CNG	300-10000
4	MQ 5	Natural Gas, LPG	300-10000
5	MQ 7	CO	10 - 500
6	MQ 8	Hydrogen	100-10000
7	MQ 135	Air quality	10 - 1000
8	MQ 136	Hydrogen sulfide	1 - 200
9	MQ 137	Ammonia, Carbon-monoxide	10 - 300
10	MQ 138	VOCs	1 - 100

Working principle: The tin-oxide MOX-based gas sensors work on the principle of changes in the sensor's electrical resistance when it interacts with the target gas. The sensor's surface is coated with a thin film of tin-oxide, which acts as a sensing material. When the gas molecules come in contact with the sensing material, the sensor's resistance changes due to the changes in the conductivity of the sensing material. Details of MQ series sensors are shown in Figure 2.2.

Figure 2.2 Details of MQ series sensors

2.1.1.2 PM Sensor

Particles of small size of solid or liquid substance floating in the air are called particulate matter (PM). Dust, soot, smoke, chemicals, and other things may all be found in these particles. They might be as little as a few nanometers or as large as tens of micrometres. They may originate from man-made sources, including vehicle exhaust, industrial pollutants, building activities and natural sources like fires and dust storms.

Depending on their size, PM may be categorised. PM2.5 indicates particles with a diameter of no more than 2.5 micrometres, while PM10 indicates particles of no more than 10 micrometres. Tiny particles can be more dangerous to human health because they may go deep inside the lungs and even reach the bloodstream. High PM exposure may adversely affect a person's well-being, including issues with the heart and lungs. To minimise PM levels, several nations have established air quality guidelines.

The PMS 5003 sensor monitors the amount of PM in the air. It can measure particles with a diameter of 1 micron or more, including PM1.0, PM2.5, and PM10. It employs laser scattering technology to detect the PM concentration in the air. It illuminates particles with a laser as they go through a detecting chamber within the sensor. A photodetector can detect the pattern of light created due to the particles' scattering of the laser light as they go through the chamber. This sensor employs the

Mie scattering method to detect particles with a diameter of 1 micron or larger. PMS 5003 PM sensor is shown in Figure 2.3.

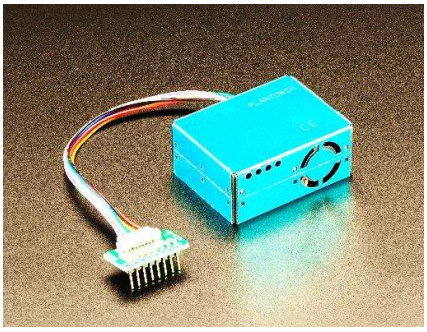

Figure 2.3 PMS 5003 PM sensor

Systems for measuring air quality often use the PMS 5003 sensor. It is used in many different applications, such as portable air quality monitors, indoor and outdoor air quality monitors, and interior air quality monitors. The sensor may transmit data using a variety of interfaces, including UART, I2C, and analog voltage output, and commonly produces data in the form of PM concentration levels. The airflow diagram of the PM sensor is shown in Figure 2.4, and the circuit connection diagram of PMS 5003 with Arduino UNO in Figure 2.5.

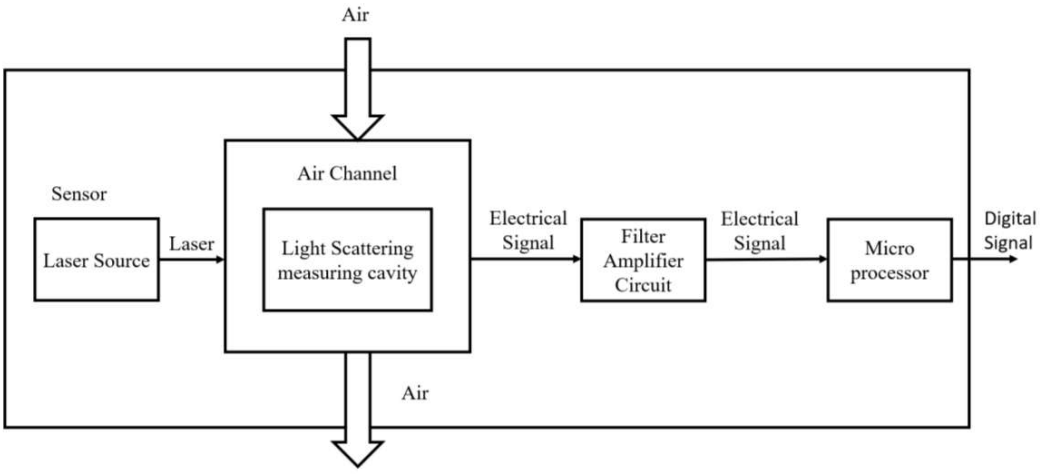

Figure 2.4 The air flow diagram of PM sensor

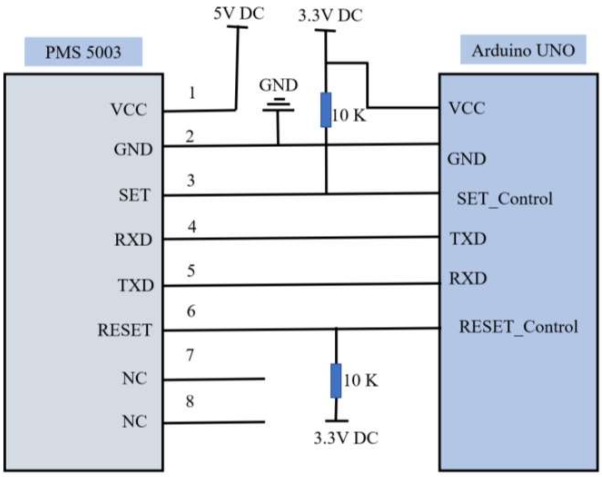

Figure 2.5 Circuit Connection diagram of PMS 5003 with Arduino UNO

2.1.1.3 DHT 22

An electrical device known as a temperature and humidity sensor detects the immediate surroundings' relative humidity and ambient temperature. These sensors are frequently utilised in various applications, including controlling indoor air quality, HVAC (heating, ventilation, and air conditioning) systems, and weather monitoring.

A thermistor, a form of resistor that changes resistance in response to temperature, or a thermocouple, which produces a voltage that changes with temperature, are the two most common temperature sensors. Humidity sensors monitor the quantity of moisture in the air using various methods, including capacitive, resistive, and optical sensing.

The DHT 22 sensor is a digital temperature and humidity sensor that measures the temperature and humidity of the surrounding air using a capacitive humidity sensor and a thermistor. It is a reasonably priced sensor with various uses, including home automation, weather monitoring, and greenhouse management systems. The DHT 22 sensor has a one-wire interface for communication with microcontrollers or other devices, making integration simple. It offers precise temperature measurements in the -40 to 125 degrees Celsius and 0 to 100% relative humidity ranges. The temperature and humidity sensor is shown in Figure 2.6.

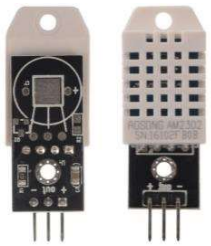

Figure 2.6 DHT 22 Temperature and Humidity Sensor

2.1.1.4 ESP32

The ESP32 is a powerful microcontroller developed by Espressif Systems. It is based on the Xtensa LX6 processor and features dual-core processing, Wi-Fi, Bluetooth, and a wide range of peripherals. The ESP32 is designed for low-power applications and has a range of advanced features that make it an ideal choice for IoT (Internet of Things) projects.

Some of the key features of the ESP32 microcontroller include:

- Dual-core processing with up to 240 MHz clock speed
- Wi-Fi connectivity with support for both 2.4 GHz and 5 GHz bands
- Bluetooth Low Energy (BLE) connectivity
- GPIO (General Purpose Input/Output) pins for interfacing with external devices
- SPI, I2C, UART, and other communication interfaces
- Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs)
- Support for various operating systems, including FreeRTOS and Zephyr

The ESP32 has gained popularity among developers due to its low cost, versatility, and ease of use. It is widely used in various IoT applications such as home automation, smart agriculture, industrial automation, and many more. The ESP 32 is shown in Figure 2.7 and the pin-diagram is shown in Figure 2.8.

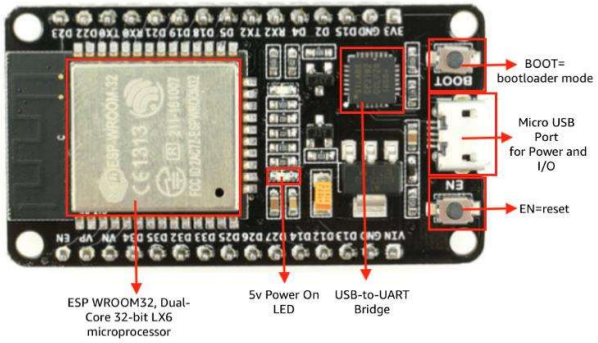

Figure 2.7 DOIT ESP32 DEVKIT V1 Diagram

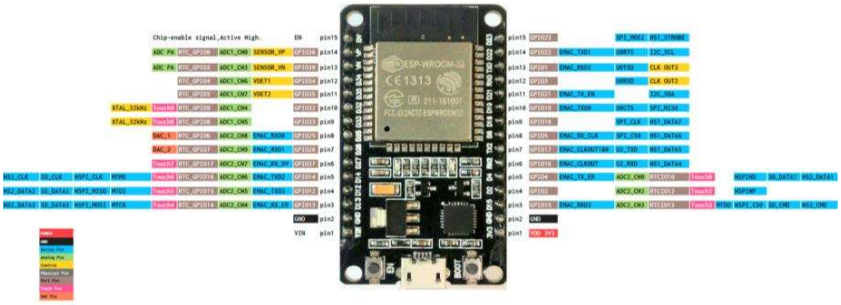

Figure 2.8 DOIT ESP32 DEVKIT V1 Pin Out Diagram

2.1.1.5 Arduino Uno

A popular microcontroller board known as the Arduino UNO was created by the Italian firm Arduino and is based on the ATmega328P processor. Since its 2010 release, it has developed to be among the microcontroller boards that professionals, academics, and beginners alike most often use. The Arduino UNO contains a 16 MHz quartz crystal, six analog inputs, 14 digital input/output pins, a USB port, a power connector, an ICSP header, and a reset button. A USB connection or an external power source may power it. The Arduino IDE (Integrated Development Environment), an open-source software platform that offers a straightforward and user-friendly interface for creating, developing, and uploading code to the board, may be used to program the board. There are several uses for the Arduino UNO, ranging from basic robotics projects to controlling LED circuits. Beginners and expert developers choose it due to its accessibility and simplicity of usage. The Arduino UNO pin diagram is shown in Figure 2.9.

