Classification of Gas-Sensor array data using Machine Learning

A Dissertation Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF SCIENCE

in

School of Mathematics

by

Prashant Sharma

Roll No. IMS19175

to

SCHOOL OF Mathematics

INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH

THIRUVANANTHAPURAM - 695 551, INDIA

April 2024

DECLARATION

I, **Prashant Sharma (Roll No: IMS19175)**, hereby declare that, this report entitled **"Classification of Gas-Sensor array data using Machine Learning"** submitted to Indian Institute of Science Education and Research Thiruvananthapuram towards partial requirement of **Master of Science** in **School of Mathematics** is an original work carried out by me under the supervision of **Dr.Dharmatti Sheetal and Dr. Vinayak B. Kamble** and has not formed the basis for the award of any degree or diploma, in this or any other institution or university. I have sincerely tried to uphold the academic ethics and honesty. Whenever an external information or statement or result is used then, that have been duly acknowledged and cited.

Thiruvananthapuram - 695 551 **Prashant Sharma**

April 2024

CERTIFICATE

This is to certify that the work contained in this project report entitled **"Classification of Gas-Sensor array data using Machine Learning"** submitted by **Prashant Sharma** (**Roll No: IMS19175**) to Indian Institute of Science Education and Research, Thiruvananthapuram towards the partial requirement of **Master of Science** in **School of Mathematics** has been carried out by him under my supervision and that it has not been submitted elsewhere for the award of any degree.

Thiruvananthapuram - 695 551

Dr Dharmatti Sheetal

Dr. Vinayak B. Kamble

April 2024

Project Supervisors

ACKNOWLEDGEMENT

I thank everyone who helped me see this project through to completion. I would like first to express my profound gratitude and deep regard to Dr Dharmatti Sheetal and Dr Vinayak B. Kamble, IISER Thiruvananthapuram and sincerely wish to acknowledge their vision, guidance, valuable feedback and constant support throughout the duration of this project. I would also like to thank Shivam, Niharika and Sajana for the data collection.

I am indebted to all my friends especially Pratheek, Samarpita, Rishica, Meghnath, and Phalguni, for their steadfast encouragement and time. I am lastly grateful to the Indian Institute of Science Education and Research Thiruvananthapuram for providing the necessary resources and facilities to complete this project to the best of my ability.

Thiruvananthapuram - 695 551 **Prashant Sharma**

April 2024

ABSTRACT

The execution of experiments involves and creates a large amount of data that needs to be acquired, organized and assessed to find a meaningful outcome of the experiments. The main aim of the project is to understand and create a Machine-learning model to identify the gases present in a system based on information provided by the sensors. To begin with we started to learn Principal Component Analysis (PCA), a well-known technique in data analytics. Various methods and techniques are studied and have been implemented in the well-known data sets. A python-based library is generated to perform PCA. It was compared with the commercially available SKlearn library. Apart from this, to collect thermal expansion data of the samples, a device was built which can measure expansion in a temperature range varying from room temperature to 900 K. The device needs to be optimized for collecting reliable data which is the work under progress.To understand the mathematics behind some useful Machine Learning Algorithms for classification problems and to apply these algorithm to a given set of data generated by Gas Sensors and identify the gases based upon the parameters given in dataset. for that classification algorithms such as K-Nearest Neighbour, Decision Tree, Random Forest and specially neural networks were used. before classification, PCA(Principal Component Analysis) was used in attempt to reduce the number of features/parameters in the dataset.

Contents

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Gas Sensor Array 2

1.2 Classifiers in ML 3

1.2.1 Non-Linear Classifier 4

2 Data 7

2.1 Data Overview 7

2.2 Prepossessing Data 9

3 KNN and Decision Tree **11**

3.1 KNN 12

3.1.1 KNN Model 12

3.1.2 KNN results 13

3.2 Decision Tree 15

3.2.1 Analysis with Decision Tree 15

3.2.2 Decision Tree Results 17

3.2.3 Random Forest 17

3.2.4 Random Forest Results 19

4 Neural Networks Based Classifiers **20**

4.1 How Neural Networks Works 20

4.2 Architecture of Neural Network 22

4.2.1 Forward Neural Networks 23

4.2.2 Activation Function 25

4.2.3 Multi-Layer Perceptron 27

4.2.4 Model Building 30

4.2.5 Back Propagation 32

4.3 stochastic gradient descent 34

4.3.1 Gradient Descent 34

4.4 Neural Network Classification Model 35

4.5 Neural Networks Results 37

5 Multi-Sensor Array **39**

6 Conclusion **43**

Appendices **46**

A Data collection of three simultaneous sensors **46**

B Building Thermal Expansion Measurement System and its interfacing **48**

B.1 Building Thermal Expansion Measurement System and its interfacing 48

B.1.1 The principle of measurement system 48

B.1.2 Instrumentation 49

B.1.3 Calibration and Measurement 54

B.2 Thermal Expansion Temperature v/s Voltage 56

Bibliography **58**

List of Figures

2.1 Data Head 8

2.2 Percentile Plot 10

2.3 K-Density Plot of dataset 10

3.1 KNN Confusion Matrix 14

3.2 Confusion Matrix Decision Tree 18

3.3 Confusion Matrix Random Forest 19

4.1 Dot-products in Matrix Formatting 29

4.2 Confusion Matrix for trained neural networks on New Data set 38

5.1 Schematics of six sensor arrays with individual heater control using microcontroller and the sensor output leads (Interdigitated gold electrodes) to simultaneously measure sensor response of 6 nodes. The typical single-sensor device is shown with top electrodes and a bottom heater. 40

5.2 The data recorded with the assembly described in Fig 5.1 for a single sensor using a controller made. (measurement courtesy Ms. Niharika) 42

6.1 Model Comparison on training and unknown data set 44

A.1 The schematic diagram of the gas sensing setup used for the experiment [11] 47

B.1 The schematic diagram of the home built Thermal expansion measurement system and its digital photograph. 50

B.2 Circuit diagram of LVDT primary winding and Secondary winding. . 52

B.3 Different placements of metal core inside LVDT and corresponding voltage signal being generated. 53

B.4 (a) Without and (b) With sample thermal expansion measurement data. Two Cu samples in the stack (each with L = 4.12mm) were used as samples. 55

B.5 Measurements performed on the (a) same Cu sample, L= 4.12mm and (b) longer sample (8.04 mm length Cu sample) 56

B.6 without sample thermal expansion measurement 57

List of Tables

2.1 Data Overview 8

4.1 Training Neural Network Model 37

Chapter 1

Introduction

Gas sensors are essential instruments in many fields. These sensors are mainly used for safety, and they can identify dangerous gases such as carbon monoxide, hydrogen sulfide, methane, or any volatile organic compounds(VOCs).[11] They are vital for avoiding mishaps and protecting people's health in working environments. Identifying contaminants in indoor and outdoor environments is essential to environmental monitoring[10] because it helps reduce pollution and protect public health, even early disease detection. For the food [1] and pharmaceutical industries[8], precise control over gas concentrations is essential in industrial processes provided by gas sensors to preserve product quality and safety. Gas sensors are used in medical devices to measure carbon dioxide and oxygen levels, which are vital for patient care, as well as exhaust emissions in automotive applications. By warning residents and first responders about possible fire threats, gas sensors help fire detection systems improve safety. To put it simply, gas sensors are invaluable resources that support environmental preservation, safety, and quality assurance in a variety of sectors.

1.1 Gas Sensor Array

When using a single gas sensor for detection, there may be many reasons why the results could be false or inaccurate. As a result, sensor arrays need to be used. The ability of an array to improve accuracy by cross-referencing data from various sensors is one of its significant advantages. Different gases frequently create similar reactions on a single sensor, which could result in misunderstandings or inaccurate results. On the other hand, an array can more accurately distinguish between different gases by combining sensors with different selectivity. Additionally, sensor arrays' redundancy guarantees fault tolerance, which is essential in critical applications requiring ongoing monitoring. Even though a single gas sensor can give helpful information, an array of sensors can provide advantages like fault tolerance, enhanced accuracy, selective detection, broader dynamic range, continuous calibration, and better spatial coverage. An array of sensors is essential for critical applications where accuracy and dependability are crucial.

The complexity of data from an array can be too complex. For this work, resistance, the concentration of gas, temperature, sensor element and the number of sensors are the parameters collected, so with these features, the data becomes too complex to visualise. With this complexity, predicting the target/result using the usual linear methods is difficult. So, some non-linear method in Machine Learning is required to fit the data and predict the gas based on given new parameters. With an array of gas sensors, it becomes handy to check the result, as individual sensors can have a false result. For that, supervised Machine learning algorithms such as KNN, Decision Tree, Random Forest, and Neural Networks were used to classify the data for four different gases.

In this project, we have analysed data collected from three sensors exposed to four different gases and explored classification. Also designed hardware where four to six sensors can be accommodated, and data can be collected simultaneously.

1.2 Classifiers in ML

Classifiers in machine learning are algorithms that categorise or classify input data according to its features. These algorithms apply labelled training data to identify patterns, which can be applied to predict the labels of new data points. A key component of supervised learning is the classifier, in which the computer gains knowledge from input-output relationships given during training.

In machine learning, classifiers are algorithms that assign labels or categories to input data based on their features. These algorithms learn patterns from labelled training data and then use this knowledge to predict the labels of new, unseen data points. Classifiers are a fundamental part of supervised learning, where the algorithm learns from input-output pairs provided during training.

Binary Classifiers

These classifiers classify inputs into one of two classes or categories, such as "spam" or "not spam," "positive" or "negative," etc.