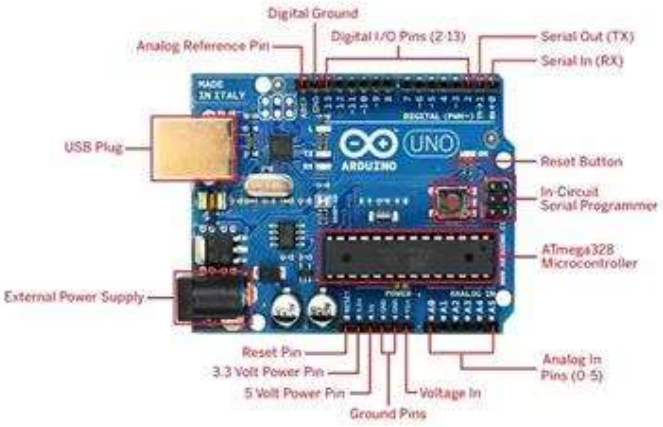

Figure 2.9 Arduino UNO pin diagram

2.1.1.6 Regulator

An electrical or electronic circuit's voltage or current level is controlled and maintained constant by a regulator circuit. This is important because many circuit components need a precise voltage or current level to operate correctly, and changes in these values may harm components or affect performance. A regulator circuit may be utilised to adjust the voltage level according to the needs of the circuit's components. Additionally, it could protect the circuit from short circuits, over-voltages, and under-voltages. It can assist in reducing interference and noise in a circuit, enhancing the functionality of sensitive components. A regulator circuit is essential for ensuring an electrical or electronic circuit is stable and reliable. The regulator circuit diagram is shown in Figure 2.10.

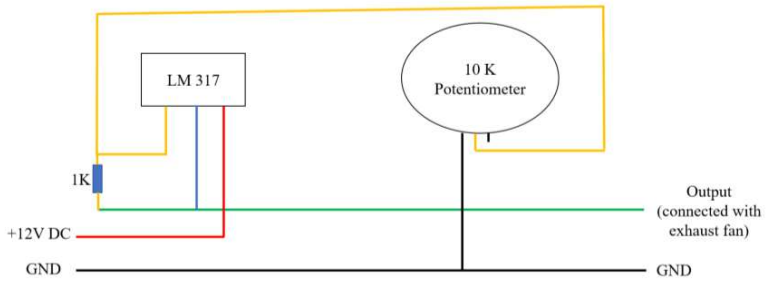

Figure 2.10 The regulator circuit diagram

2.1.1.7 DC-DC Buck Converter

A high-efficiency electrical circuit, a DC-DC buck converter, transforms higher voltages into lower voltage levels. It operates according to the pulse-width modulation (PWM) principle, which involves rapidly using a transistor switch to turn on and off the input voltage. Energy is stored in an inductor during the switch's "on" period and released to the output load during the switch's "off" period. The output voltage may be adjusted by varying the duty cycle of the PWM signal. The DC-to-DC Buck Converter is shown in Figure 2.11.

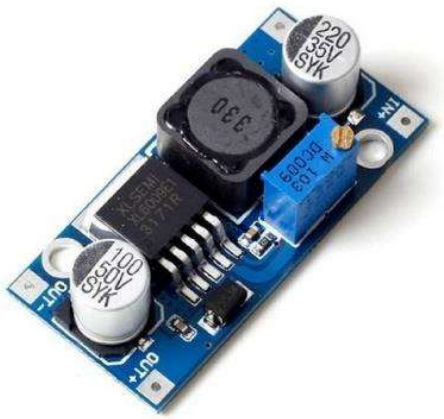

Figure 2.11DC-to-DC Buck Converter

2.1.1.8 Voltage-Divider Circuit

A voltage divider circuit is a simple circuit that divides a voltage into smaller fractions. It typically comprises two resistors in series, where the output voltage is taken from the connection between the two resistors. The voltage divider circuit works on the principle that the voltage drop across each resistor is proportional to the resistance of that resistor. The output voltage is determined by the ratio of the resistance values of the two resistors.

$$Vout = Vin * (R2 / (R1 + R2)) \quad (3)$$

Where V(esp32)/Vout = output voltage, Vcc/Vin= input voltage, R1= resistance of the first resistance and R2 = resistance of the second resistor

The voltage divider circuit is shown in Figure 2.12.

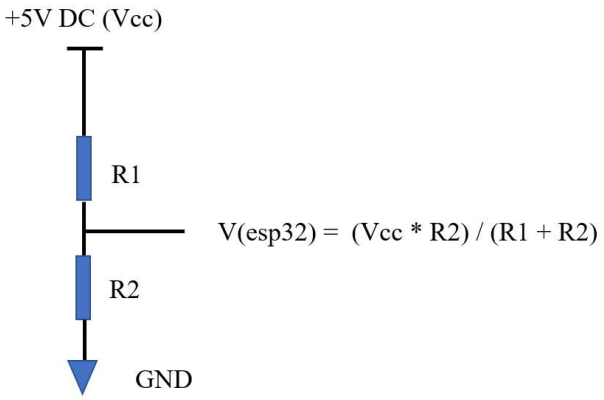

Figure 2.12 Voltage divider circuit

2.1.1.9 Gas Chamber

A gas chamber is typically a sealed container where the gas concentration can be precisely controlled and monitored. Gas chambers can be used to test and calibrate various gas sensors, including those used for environmental monitoring, industrial safety, and medical applications. Gas chambers for sensor experimentation can be used to test the sensitivity and specificity of gas sensors under controlled conditions and evaluate their response time, stability, and durability.

The chamber's internal dimensions were 30 cm × 21 cm × 12 cm (L × W × H), with a total interior volume of 7.560 L (7650 cm^3) and are made of 6 mm thick glass and needful connections were made according to the circuit. Inlet and outlet holes have been created on two opposite side walls of the chamber. For proper flow of VOCs, gases and odors, an exhaust fan of 120 mm diameter rated at 12V, 0.25A has been placed at the chamber's outlet hole. This fan is directly powered by the primary 12V,18Ah SMF battery. This exhaust fan's speed is controlled using a transistor LM137-based voltage regulator. The gas chamber is shown in Figure 2.13.

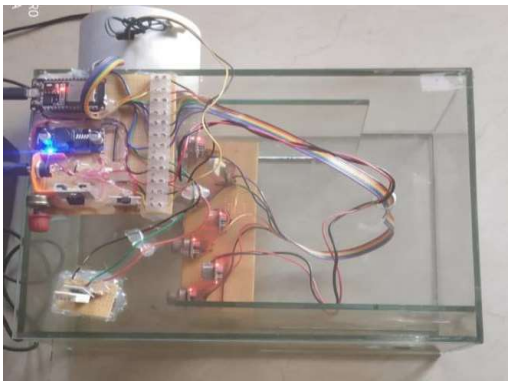

Figure 2.13 The Gas Chamber

2.1.2 Gas Sensor Array Responses

A gas sensor array is a collection of gas sensors designed to work together to detect and identify different types of gases. The sensors in a gas sensor array are typically made up of materials sensitive to different types of gases. When a gas comes into contact with the sensor material, it causes a change in the electrical conductivity or other physical properties of the material, which can be measured and used to detect the presence of the gas. Gas sensor arrays are used to detect the presence of hazardous gases, and medical applications monitor the breath of patients, as the composition of exhaled air can provide information about the patient's health.

The response of a gas sensor array refers to the change in the sensor output signal in response to exposure to a particular gas or a mixture of gases. The response of each sensor in the array is typically measured and analysed to identify the types of gases present and their concentrations. The gas sensor array is shown in Figure 2.14.

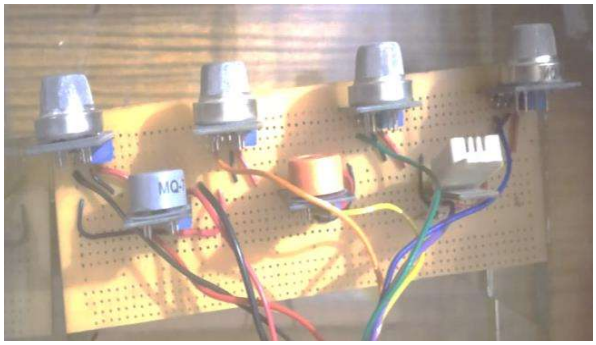

Figure 2.14 Gas Sensor Array

Gas sensor arrays can have various types of responses, depending on the design of the sensors and the nature of the gases being detected. Some common types of responses include:

- Selective Response: A selective response occurs when a specific sensor in the array responds to a particular gas but does not respond to other gases. This type of response helps identify the presence of specific gases and can be used for gas-specific detection.
- Cross-Selective Response: A cross-selective response occurs when a specific sensor responds to more than one gas, but the response to one gas is significantly more robust than the response to the other. This type of response can be used to identify specific gases in the presence of other gases.
- Non-Selective Response: A non-selective response occurs when a sensor in the array responds to multiple gases without distinguishing between them. This type of response helps detect the presence of gases but may not be able to identify the specific types of gases present.
- Dynamic Response: A dynamic response occurs when the sensor response changes over time in response to the presence of the gas. This type of response helps monitor changes in gas concentrations over time.