Multiclass Classifiers

These classifiers classify inputs into one of multiple classes or categories. For example, classifying images of animals into "dog," "cat," "bird," etc.

Linear Classifiers

These classifiers separate classes using a linear decision boundary. Examples: logistic

regression and linear support vector machines (SVM).

Non-Linear Classifiers

These classifiers can capture non-linear relationships between features and labels. Examples: decision trees, random forests, k-nearest neighbours (KNN), and neural networks.

Probabilistic Classifiers

These classifiers predict the class label and provide the probability of a data point belonging to each class. Examples include logistic regression and Naive Bayes classifiers.

Deep Learning Classifiers

These classifiers are based on deep neural networks, which can automatically learn hierarchical representations of data. They are particularly effective for tasks such as image classification, natural language processing, and speech recognition.

Our goal is to predict the gas using four parameters. We also know there are only four types of gas in our dataset, so we can use one of the classifiers, such as Multiclass Classifiers or Non-Linear Classifiers. Since our dataset performs better with non-linear classifiers, our approach used non-linear classifiers, including decision trees, random forests, K-nearest neighbours, and neural networks.

1.2.1 Non-Linear Classifier

Non-linear Classifier can capture Complex relationships between input features and target labels; here, we prefer the Non-Linear approach because linear classifiers assume that the decision boundary separating different classes is a straight line or a hyperplane in higher dimensions; non-linear classifiers are capable of capturing more

intricate decision boundaries even in an irregular shape.

k-Nearest Neighbors (KNN)

KNN is a simple instance-based learning algorithm where the prediction for a new data point is based on the labels of its k nearest neighbours in the feature space. KNN is non-parametric and can learn complex decision boundaries directly from the training data.

Decision tree

Decision trees recursively partition the feature space into smaller regions by making decisions based on feature values at each node. Each decision leads to a split, creating branches that eventually terminate in leaf nodes representing the predicted class labels.

Random Forest

Random forests are ensembles of decision trees where each tree is trained on a random subset of the training data and a random subset of features. The final prediction is made by aggregating the predictions of individual trees, often by a majority vote.

Neural Networks

Neural networks, particularly deep neural networks, are highly flexible non-linear classifiers capable of learning complex mappings between inputs and outputs through

multiple layers of interconnected neurons. It has achieved state-of-the-art performance in various machine learning tasks, including image and speech recognition, natural language processing, and many others. We worked and spent more time on the neural Network Approach to Our Problem.

Chapter 2

Data

2.1 Data Overview

The dataset was generated in VK Lab with the aim of Identifying the gas by using a gas sensor of different types at different temperatures. A sensor array of 3 component metal oxides was used to identify the mixture's four distinct volatile organic compounds(VOCs). The metal oxide sensor array comprises NiO-Au(ohmic), CuO-Au(Schottky), and ZnO-Au(Schottky) sensors made by DC reactive sputtering method and having a thickness of 80-100nm. This array was subject to various VOC concentrations, including ethanol, acetone, toluene, and chloroform, one by one and in a pair of gases [11]. The dataset consists of about 22 lac data points with features/variables such as Resistance, Sensor, Gas Concentration, Temperature, and Gas. The last one is the target variable. By using some suitable ML classification for Multiclass Classification, the models were trained to identify the Gas(Target Variable).

	Resistance	Sensor	Temp	Conc	Gas
0	2032.37575	0	200.0	2400	2
1	2032.42944	0	200.0	2400	2
2	2032.60295	0	200.0	2400	2
3	2032.66493	0	200.0	2400	2
4	2032.65253	0	200.0	2400	2
...	...	...	...	...	...
2262999	100272.94300	2	350.0	1000	0
2263000	100295.06810	2	350.0	1000	0
2263001	100314.88840	2	350.0	1000	0
2263002	100335.92470	2	350.0	1000	0
2263003	100356.66760	2	350.0	1000	0

2263004 rows × 5 columns

Figure 2.1: Data Head

Table 2.1: Data Overview

	Resistance	**Sensor**	**Temp**	**Conc**	**Gas**
min	8.655768e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
max	1.480000e+06	2.000000e+00	2.000000e+00	3.000000e+01	3.000000e+00

The resistance column has a range of order 2 to 6; the sensor column consists of three different types of sensors: NiO, CuO, and ZnO. The Temperature column consists of three unique values. [2400, 2100, 1800, 1500, 1200, 600, 300, 500, 100, 50, 20, 10, 5, 2, 2500, 2000, 1000, 2700, 1900, 1700, 1550, 1370, 1600, 1400, 900, 720, 540, 360, 180, 90, 30] These were the concentration used for sensing in the Conc column. The Last column is the target column, which has four types of gases: ethanol, acetone, toluene, and chloroform.