A typical sensor array response as obtained during the experiment has been shown in Figure 2.15.

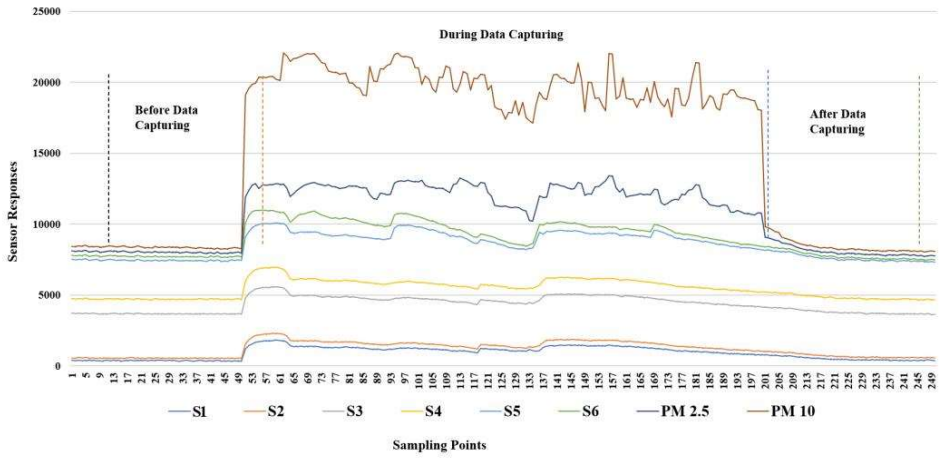

Figure 2.15. A typical sensor array response.

2.1.2.1 Signature Patterns from Non-selective Gas Sensors

Non-selective gas sensors typically detect multiple gases and cannot distinguish between them based on their chemical composition alone. However, by analysing the sensor response patterns, it may be possible to identify certain signature patterns that are characteristics of specific gases or gas mixtures.

Some common signature patterns from non-selective gas sensors include:

- Cross-sensitivity: non-selective gas sensors often exhibit cross-sensitivity, meaning they respond to multiple gases differently. For example, a sensor may be more sensitive to one gas than another or respond more strongly to a specific combination of gases.
- Baseline shift: When exposed to a gas, the baseline reading of a non-selective gas sensor may shift up or down depending on the gas concentration and the sensor's sensitivity to that gas.
- Response time: Different gases may cause the sensor to respond at different rates. For example, some gases may be detected immediately, while others may take longer to register on the sensor.
- Recovery time: After exposure to a gas, a non-selective sensor may require some time to recover and return to its baseline reading. The recovery time may depend on factors such as the gas concentration, the sensor's sensitivity, and the environmental conditions.
- Sensor drift: Over time, the sensor response may drift due to changes in the sensor's characteristics or environmental conditions. This can make it more challenging to identify signature patterns accurately and may require calibration or sensor replacement.

By analysing these signature patterns, it may be possible to develop algorithms or machine learning models to accurately identify specific gases or gas mixtures based on the sensor readings.

2.2 Experimental Data Collection and Storing Process (Data Acquisition System)

Data acquisition is the process of collecting and storing experimental data for analysis. The data acquisition system (DAS) is a critical component of any experiment, as it ensures that data is collected accurately and reliably. The data acquisition process:

- Identify the data type to be collected: Before beginning any experiment, it is essential to determine what type of data will be collected. This will inform the types of sensors, probes, or other devices needed to collect the data.
- Choose the appropriate data collection hardware: Once you know what type of data you need to collect, you can choose the appropriate hardware for the job. This may include sensors, transducers, data loggers, or other devices.
- Set up the data acquisition system: Once you have the hardware in hand, you can set up the data acquisition system. This typically involves connecting the sensors or devices to a data acquisition unit, which may be a stand-alone or computer-based system.
- Configure the data acquisition software: Next, you must configure it to ensure it is set up to collect the data you need. This may include setting up data channels, sampling rates, and triggering parameters.
- Calibrate the sensors: Before collecting any data, it is essential to calibrate the sensors to ensure that they provide accurate and reliable data. This typically involves measuring known quantities and adjusting the sensor output to match the expected values.
- Collect the data: With the data acquisition system set up and calibrated, you can begin collecting data. This may involve running an experiment or simply collecting data over some time.
- Store the data: As collected, it should be stored securely to ensure it is not lost or corrupted. This may involve saving data to a computer, external hard drive, or cloud-based storage solution.
- Analyse the data: Once collected and stored, it can be analysed to draw meaningful conclusions. This may involve using statistical analysis, data visualisation tools, or other techniques to identify patterns, trends, or relationships in the data.

2.2.1 Cyber-physical System (CPS)

A networked system incorporating physical sensors for gas detection with computational and communication technologies is called a cyber-physical system (CPS) for gas sensing. In order to provide real-time monitoring, analysis, and response to gas-related events or circumstances, it integrates the physical world of gas sensing with the virtual world of data processing and control. Physical sensors are used in a CPS for gas sensing to detect and measure several gas properties, including concentration, temperature, pressure, or humidity. These sensors gather real-time information about the gas environment and send it to a centralized computer system.

A CPS's computer system takes sensor data, analyzes it using algorithms or models, and provides insightful findings or information that may be put to use. It is capable of identifying hazardous circumstances like gas leaks or unusual gas concentrations. Data fusion methods, which combine data from several sensors to improve accuracy and dependability, may be used in this investigation.

The real-time sharing of sensor data and analytical findings throughout the network is made possible by the communication component of a CPS. It makes it easier to monitor and manage the gas sensing system remotely, allowing quick reactions to gas-related incidents or crises. Numerous fields, including industrial settings, smart buildings, environmental monitoring, and safety systems, have used CPS for gas detection. A CPS for gas sensing, for instance, may be used in an industrial environment to continually monitor the facility's gas levels, find leaks, gases/VOCs/hazardous detection and monitoring, fire hazard detection, health hazard detection and automatically initiate shutdown processes or alerts to avoid mishaps. A simplified Schematic Diagram of the Cyber-physical System has been depicted in the Figure 2.16.

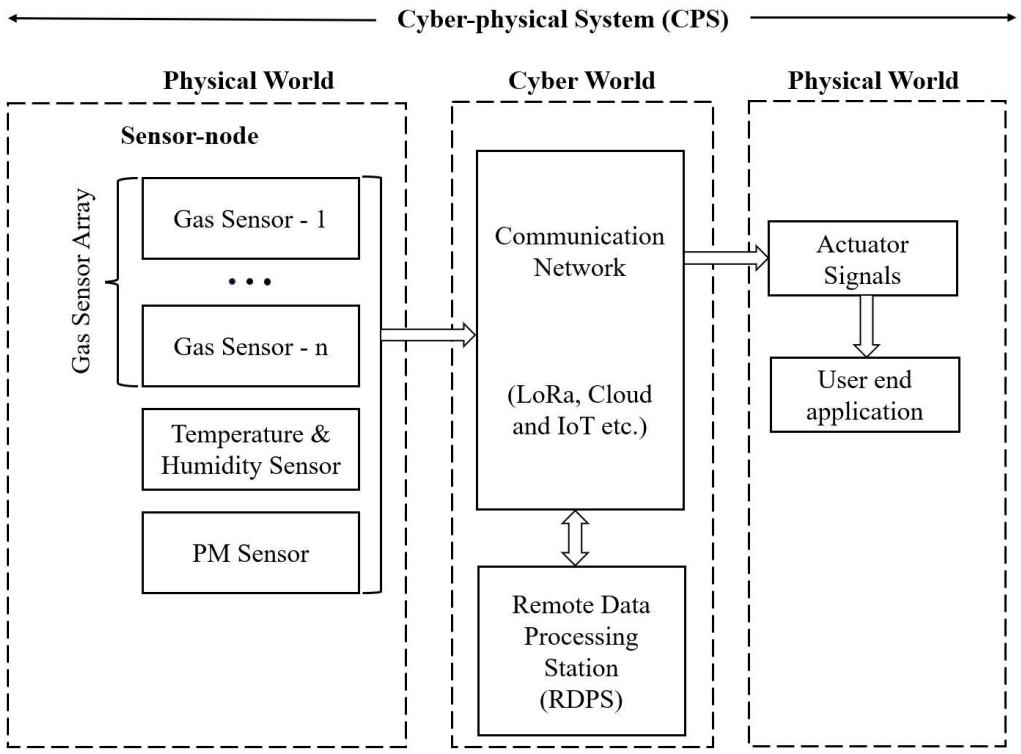

Figure 2.16 A simplified Schematic Diagram of the Cyber-physical System (CPS)

2.2.2 Amazon Web Services (AWS)

AWS is a cloud computing platform provided by Amazon that offers a wide range of services, including computing power, storage, databases, analytics, machine learning, and more. AWS provides on-demand access to computing resources that can be scaled up or down according to the user's needs, allowing businesses to efficiently and cost-effectively run their applications and services in the cloud.