2.2 Prepossessing Data

So, our dataset has a total of five columns, the last of which is the target column. Since it is not a numerical attribute and we also want to classify them, it is necessary to hot encode the labels in the target column. so, inside the Gas column (target column) we have 'Gas' encoding: Unique values: ['Acetone' 'Chloroform' 'Ethanol' 'Toluene'] Encoded labels: [0 1 2 3]

and the sensor also has a non-numerical attribute therefore hot encoding is needed 'Sensor' encoding: Unique values: ['CuO' 'NiO' 'ZnO'] Encoded labels: [0 1 2]

Data Processing with one hot encoding for target labels, resistance, and other features was scaled using a min-max scalar in zero to one range. Missing values in the dataset may result in bad model training, so using the mean method to find missing values was also added. Now, the data set has all the numerical values.

To train any model on a dataset, we need to shuffle and split it. Here, 80% of data was used to train the model and 20% of data was kept as a completely new unknown dataset to verify the model externally. That 80% of the data was further used for supervised learning with a train-test split to the same ratio. Just to see if there are extreme fluctuations in the resistance column, the percentile plot(Fig 2.2) was made. A K-density plot was used to visualize how each gas was distributed.Fig(2.3)

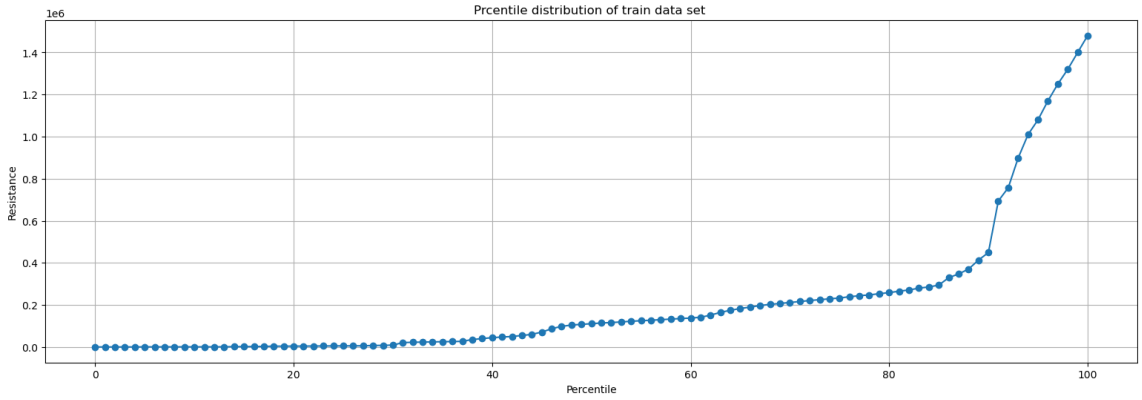

Figure 2.2: Percentile Plot

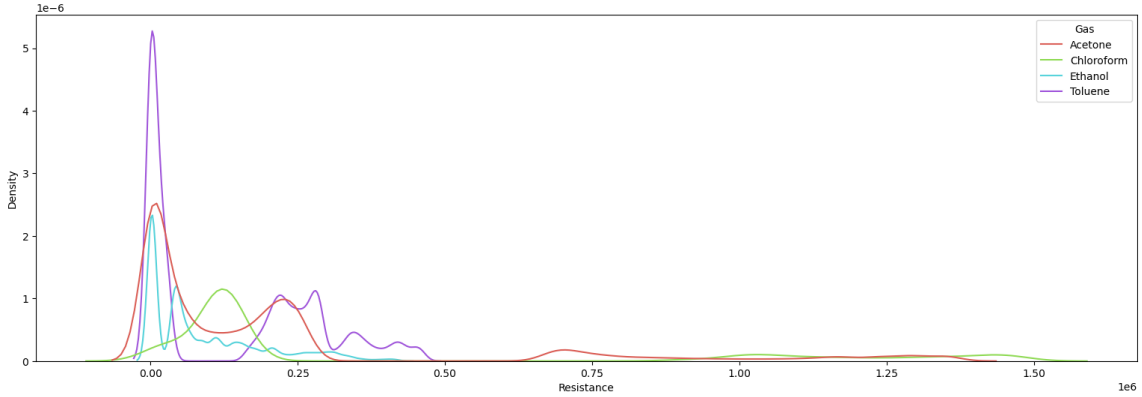

Figure 2.3: K-Density Plot of dataset

Chapter 3

KNN and Decision Tree

This chapter will analyse the mathematics behind KNN, decision trees, and random forests as multiclass classifiers.

Definition 3.0.1. Shattering is the ability of a model to classify a set of points perfectly. More generally, the model can create a function that can divide the points into two distinct classes without overlapping. It is different from simple classification because it considers all possible combinations of labels upon those points.

Definition 3.0.2. VC dimension The Vapnik-Chervonenkis dimension, more commonly known as the VC dimension, is a model capacity measurement used in statistics and machine learning. The VC dimension of a model is the size of the largest set of points that that model can shatter.