AWS offers a variety of services, including:

- Compute services: Amazon Elastic Compute Cloud (EC2), Amazon Elastic Container Services (ECS), and AWS Lambda are examples of AWS compute services.
- Storage services: Amazon Simples Storage Services (S3), Amazon Elastic Block Store (EBS), and Amazon Glacier are examples of AWS storage services.
- Database services: Amazon Relational Database Services (RDS), Amazon DynamoDB, and Amazon Redshift are examples of AWS database services.

- Networking services: Amazon Virtual Private Cloud (VPC), Amazon Route 53, and AWS Direct Connect are examples of AWS networking services.
- Analytics services: Amazon Kinesis, Amazon EMR, and Amazon QuickSight are examples of AWS analytics services.
- Machine learning services: Amazon Sagemaker, Amazon Recognition, and Amazon Comprehend are examples of AWS machine learning services.

2.2.3 Visual Studio Code (VS code)

Visual Studio Code is a free and open-source code editor developed by Microsoft. It is designed for developers working on various programming languages and platforms such as JavaScript, Node.js, TypeScript, and Python. Some of its key features include:

- IntelliSense: VS-Code includes a powerful IntelliSense engine that suggests code completion, syntax highlighting, and code formatting. It also provides online documentation for functions and libraries.
- Git integration: VS-Code has built-in support for Git, allowing developers to manage their source code directly within the editor.
- Extensions: It supports a wide range of extensions that add functionality to the editor. These extensions can be used to add support for new programming languages, debuggers, and more.
- Debugging: It includes a built-in debugger that allows developers to debug their code directly within the editor.
- Integrated terminal: It includes an integrated terminal that allows developers to execute commands and run scripts directly within the editor.

2.2.4 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a software tool used to write, compile, and upload code to an Arduino board. It provides a user-friendly interface that allows users to program their Arduino boards without using the command line. The IDE is free and open-source.

The Arduino IDE supports several programming languages, including C and C++, and it includes a code editor with features like syntax highlighting, auto-completion and error highlighting. It also has a built-in serial monitor that allows users to communicate with their Arduino boards and view the output of their programs in real time.

The IDE is compatible with a wide range of Arduino boards and shields, and it includes an extensive library of pre-written code that users can use as a starting point for their projects. Additionally, the IDE is highly customisable, and users can install plugins and customise the settings to suit their needs.

2.2.5 LoRa (Long Range)

LoRa is a wireless communication technology for long-range, low-power, and low-data-rate applications. It uses a spread-spectrum modulation technique called Chirp Spread Spectrum (CSS) to transmit data over long distances while consuming little power.

It operates in the unlicensed radio spectrum, meaning anyone can use it without a license. It typically operates in the frequency range of 868 MHz to 915 MHz in most countries and 433 MHz in some countries. The low frequencies used by LoRa enable the signals to penetrate through obstacles such as walls and buildings, making it suitable for indoor and outdoor applications.

LoRa technology is commonly used in Internet of Things (IoT) applications such as smart agriculture, asset tracking, smart cities, and industrial automation. It can provide long-range connectivity with low-power consumption, which is ideal for battery-powered devices that send small amounts of data over long distances. The LoRa devices are shown in Figure 2.17.

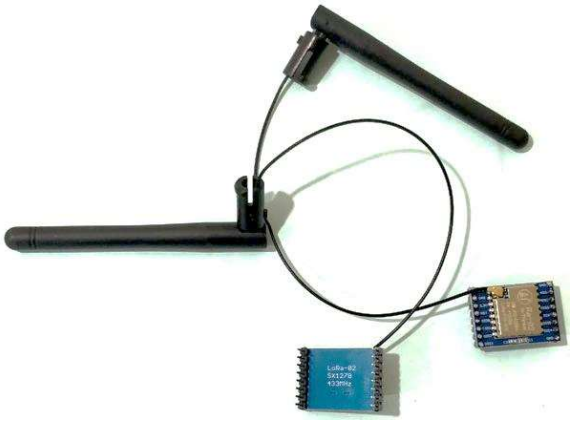

Figure 2.17 LoRa Devices

2.2.5.1 Working Principle of LoRa

The working principle of LoRa involves a combination of spread-spectrum modulation, CSS technology, and forward error correction (FEC) coding.

- Spread-spectrum modulation: LoRa uses spread-spectrum modulation to spread the signal over a large frequency band, which allows it to transmit over long distances without interference. This means the data signal is spread out over a wide range of frequencies instead of concentrated in a narrow band.
- CSS: LoRa uses a specific spread-spectrum modulation called CSS. In CSS, the signal frequency linearly changes over time, producing a "chirp" waveform. The chirp waveform allows LoRa signals to have a very narrow bandwidth while still being able to travel long distances.
- FEC coding: To improve the reliability of the transmission, LoRa uses FEC coding. FEC coding adds extra bits to the message being transmitted, which can be used to correct errors in the message when it is received. This helps ensure the message is received correctly, even with noise or interference.
- Gateway: LoRa devices transmit their signals to a gateway, which bridges the LoRa network and the internet. The gateway receives the LoRa signal and forwards it to the LoRa network server, which can then process the data and send it to the appropriate application or service.

2.3 The Datasets

As described in the previous sections, we have developed five different sensor prototypes for their applications in five different scenarios:

2.3.1 The Dataset I (Smart Home)

In this experiment, the most common 17 types of VOCs, gases and odors have been considered. They are (i) released from various carcinogenic burning of food items (milk, wheat-bread, pigeon peas), (ii) smoke of different brands of incense sticks (Indian Juniper, Zed-Black, Millennium, Heritage, first-Spray), and (iii) miscellaneous aromas and gases emanated from house cleaning disinfectant (phenyl, Acid, Harpic), Camphor, mosquito repellent brands (All-out, Mosquito-coil), (Liquefied petroleum gas) LPG, Cigarette. The raw sensor array responses consisted of 2550 steady-state responses, where (145 × 17 = 2465) samples were used to train the classifier. The remaining (17 × 5 = 85) samples were separated out for testing purposes. The raw sensor responses of Dataset-I are shown in Table 2.4.

Table 2.4 Sensor responses of Dataset-I

MQ 3	MQ 4	MQ 5	MQ 7	MQ 8	MQ 135	Class
469	239	3185	1284	2820	35	1
320	590	3625	1507	4000	624	2
641	876	3945	1737	4000	3003	3
…..	….	….	…..	…..	….	….
487	550	3377	1344	3723	3120	15
774	613	3457	1155	3062	963	16
455	669	3824	2021	4070	2354	17

(2550 × 7)

2.4 The Dataset II (Fire Hazard Detection and Mapping)

Based on the kind of material being burnt and as per the National Fire Protection Association (NFPA) standards, fire has been categorised into six classes, viz., Class A (combustible solids – paper, cloth, wood, etc.), Class B (flammable liquids – paints, kerosene, diesel, etc.), Class C (electrical components – PVC, rubber, electrical wires,

etc.), Class D (fats and cooking oils – refine, mustard oil, coconut oil, etc.), Class K (combustible metals - Magnesium), and Class F (flammable gases – LPG, CNG, etc.)

Regarding the samples belonging to the six classes of fire smokes, a total of 2400 samples were captured for the sixteen considered materials. The dataset contains 450 samples of class-A (cloth, paper and wood), 600 samples belonging to Class B (paints, grease, kerosene and diesel), 450 samples belonging to Class C (Rubber, PVC, Electrical-wire), 600 samples of Class D (butter, mustard oil, refined oil and coconut oil), 150 samples of Class K (Magnesium) and 150 samples of Class F (LPG).

The captured dataset was then segregated into two sets, i.e., training and testing datasets. Accordingly, the training dataset consisted of 145×3 = 435 samples for class A, 145×4 = 580 samples for class B, 145×3 = 435 samples for class C, 145×4 = 580 samples for class D and 145 samples each for class K and class F, respectively. Further, for testing purposes, we have used 15 samples for class A, 20 samples for class B, 15 samples for class C, 20 samples for class D and five samples each for class K and class F, respectively, called the testing dataset. The raw sensor responses of Dataset-II are shown in Table 2.5.

Table 2.5. Sensor Responses of Dataset-II

MQ 3	MQ 4	MQ 5	MQ 7	MQ 8	MQ 135	PM 2.5	PM 10	Class
960	272	3126	1078	3218	286	2378	7573	Class A
1854	239	2332	2223	3719	2567	1213	5192	Class B
1029	171	3175	1089	2995	960	1622	7432	Class C
1347	233	3169	1259	3902	1856	1350	4861	Class D
1635	1653	3730	1946	2779	2388	423	451	Class K
1476	436	1047	542	128	279	1815	5656	Class F

(1950 × 8)

2.5 The Dataset III (Pollution Hazard-Using LoRa Module)

The samples belong to the six classes of VOCs, gases, and smoke released by burning different pollutants in indoor ambient conditions. The dataset consists of 300 samples of each class: ambient air, tobacco, paints, carpet, incense sticks, and alcohol. The captured dataset (300 × 6 = 1800 samples) was then segregated into training and testing datasets. The training dataset consisted of 295 × 6 = 1770 samples and 5 × 6 = 30

samples for testing purposes. Finally, the testing dataset was separated from the training or validation of the classifiers at any stage. The raw sensor responses of Dataset-III are shown in Table 2.6.

Table 2.6. Sensor responses of Dataset-III

MQ 2	MQ 3	MQ 5	MQ 6	MQ 7	MQ 8	MQ 135	Class
563	515	370	333	631	420	390	Ambient Air
735	529	393	385	607	555	431	Tobacco
545	351	332	371	568	548	278	Paints
542	425	380	357	483	340	414	Carpet
1808	816	803	1890	1696	87	903	Incense
2287	1493	893	2143	2252	93	1901	Alcohol

(1800 × 7)

2.6 The Dataset IV (Disinfectants Detection and Monitoring)

In this experiment, we used six different types of materials. These are hydrogen peroxide (surface-cleaning, cleaning cuts, hair dye, etc.), sanitiser and alcohol (hand-cleaning), phenyl, acid, and harpic (floor and bathroom cleaning). During this experimental period, 900 samples (150 samples × 6) were captured at the sampling rate of 15 samples per minute. In this dataset, 870 samples (145 samples × 6) were for training, and 30 samples (5 samples × 6) were for testing. All 30 samples of the testing dataset have not been used for training and validation purposes. The raw sensor responses of Dataset IV are shown in Table 2.7.

Table 2.7. Sensor responses of Dataset-IV

MQ 3	MQ 7	MQ 135	MQ 136	MQ 137	MQ 138	Class
469	239	3185	1284	2820	35	Hydrogen peroxide
417	226	3248	1111	3844	2111	Sanitizer
385	241	3227	1141	3403	2190	Alcohol
290	291	3264	1112	3435	270	Phenyl
500	524	999	1513	1103	2664	Acid
1956	2350	2112	722	1679	1661	Alcohol

(900 × 6)

2.7 The Dataset V (Health Hazard - Diabetes detection using AWS)

During this period, a total of 1800 samples were captured at the sampling rate of 10 samples per minute. In regard of the samples belong to the three classes (before and after breakfast each) of BGL. The dataset contains 600 samples for low BGL, 600 for normal BGL, and 600 for high BGL in both situations. The captured dataset was then segregated into training and testing datasets. The training dataset consisted of 590 × 3 = 1970 samples and 10 × 3 = 30 samples for testing purposes. The testing dataset was separated from the training or validation of the classifiers at any stage. The raw sensor responses of Dataset-V are shown in Table 2.8 and Table 2.9.

Table 2.8. Sensor responses of Dataset-V(a) (Before Breakfast)

MQ 2	MQ 3	MQ 5	MQ 6	MQ 7	MQ 8	MQ 135	Class
793	537	483	453	652	695	504	High BGL
724	529	403	395	627	592	442	Low BGL
636	360	396	412	575	578	298	Normal BGL

(900 × 7)

Table 2.9. Sensor responses of Dataset-V(b) (After Breakfast)

MQ 2	MQ 3	MQ 5	MQ 6	MQ 7	MQ 8	MQ 135	Class
786	532	457	446	578	678	506	High BGL
791	531	496	485	682	736	502	Low BGL
584	351	344	379	570	568	285	Normal BGL

(900 × 7)

2.8 Data Pre-Processing Techniques

Data pre-processing is a crucial step in data analysis that involves transforming raw data into a suitable format for further analysis. Data pre-processing techniques are used to clean, transform, and prepare data for analysis to be used effectively in machine learning models, statistical analyses, and other data-driven applications.

Here are some reasons why data pre-processing is essential:

- Handling missing data: Missing data is a common problem in datasets that can affect the accuracy of any analysis. Imputation can help fill in missing values in a dataset.
- Removing outliers: Outliers are extreme values in a dataset that can skew the results of any analysis. Filtering and trimming can help remove outliers from the data.
- Feature scaling: The features' scale can significantly impact the results in many machine learning algorithms. Normalisation and standardisation can help scale the features to have a similar range.
- Space transformation:
- Encoding categorical variables: Machine learning algorithms require numerical data, and categorical variables must be converted into numerical values for analysis. One-hot encoding can be used to convert categorical variables into numerical data.
- Dimensionality reduction: large datasets with many features can be challenging to analyse. Principal Component Analysis (PCA), Independent Component Analysis (ICA), Standardised Linear Component Analysis (SLDA), Kernel Principal Component Analysis (KPCA), Quantile Principal Component Analysis (QPCA), Standardised Principal Component Analysis (SPCA) and feature selection can reduce the number of features in a dataset, making it easier to analyse.

2.8.1 Independent Component Analysis (ICA)

Independent component analysis is known as a blind source separation technique. It attempts to extract underlying signals that, when combined. The original inputs are linearly transformed in ICA into mutually statistically independent features.

1. Centring

Subtracting the mean from all samples.

$$D = X - \mu = (x_1 - \mu, x_2 - \mu, \ldots., x_n - \mu) \quad (4)$$

Where D = mixture signals after centring,

$$\mu \in R^{1 \times N} \quad (5)$$

Where μ = mean of all mixture data

2. Whitening data

Transforming signals into uncorrelated signals and then rescaling each signal with unit variance.

2.1 Decorrelation

Make each signal uncorrelated from the others. In ICA, the PCA technique can be used for decorrelating signals.

- Calculate covariance matrix,

$$\sum = E[DD^T] \quad (6)$$

Where D= centered data

$$V\sum = \lambda v \quad (7)$$

Where V = eigenvectors and λ=eigenvalues

- Project onto the calculated PCA space as follows:

$$U = VD \quad (8)$$

Where U = unit length

2.2 Scaling

Scale each decorrelated signal to be with a unit variance as follows,

$$Z = \lambda^{-1/2}U = \lambda^{-1/2}VD \quad (9)$$

Where Z= whitened or sphered data

2.8.2 Standardised Linear Discriminant Analysis (SLDA)

SLDA is a statistical method that can be used for dimensionality reduction. It is a linear transformation process on the input data, which transforms the raw sensor responses into new components to maximise the separation between different classes.

The process of the SLDA space transformation method has been described in the following steps:

- Calculate the mean vector, mi (i=1,2,3…) of each class of a dataset
- Scatter matrix within the class

$$S_W=\sum_{i=1}^{c} S_i \tag{10}$$

Where S_W= data points within each class deviate from their respective class

$$S_i = \sum_{x\in D_i}^{n}(x - m_i)(x - m_i)^T \tag{11}$$

Where S_i = scatter matrix of each class, x = data point, mi = mean vector and T= transpose matrix

$$m_i = \frac{1}{n_i}\sum_{x\,\in\, x_i}^{n} x_i \tag{12}$$

- Calculate the covariance matrix by adding the scaling factor (1/(N-1)) to the within-class scatter matrix,

$$\sum_i = \frac{1}{N_i-1}\sum_{x\in D_i}^{n}(x - m_i)(x - m_i)^T \tag{13}$$

$$\text{And } S_W = \sum_{i=1}^{c}(N_i - 1)\sum_i \tag{14}$$

Where N_i= sample size of the VOC class (here = 300 × 6), Now we can drop (Ni -1) because all classes have equal sample size

- Scatter matrix between each class (S_B)

$$S_B = \sum_{i=1}^{c} N_i(m_i - m)(m_i - m)^T \tag{15}$$

Where m = overall mean, mi= sample mean and Ni= sample size of the respective class

- Compute the eigenvectors and eigenvalues

$$A = S_W^{-1}S_B \tag{16}$$

$$AV = \lambda V \tag{17}$$

Where λ = eigenvalue and V = eigen vector of same eigen value

- Project the data onto the new subspace

$$Y = X \times W \tag{18}$$

Where X = n dimensions matrix representation the n samples and Y = Transmitted n × k dimensional samples in the new subspace

2.8.3 Kernel Principal Component Analysis (KPCA) [140]

Let data mapped into feature space, $\phi(x_1),\ldots., \phi(x_l)$, is centered,

$$\sum_{k=1}^{l} \phi(x_k) = 0 \tag{19}$$

Calculate PCA for the covariance matrix,

$$C = \frac{1}{l}\sum_{j=1}^{l} \phi(x_j)\phi(x_j)^T \tag{20}$$

Further, calculate Eigen values $\lambda \geq 0$ and Eigenvectors $V \in F\setminus \{0\}$,

$$\lambda V = CV \tag{21}$$

V lie in the span of $\phi(x_1),\ldots., \phi(x_l)$,

This implies that we can consider the equivalent system,

$$\lambda(\phi(x_1).V) = (\phi(x_k).CV) \text{ for all } k = 1,\ldots., l \tag{22}$$

and that there exist coefficients $\alpha_1,\ldots, \alpha_l$ such that,

$$V = \sum_{i=1}^{l} \alpha_i \, \phi(x_1) \tag{23}$$

Substituting equation (3) and (5) into (4) and defining an l ẋ l matrix K by,

$$K_{ij} = \phi(x_i) \,.\, \phi(x_j) \tag{24}$$

2.8.4 Standardised Principal Component Analysis (SPCA)

The raw sensor response dataset is first standardised for zero mean and unit variance. Then its covariance matrix is calculated to obtain its eigenvectors and eigenvalues. The highest eigenvalues and their associated eigenvector project the first principal component (PC), along which the transformed dataset shows the highest variance. Similarly, other principal components are determined by taking the next higher eigenvalues and eigenvectors. The information contained in each PC is obtained by normalising the eigenvalues and obtaining its percentage. The process of

transforming the raw sensor array responses into SPCA analysis domain has been depicted in Figure 2.18.