3.1 KNN

KNN relies on the idea that similar data points tend to have similar labels or values. The KNN algorithm stores the entire training dataset as a reference during the training phase. When making predictions, it calculates the distance between the input data point and all the training examples, using a chosen distance metric such as Euclidean distance.

Next, the algorithm identifies the K nearest neighbours to the input data point based on their distances. In the case of classification, the algorithm assigns the most common class label among the K neighbours as the predicted label for the input data point. For regression, it calculates the average or weighted average of the target values of the K neighbours to predict the value for the input data point.

3.1.1 KNN Model

Feature matrix X ($n\_samples \times n\_features$), target vector y ($n\_samples$) Accuracy of the KNN classifier on the test data **Step 1: Split the Data** Split the data into training and testing sets using $train\_test\_split$ **Step 2: Import Libraries** import $KNeighborsClassifier$ from scikit-learn and $accuracy\_score$ from scikit-learn.metrics **Step 3: Instantiate the KNN Classifier** Create an instance of $KNeighborsClassifier$ with the desired number of neighbors **Step 4: Train the Classifier** Fit the classifier to the training data **Step 5: Make Predictions** Use the trained classifier to predict the labels of the test data **Step 6: Evaluate the Model** Calculate the accuracy of the model by comparing the predicted labels with

the true labels of the test data **Step 7: Output the Results** Print or return the accuracy of the model on the test data

- **Split data** splitting the data into training and testing sets to evaluate the model's performance is not necessary when using KNN on the same dataset.
- **Import libraries** import the KNeighborsClassifier from scikit-learn.
- Instantiate the KNN classifier: Create an instance of the KNeighborsClassifier class. Specify the number of neighbours (n_neighbors) and other parameters.
- **Train classifier** Fit the classifier to your entire dataset.
- **Make predictions** Use the trained classifier to make predictions on the same dataset.

Since predicting on the same dataset trained on, we won't get a true evaluation of the model's performance. So, we used the trained model on a new data set to achieve accuracy.

3.1.2 KNN results

A better model was obtained by KNN with n value 5. For all input features were converted to 0 to 1 as input and target variable hot encoded. The true negative, true positive, false negative, and false positive predictions for the entirely unknown data set of around four lac data points are explained in the confusion matrix.

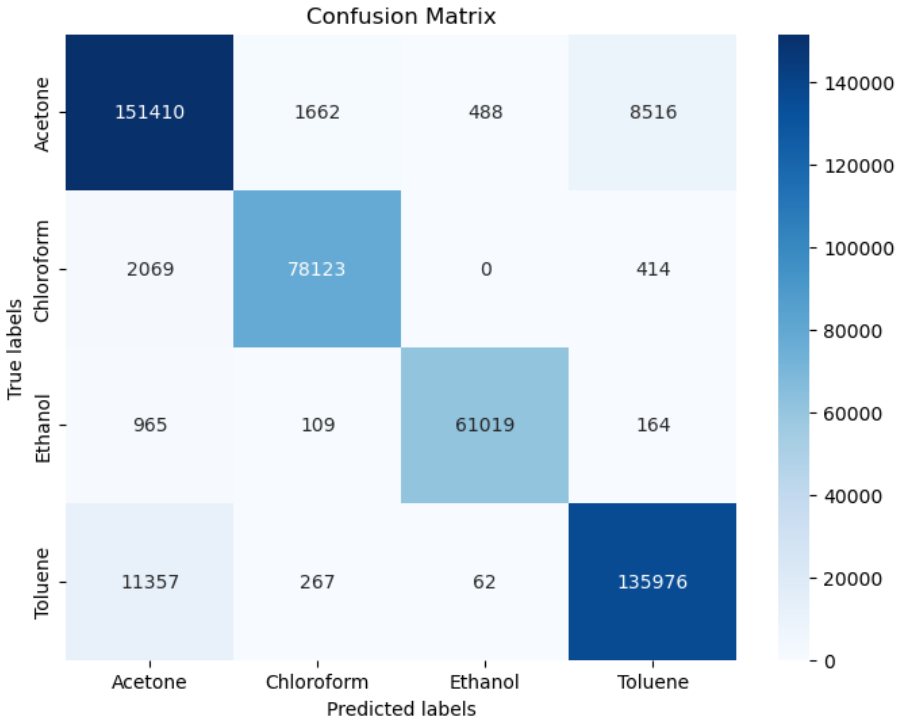

Figure 3.1: KNN Confusion Matrix

3.2 Decision Tree

A general framework for growing a decision tree starts with a tree with a single leaf(the root) and assigns this leaf a label according to a majority vote among all labels over the training set. Then, a series of operations are performed. On each iteration, the effect of splitting a single leaf is examined. Then, among all possible splits, we choose the one that maximizes the gain and performs it or chooses not to split the leaf.