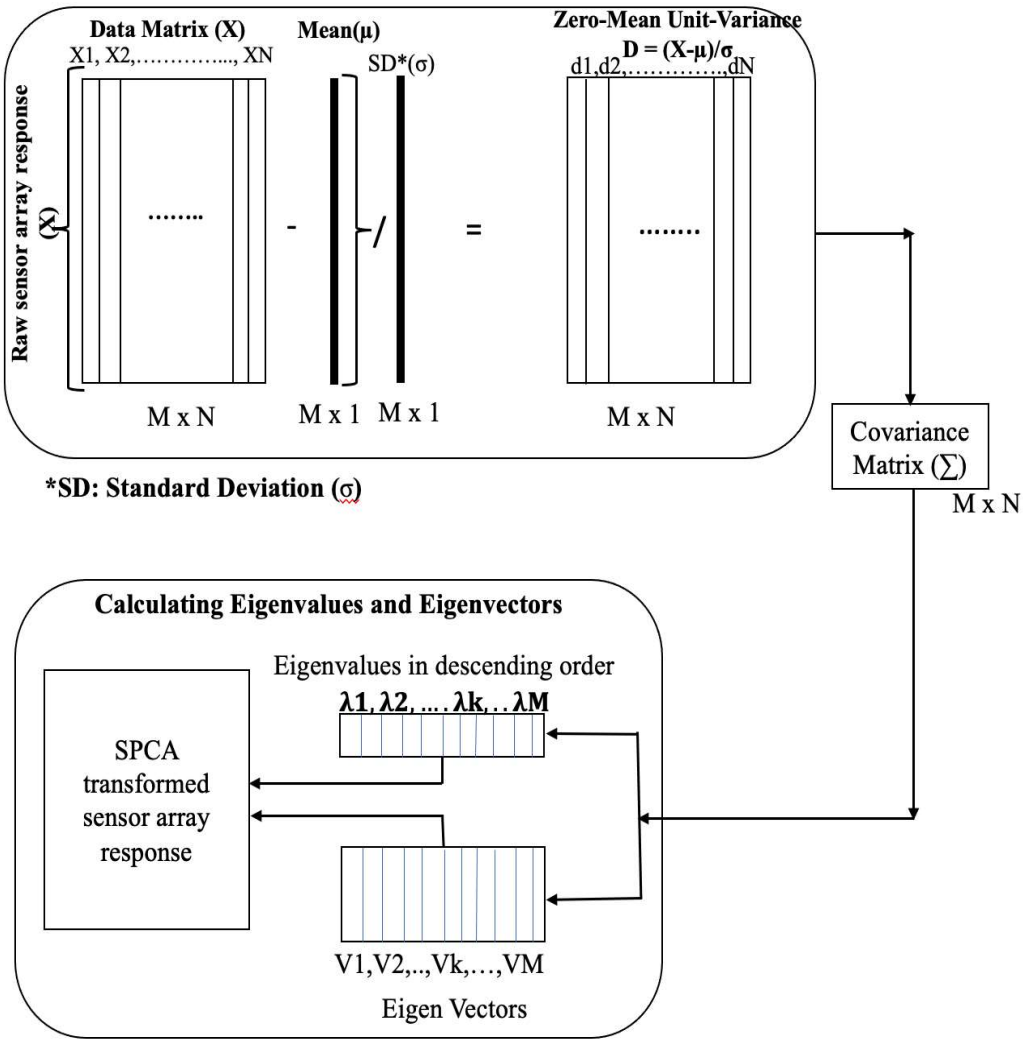

Figure 2.18. Graphical representation of the process of transformation of raw sensor responses into SPCA domain

2.9 The concept of Analysis Space Transformation and Cluster formation

Analysis Space Transformation is transforming the data representation from one space to another. This technique is used to simplify the complexity of the data and make it more suitable for analysis. It involves changing the original features of the data into a new set of features that are easier to analyse. This can be done through various techniques, such as ICA, LDA, KPCA, PCA and SPCA [47]-[54].

Cluster formation is a process of grouping data points based on their similarities. In this technique, data points similar to each other are placed in the same cluster, while those dissimilar are placed in different clusters. This helps to identify patterns and structures within the data.

Analysis Space transformation and cluster formation are powerful data analysis tools, helping simplify and structure complex data and making it easier to identify patterns and insights.

2.9.1 K-Fold Cross-Validation Process

K-fold cross-validation is a statistical method used to estimate the performance of a machine learning algorithm on an unseen dataset. K-fold cross-validation aims to split the dataset into K equal parts (or "folds"), train the algorithm on K-1 folds, and evaluate its performance on the remaining fold. This process is repeated K times, with each fold serving as the test set once, and the average performance across all K folds is computed as the final performance estimate. It can help reduce overfitting and improve the model's generalisation performance.

Steps for K-fold cross-validation:

- Split the dataset into K equal parts or folds.
- Train the machine learning model on K-1 folds, and use the remaining fold as the test set.
- Evaluate the performance of the model on the test set.
- Repeat steps 2 and 3 for all K-folds.
- Compute the average performance across all K folds.

2.9.2 Introduction to different Classification Algorithms

We have analysis using following classification algorithms: KNN, LR, SGD, DT, NB, SVM, RDA, RF, Adaboost, XGBoost and MLP [47]-[64].

2.9.2.1 K-nearest neighbors (KNN)

KNN is a non-parametric algorithm that uses the distance metric to find the k-nearest data points to a given input data point. Define the number of neighbors (k) and

calculate the distance between the input data point and all the data points in the dataset using a distance metric such as Euclidean distance or Manhattan distance. Select the k-nearest data points to the input data point based on the calculated distances. Once the distances between the input data point and all the data points in the dataset are calculated, we can sort the distances in ascending order and select the k-nearest data points to the input data point.

2.9.2.2 Logistic Regression (LR)

LR is a statistical method used to analyze data with one or more independent variables that determine the outcome of a binary or categorical dependent variable. It is widely used in machine learning, data science, and statistical modelling for classification tasks. The logistic regression algorithm uses a logistic function or sigmoid function to transform the input data into a value between 0 and 1, representing the probability of the outcome variable taking a specific value.

The equation for logistic regression is as follows:

$$p(x) = 1/ (1+e\text{^} (-(\beta 0+\beta 1x1+\beta 2x2+...+\beta nxn))) \quad (25)$$

where $p(x)$ is the probability of the outcome variable taking a specific value, e is the base of the natural logarithm, $\beta 0$ is the intercept, and $\beta 1, \beta 2, \dots, \beta n$ are the coefficients of the independent variables $x1, x2, \dots, xn$.

The logistic function produces an S-shaped curve, which is used to classify the input data into two or more classes. The threshold value is set at 0.5, which means that if the probability of the outcome variable taking a specific value is greater than 0.5, it is classified as 1, and if it is less than 0.5, it is classified as 0. The logistic regression algorithm uses a maximum likelihood estimation method to find the optimal values of the coefficients that maximize the likelihood of the observed data.

This is done by minimizing the cost function, which is the negative log-likelihood of the data:

$$J(\beta) = -[\sum (yi \log(p(xi))+(1-yi)\log(1-p(xi)))] \quad (26)$$

where J(β) is the cost function, yi is the observed value of the outcome variable for the i-th sample, and $p(xi)$ is the predicted probability of the outcome variable taking a specific value for the i-th sample.

2.9.2.3 Stochastic Gradient Descent (SGD)

SGD is an optimization algorithm commonly used in machine learning for finding the optimal parameters of a model. In traditional gradient descent, the model parameters are updated by computing the gradient of the loss function with respect to the parameters for the entire training dataset. This can be computationally expensive and slow for large datasets. SGD is a variation of gradient descent that randomly selects a single data point from the training dataset at each iteration and updates the parameters based on the gradient of the loss function with respect to that particular data point. This makes the algorithm more computationally efficient and faster. The equation for updating the parameters using SGD can be represented as:

$$\theta = \theta - \alpha * \nabla J (\theta, xi, yi) \quad (27)$$

Where, θ is the vector of model parameters α (alpha) is the learning rate,

which determines the step size for the parameter updates ∇J (θ, xi, yi) is the gradient of the loss function with respect to the parameters evaluated at the selected data point (xi, yi) The algorithm repeats this process for a fixed number of iterations or until convergence is achieved, where the loss function is minimized and the model parameters are optimized.

2.9.2.4 Decision-Tree-Based Techniques

A decision tree classifier is a machine-learning algorithm for classification tasks. It is a tree-like model where each node represents a feature or attribute, and each branch represents a decision rule. The tree is constructed by recursively partitioning the data into subsets based on the most informative features until a stopping criterion is met, such as the purity of the subsets. To classify a new instance, the algorithm starts at the root node and evaluates the corresponding feature. Based on the decision rule, it moves down the tree to the next node until it reaches a leaf node, representing a class label. In this case,

the criterion used is "entropy," which measures the impurity of the nodes in the decision tree.

Entropy measures the randomness of the target variable's distribution in a node, and the algorithm minimises it. The splitter is the strategy to split a node in a decision tree. The two options are "best" and "random." In this case, the "best" splitter is used, which means the algorithm will try to find the best split among all possible features. The maximum depth is the maximum number of levels allowed in the decision tree. In this case, the maximum depth is set to 3, which means that the decision tree can have, at most, three levels of nodes. The maximum features are the maximum number of features the algorithm considers when looking for the best split. In this case, the value "auto" is used, which means that the algorithm will consider all the features. The architecture diagram of a decision tree is presented in Figure 2.19.

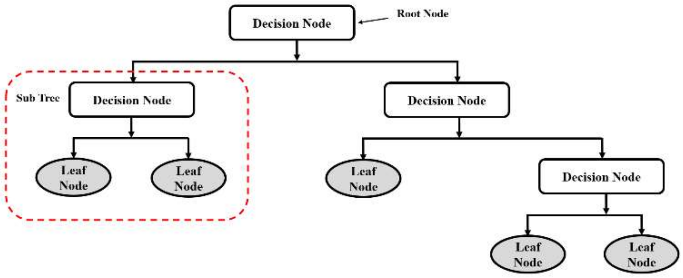

Figure 2.19. The architecture of the Decision Tree Classifier.

2.9.2.5 Naïve Bayes (NB)

NB is a probabilistic classification algorithm that is based on Bayes' theorem. It assumes that the presence or absence of a particular feature in a class is independent of the presence or absence of any other feature. This is called the "naive" assumption.

Here's how the Naive Bayes classification algorithm works:

1. Collect a set of training data. Each training data point consists of a set of features (X1, X2, ..., Xn) and a class label (y).
2. Calculate the prior probability of each class label. This is the probability of a data point belonging to each class based on the proportion of training data points that belong to that class. It can be calculated as follows:

P(y) = (number of data points with class y) / (total number of data points) (28)

3. For each feature Xi, calculate the conditional probability of that feature given each class label. This is the probability of observing feature Xi given that the data point belongs to class y. It can be calculated as follows:

P(Xi | y) = (number of data points with feature Xi and class y) / (number of data points with class y)

(29)

4. Given a new data point with features X1, X2, ..., Xn, calculate the probability of that data point belonging to each class label using Bayes' theorem:

P(y | X1, X2, ..., Xn) = (P(y) * P(X1 | y) * P(X2 | y) * ... * P(Xn | y)) / P(X1, X2, ..., Xn)

(30)

5. The class label with the highest probability is chosen as the predicted class for the new data point. Here's the equation for Naive Bayes classification algorithm:

P(y | X1, X2, ..., Xn) = (P(y) * P(X1 | y) * P(X2 | y) * ... * P(Xn | y)) / P(X1, X2, ..., Xn)

(31)

where:

- P(y) is the prior probability of class y
- P(Xi | y) is the conditional probability of feature Xi given class y
- P(X1, X2, ..., Xn) is the probability of observing features X1, X2, ..., Xn
- P(y | X1, X2, ..., Xn) is the posterior probability of class y given features X1, X2, ..., Xn

2.9.2.6 Support Vector Machine (SVM)

SVM is a popular machine learning algorithm used for classification and regression problems. SVM is a binary classification algorithm that finds a hyperplane in an n-dimensional space (where n is the number of features) that optimally separates the data into two classes. The key idea behind SVM is to find the hyperplane that

maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the closest data points from both classes. The intuition behind maximizing the margin is that it makes the classifier more robust to noise and generalizes better to unseen data.

The SVM algorithm can be formulated as an optimization problem:

minimize:

$$1/2 \, ||w||^2 \tag{32}$$

subject to:

$$y(i) \, (w^T x(i) + b) >= 1 \text{ for } i = 1, 2, ..., n \tag{33}$$

Where w is the weight vector, b is the bias term, x(i) is the i-th training example, y(i) is the corresponding label (+1 or -1), and n is the number of training examples.

The objective function to be minimized is the squared norm of the weight vector, corresponding to maximizing the margin. The constraints ensure that all data points are correctly classified and lie on the correct side of the hyperplane.

Once the optimization problem is solved, the SVM classifier can be defined as:

$$f(x) = \text{sign} \, (w^T x + b) \tag{34}$$

Where the sign is the sign function, which returns +1 if the argument is positive, and -1 otherwise.

In practice, the optimization problem is often solved using specialized algorithms such as Sequential Minimal Optimization (SMO) or gradient descent. Additionally, SVM can be extended to handle non-linearly separable data by using kernel functions, which map the input data into a higher-dimensional space where it becomes linearly separable. The classification architecture of SVM is shown in Figure 2.20.

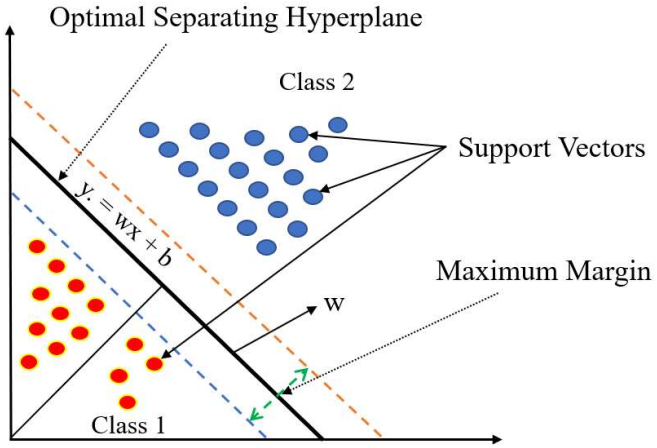

Figure 2.20. Classification architecture of SVM

1.11.4.7 Recursive Discriminant Analysis (RDA)

RDA is a classification algorithm combining linear discriminant analysis (LDA) and recursive partitioning elements. RDA is particularly useful when dealing with high-dimensional datasets that are difficult to model using traditional LDA techniques.

The basic idea behind RDA is to recursively partition the feature space into smaller regions and then fit a discriminant function to each region. The algorithm works by repeatedly splitting the data into smaller subsets based on a set of splitting criteria, and then fitting an LDA model to each subset.

The RDA algorithm can be summarized in the following steps:

1. Initialize the algorithm with the entire dataset.
2. Choose a splitting criterion based on some measure of impurity (e.g., Gini index, entropy).
3. Select the best-split point based on the chosen criterion.
4. Split the data into two subsets based on the selected split point.
5. Fit an LDA model to each subset.
6. Repeat steps 2-5 until some stopping criterion is met (e.g., maximum tree depth, minimum number of samples per leaf).

The discriminant function for each region is given by:

$$f(x) = \log(\text{prior}) + x^T \Sigma^{-1} \mu - \frac{1}{2} \mu^T \Sigma^{-1} \mu \quad (35)$$

where:

- x is the input feature vector
- prior is the prior probability of the class
- mu is the mean vector for the class
- Sigma is the covariance matrix for the class

The RDA algorithm is particularly useful when dealing with high-dimensional datasets that are difficult to model using traditional LDA techniques. By recursively partitioning the feature space into smaller regions, RDA can capture complex decision boundaries that other linear classifiers may miss.

1.11.4.8 Random Forest (RF) Classifier

Random forest is a popular machine-learning algorithm that can be used for classification tasks. An ensemble method combines multiple decision trees to create a more accurate and robust model. Here's an explanation of how random forest classification works, along with some equations to illustrate key concepts:

Data preparation: Random Forest classification requires a dataset with a set of input features and corresponding output labels. The dataset is divided into training and testing sets, with the training set used to build the random forest model and the testing set used to evaluate its performance.

Decision trees: A decision tree is a simple machine learning model that makes predictions by recursively partitioning the input space into smaller and smaller regions based on the values of the input features. Each partition is associated with a decision node that tests a particular feature value, and each leaf node corresponds to a particular output label. The process of constructing a decision tree involves selecting the best feature to split on at each node, based on some criterion such as information gain or Gini impurity.

RF: A RF is an ensemble of decision trees constructed independently and combined to make a final prediction. The randomization comes from the fact that each decision tree is built using a random subset of the training data and a random subset of the input features. This helps to prevent overfitting and to ensure that the ensemble is diverse.

Voting mechanism: To make a prediction using the random forest, each decision tree is evaluated on the input features. The output labels from all the trees are combined using a voting mechanism. In the case of binary classification, the majority vote is taken as the final prediction.

Here are some equations that illustrate key concepts in random forest classification:

Information gain: Information gain is a measure of the reduction in entropy that results from partitioning the data based on a particular feature. The entropy of a binary classification problem can be calculated as:

$$H(y) = - p(y=1) \log2(p(y=1)) - p(y=0) \log2(p(y=0)) \quad (36)$$

Where y is the output label (0 or 1), and p(y=1) and p(y=0) are the probabilities of y=1 and y=0, respectively.

The information gain for a particular feature f can then be calculated as:

$$IG(f) = H(y) - H(y|f) \quad (37)$$

Where H(y|f) is the entropy of the output labels after partitioning the data based on feature f.

Gini impurity: Gini impurity is another measure of the impurity of a data set. For a binary classification problem, the Gini impurity can be calculated as:

$$G(y) = 1 - (p(y=1)) ^2 - (p(y=0))^2 \quad (38)$$

The Gini impurity for a particular feature f can then be calculated as:

$$GI(f) = G(y) - G(y|f) \quad (39)$$

Where G(y|f) is the Gini impurity of the output labels after partitioning the data based on feature f.

Random forest prediction: To make a prediction using a random forest, each decision tree is evaluated on the input features to obtain a set of output labels. The final prediction is then made by taking the majority vote of all the output labels.

Mathematically, this can be expressed as:

$$y\_pred = argmax\_k\ (1/N)\ sum\_i\ (y\_pred\_i == k) \quad (40)$$

Where y_pred_i is the output label predicted by the i-th decision tree, k is a possible output label, and N is the total number of decision trees in the random forest. The RF classifier Model is shown in Figure 2.21.

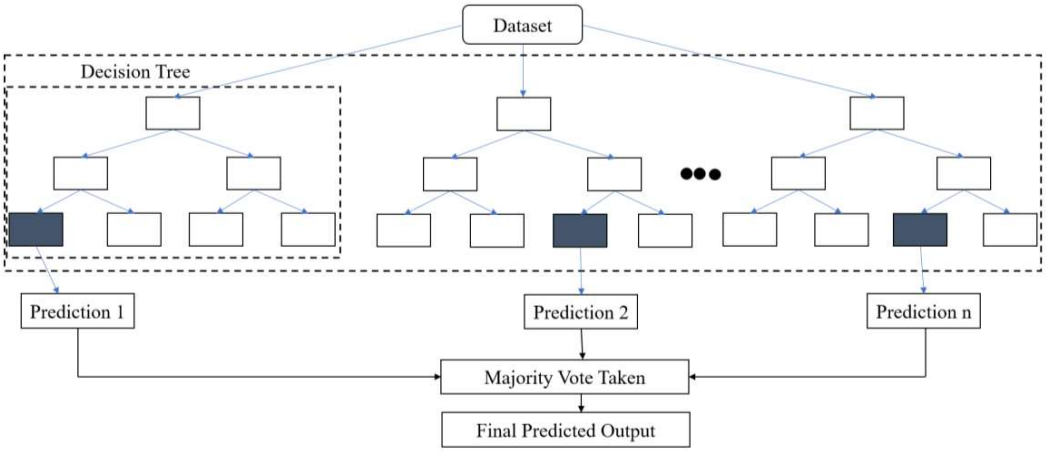

Figure 2.21. RF classifier Model

1.11.4.8. AdaBoost Classifier

The AdaBoost algorithm is a boosting algorithm that combines multiple weak classifiers to form a robust classifier. The design of an AdaBoost classifier involves selecting several hyperparameters, such as the number of estimators, the learning rate, and the random state, to optimize the model's performance. The number of estimators is the number of weak classifiers to combine to form a robust classifier. The learning rate is a hyperparameter that controls the contribution of each weak classifier to the final model. The random state is a seed value that ensures the reproducibility of the

results. In this case, we set the random state to 1. The AdaBoost Classification model is shown in Figure 2.22.