A possible Implementation is based on a popular decision tree algorithm known as "ID3" (Iterative Dichotomizer 3), the algorithm for the case of binary features, namely, $\xi = 0, 1_d$, and therefore all splitting rules are of the form $1_{[x_i=1]}$ for some feature $i \in [d]$.

3.2.1 Analysis with Decision Tree

Starting with splitting the data into training and testing sets to evaluate the model's performance 80% training data and 20% data for testing was used. Before splitting data, it is shuffled to ensure randomness. The decision tree algorithm from the library was selected and trained. And evaluated using accuracy metrics. After fine-tuning hyperparameters, the algorithm will get better accuracy. After getting the desired accuracy, the model was saved and again used for a completely new data set for cross-validation, and a new confusion matrix was used to visualize the performance of the model.

hyper-parameters

Hyperparameters in decision trees are parameters that are set before the learning process begins. They control aspects of the tree's construction and can significantly impact the resulting model's performance and behaviour.

- **Criterion** Criterion is the function used to measure the quality of a split. Common options are "Gini" for the Gini impurity and "entropy" for information gain.
- **Max Depth** Max depth controls the maximum depth of the decision tree. It limits the number of nodes in the tree. Setting this parameter helps prevent overfitting.
- **Min Samples Split** Min samples split specifies the minimum number of samples required to split an internal node. The node will not be split if the number of samples at a node is less than this parameter.
- **Min Samples Leaf** Min samples leaf is the minimum number of samples required to be at a leaf node. A split will only be made if it leaves at least this number of samples in each left and right branch.
- **Max Features** Max features determine the maximum number of features to consider when looking for the best split. It can be an integer (considering a fixed number of features) or a fraction (considering a percentage of features).
- **Splitter** Splitter determines the strategy used to choose the split at each node. Options are "best" to choose the best split and "random" to choose the best random split.

- **Min Impurity Decrease** Min impurity decrease is a threshold for the minimum decrease in impurity required for a split to happen. It helps prevent overfitting by requiring a certain amount of improvement in purity.
- **Class Weight** lass weight is used to handle class imbalance. It assigns weights to different classes to balance their representation in the tree.
- **Presort** Presort specifies whether to presort the data to speed up the finding of best splits during training. It is generally not recommended for large datasets.

3.2.2 Decision Tree Results

A model was obtained by a Decision tree for all attributes that were converted to 0 to 1 as input and target variable hot encoded. The true negative, true positive, false negative, and false positive predictions for the entirely unknown data set of around four lac data points are explained in the confusion matrix).

3.2.3 Random Forest

The Class of Decision Tree of arbitrary size has infinite VC dimension. So, a restriction on the size of the decision tree is needed. The danger of overfitting can be reduced by constructing an ensemble of trees. the method of $random forests$ was introduced by Breiman(2001). A random forest is a classifier consisting of a collection of a collection of decision trees, where each tree is constructed by applying an algorithm A on the training set S and an additional random vector, θ, where, θ is sampled independent and identically distributed from distribution. A majority vote over the individual trees' predictions obtains the random forest's prediction.

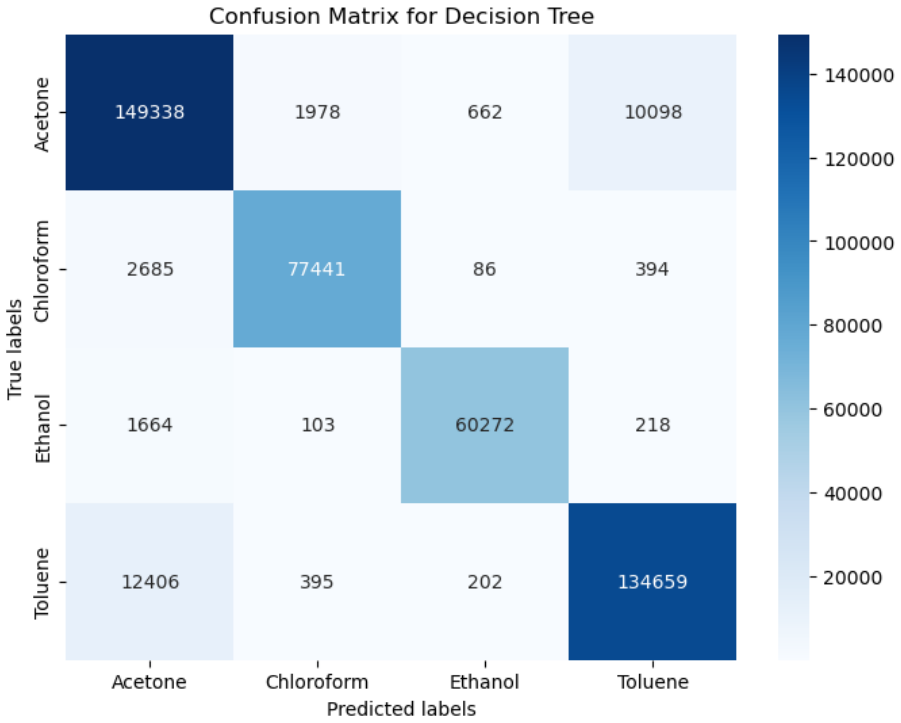

Figure 3.2: Confusion Matrix Decision Tree

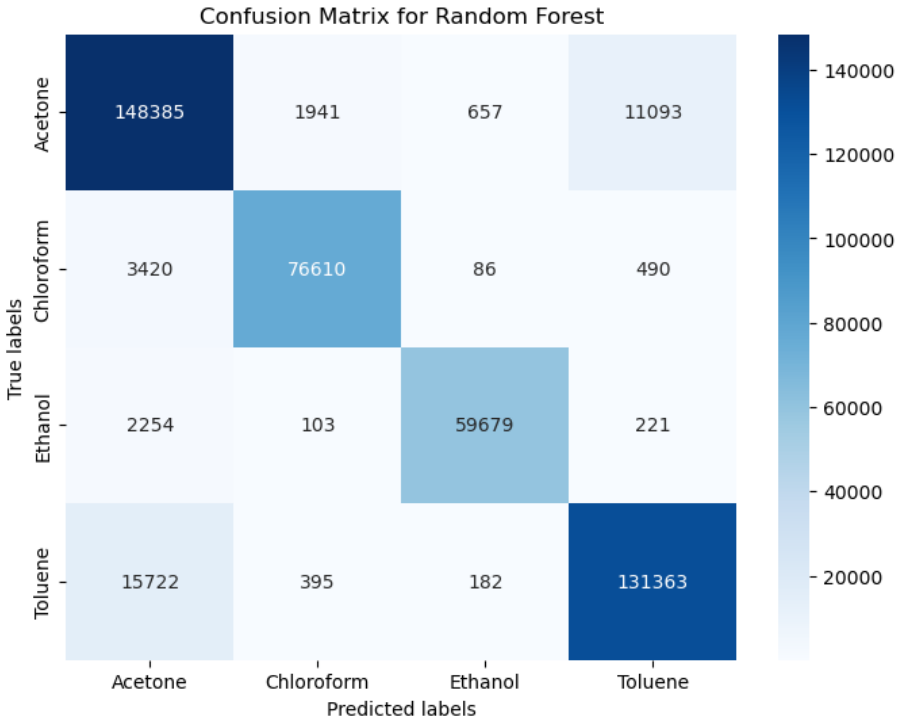

Figure 3.3: Confusion Matrix Random Forest

3.2.4 Random Forest Results

A better model was obtained by Random Forest for all attributes that were converted to 0 to 1 as input and target variable hot encoded. The true negative, true positive, false negative, and false positive predictions for the entirely unknown data set of around four lac data points are explained in the confusion matrix.

Chapter 4

Neural Networks Based Classifiers

Classification problems for a complex and large dataset with unpredictable results using linear methods can be calculated using neural networks, which can be trained to adjust with the result requirements. Also, once we train the network, it can be used on any similar dataset efficiently and with higher accuracy. Neural networks are just a set of functions that change themselves to learn and fit into a model.

4.1 How Neural Networks Works

Neural networks are computational models inspired by the structure and function of the human brain. It consists of interconnected nodes in this case neurons, organized into layers. Each neuron receives input signals, processes them using an activation function, and produces an output signal. Layers are typically organized into an input layer, one or more hidden layers, and an output layer. The input layer receives

raw data, while the output layer produces the model's predictions or classifications. There are hidden layers between the input and output layers, where the network learns to predict the result from the input data. Each connection between neurons is associated with weights, and each neuron also has a bias term. During training, the network adjusts these weights and biases using an optimization algorithm to minimize the loss function, which measures the difference between the predicted output and the true labels.

A neural network is trained by giving it labelled training data and modifying its weights and biases to reduce the difference between the labels it predicts and the actual labels. This is done through an iterative process of forward propagation, loss calculation, and backpropagation. In forward propagation, input data is passed through the network, layer by layer. At each layer, the input is transformed by the weights and biases of the neurons and then passed through the activation function to produce an output, and this Output becomes the Input of the next layer. This process continues until the output layer is reached, producing the model's prediction. After getting the Output at the end of the last layer, the loss function is calculated by comparing the predicted output to the true labels or target values. Common loss functions are mean squared error for regression tasks and cross-entropy loss for classification tasks. The network then adjusts its weights and biases through backpropagation, by computing the gradient of the loss function with respect to each weight and bias in the network and updating them accordingly. This process continues for a predefined number of iterations (epochs) or until the model's performance on a separate validation dataset reaches a satisfactory level of result.

4.2 Architecture of Neural Network

A neural network consists of many neurons called nodes. Their amount could be from a couple of dozen to even millions, and they are arranged in layers. All the nodes can be classified into input, hidden, and output nodes, which connect the layers on either side.