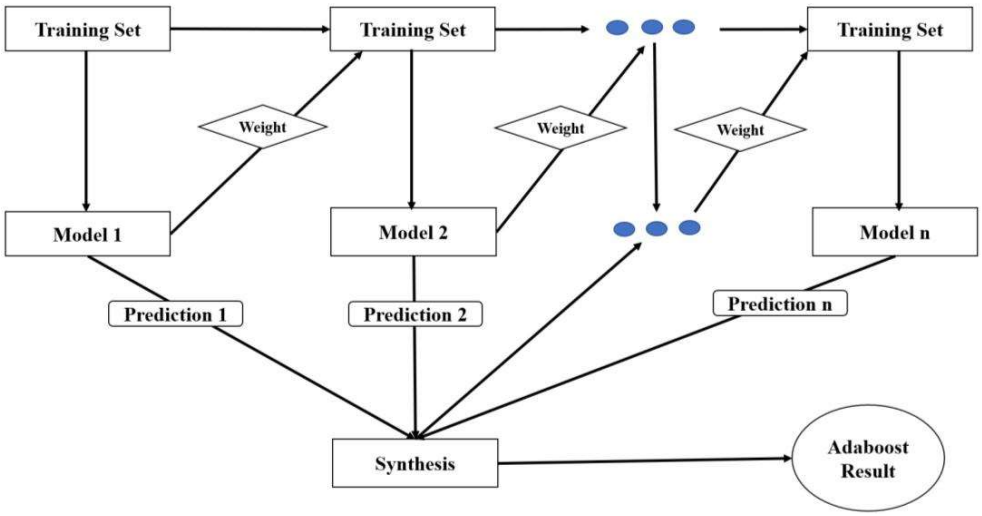

Figure 2.22. AdaBoost Classification Model.

1.11.4.10 XGBoost Classifier

The XGBoost classifier is a specific implementation of XGBoost that is used for classification tasks. It uses decision trees as its weak models and combines their predictions through gradient boosting. The algorithm works by iteratively adding decision trees to the model, with each new tree correcting the errors of the previous ones. The process continues until a maximum number of trees or a minimum improvement in accuracy is reached. The learning rate is the step size at which the algorithm learns from the mistakes of each iteration. A lower learning rate can lead to more accurate results, but it also makes the algorithm slower to converge.

In this case, the learning rate used is 0.1, which is a moderate value. The number of estimators is the number of decision trees to be created by the algorithm. Increasing the number of estimators can lead to better performance but can also lead to overfitting. The maximum depth of a decision tree is the maximum number of levels it can have. A deeper tree can learn more complex relationships between features but can also lead to overfitting. The maximum depth is set to 4, which is a moderate value. The minimum sum of instance weight needed in a child node. This parameter controls overfitting by setting it to a higher value. In this case, the value used is 6, which is a moderate value.

Gamma is a parameter used to control regularization, which penalizes the model for creating new nodes in the tree. A higher gamma value means more regularization, which can help prevent overfitting. Regularization is a method to prevent overfitting by adding a penalty term to the loss function. The reg_alpha parameter controls the L1 regularization term. The number of threads is the number of CPU cores to use for parallel processing. In this case, 4 threads are used. The architecture of the XGBoost classifier is presented in Figure 2.23.

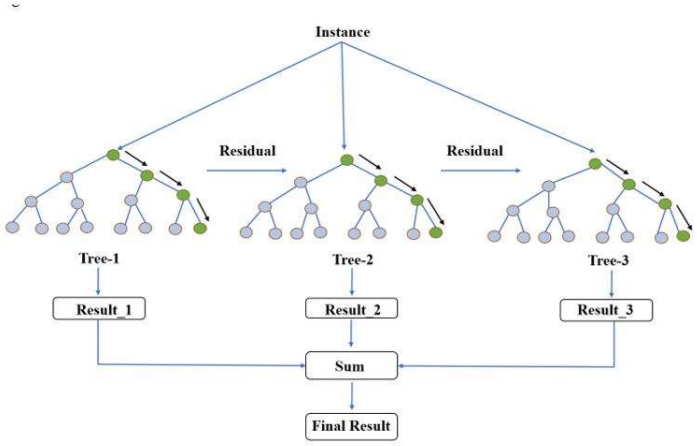

Figure 2.23. The architecture of XGBoost classifier.

1.11.4.11 Multilayer Perceptron (MLP) Classifier

An MLP classifier is an artificial neural network commonly used for supervised learning tasks, such as classification and regression. The MLP comprises multiple layers of nodes (also known as neurons) that are connected by weighted edges. The first layer is the input layer, which receives the input data. The last layer is the output layer, which produces the network's output. Between the input and output layers, there are one or more hidden layers, which help the network learn and extract useful features from the input data.

A MLP is a type of artificial neural network (ANN) that consists of multiple layers of nodes (neurons) arranged in a series of interconnected layers. MLPs are typically used for supervised learning tasks such as classification, regression, and pattern recognition. The basic components of an MLP are the input layer, one or more hidden layers, and the output layer. The input layer receives the input data, which is

passed through the hidden layers to the output layer. Each neuron in the hidden and output layers has an activation function, which transforms the weighted sum of the inputs into an output signal.

The weights between the neurons are learned through backpropagation, which involves computing the error between the predicted output and the actual output, and then adjusting the weights to minimize the error. This process is repeated many times, using different input data and adjusting the weights each time until the network can accurately predict the output for a given input. MLPs can be used for various applications, including image and speech recognition, natural language processing, and financial modelling.

Each node in the MLP computes a weighted sum of its inputs and applies a non-linear activation function to produce its output. The weights and biases of the MLP are adjusted during training using an optimization algorithm such as backpropagation, which minimizes the difference between the predicted outputs and the true labels. The MLP model consists of one or more hidden layers, and this parameter specifies the number of nodes in each hidden layer. In this case, there is one hidden layer with 11 nodes.

The activation function is used to introduce non-linearity into the MLP model. The Relu activation function is used, which stands for the rectified linear unit. It is a popular activation function that sets negative values to zero and keeps positive values unchanged. The solver is the optimization algorithm used to train the MLP model. The "adam" solver is used, which is a popular stochastic gradient descent (SGD) optimization algorithm. The batch size is the number of training examples used in each iteration of the training process. A smaller batch size can lead to more noise in the updates, but it can also speed up the training process. In this case, the batch size used is 100.

The learning rate determines the step size at which the algorithm learns from the mistakes of each iteration. The "adaptive" learning rate is used, which means that the learning rate is adjusted based on the progress of the training process. The maximum number of iterations is the number of times the MLP model is trained on the data. In this case, the maximum number of iterations used is 100. Cross-validation (CV) is used to evaluate the performance of the MLP model. In this case, a 5-fold cross-validation is

used, which means that the data is split into 5-equal parts, and the algorithm is trained on 4-parts and tested on the fifth part. This process is repeated 5-times, with each part being used as the test set once. The schematic diagram of the MLP classifier is presented in Figure 2.24.

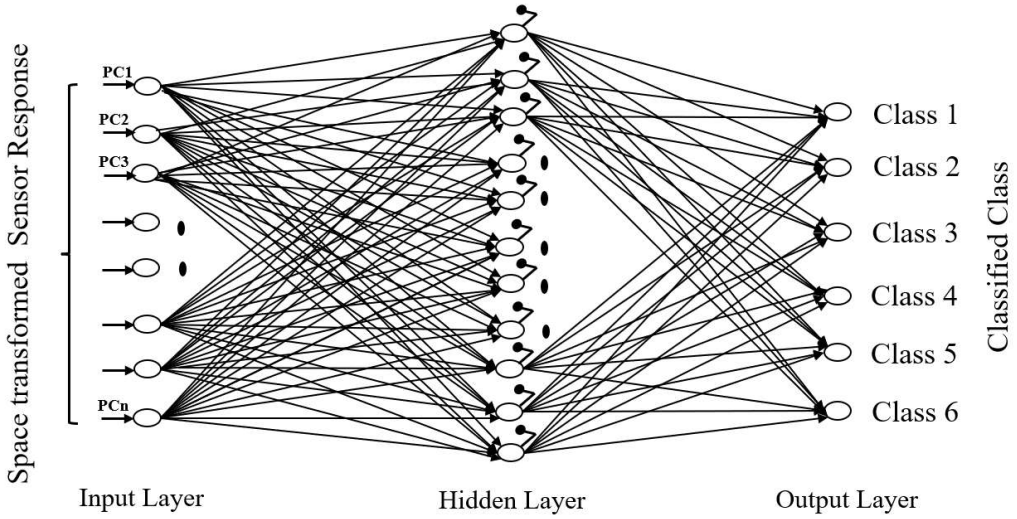

Figure 2.24. The schematic diagram of the MLP classifier.