The Input layer consists of input nodes responsible for receiving numerical data from outside of the dataset that the neural network attempts to learn about. The connection between one node from the input layer and one from the hidden layer is represented by a number called weight, which is denoted as W_i. The weight can be positive or negative, corresponding to how brain cells excite or suppress others. If a node has a higher corresponding weight, then it has more influence on the output. Initially, the weights are assigned randomly and are adjusted later in the training process.

In neural networks, information can flow through a neural network in two ways. The simplest type of Neural network is Feedforward Neural Network. In Feedforward Neural Network, the data value goes only in one direction, from the input nodes to hidden nodes, and then finally to the output nodes.

The Input layer Collects the numerical variables from the Dataset and carries them to the next stage, no computation happens here. Then, the data values are at the first hidden layer, which has an activation function. The activation functions perform computations and transfer the new values to the next hidden layer or output layer. A feedforward network will only have a single Input layer, and it can also have zero or multiple hidden layers. The output layer contains all output nodes, which

our results computed from prior hidden layers can be considered. One type of neural network, known as a recurrent neural network, has a loop or cycle connecting all the nodes inside it. The recurrent attribute allows it to exhibit temporal dynamic behaviours.

4.2.1 Forward Neural Networks

Neural networks refer to the complex links among neurons; communication links can join many neurons to carry out complex computations. The structure of a neural network can be described as a graph whose nodes are the neurons, and each edge in the graph links the output of one neuron to the input of another neuron. For the feed-forward network structure, the graph does not contain any loop. A feed-forward neural network can be described by a directed acyclic graph, G = (V, E), and a weight function over the edges, $w : E \to R$. Nodes of the graph can be considered as neurons. Each single neuron is moulded as a simple scalar function, $\sigma : R \to R$. σ can be a

sign function,

$$\sigma(a) = sign(a)$$

threshold function,

$$\sigma(a) = 1_{[a>0]}$$

the sigmoid function,

$$\sigma(a) = \frac{1}{1 + \exp -a}$$

The sigmoid function is a smooth approximation to the threshold function. σ is called the "activation function" of the neuron. Each edge in the graph links some neuron's output to another neuron's input. The input of a neuron is calculated by taking a weighted sum of the outputs of all the neurons connected to it, weighting can be tweaked by w. Assume that the network is organized in *layers*, that is, the set of nodes can be decomposed into a union of nonempty disjoint subsets, $V = \bigcup_{t=0}^{T} V_t$ (disjoint union), such that every edge in E connects some node in V_{t-1} to some node in V_t, for some $t \in [T]$.

The bottom layer, V_0 is called the input layer. It contains n+1 neurons, where n is the dimension of the input space. For every $i \in [n]$, the output of neuron i in V_0 is simply x_i. The last neuron in V_0 is the "constant" neuron, which always outputs 1.

let's denote $v_{t,i}$ the ith neuron of tth layer and $o_{t,i}(X)$ the output of $v_{t,i}$ when the network is fed with the input vector x. Therefore, for $i \in [n]$ we have $o_{0,i}(x) = x_i$ and for $i = n + 1$ we have$o_{0,i}(x) = 1$. Let's calculate in a layer-by-layer manner. Suppose we have already calculated the outputs of neurons at layer t.

Then, we can find the outputs of the neurons at layer $t + 1$ by fixing some $v_{t+1,j} \in V_{t+1}$. Let $a_{t+1,j}(x)$ denote the input to $v_{t+1,j}$ when the network is fed with the input vector x.

Then,

$$a_{t+1,j}(x) = \sum_{r:(v_{t,r},v_{t+1,j})\in E} w((v_{t,r}, v_{t+1,j}))o_{t,r}(x),$$

and

$$o_{t+1,j}(x) = \sigma(a_{t+1,j}(x)).$$

The input to $v_{t+1,j}$ is a weighted sum of the outputs of the neurons in V_t that are connected to $v_{t+1,j}$, where weighting is according to w, and the output of $v_{t+1,j}$ is simply the application of the activation function σ on its input.

Layers $V_1, \cdots, V_{T-1}$ are often called *hidden layers*. The top layer V_T is called the Output Layer. In Simple prediction problems, the output layer contains a single neuron whose output is the output of the network.

Let's Refer to T as the number of layers in the network(excluding V_0) or the "depth" of the network. The size of the network $\mod V$. the "width" of the network is $max_t \mod V_t$. See Figure Below. A layered feedforward neural network of depth 2, size 10, and width 5 is given for better understanding.

Note that a neuron in the hidden layer has no incoming edges. this neuron will output the constant $\sigma(0)$.

4.2.2 Activation Function

An activation function takes the dot-product mentioned before as an input and performs a certain computation on it. A notable property of activation functions is that they should be differentiable because this property is needed to train the neural network using backpropagation optimization.

Sigmoid or Logistic

It takes a real-valued input and returns a output in range[0,1]

$$\delta(x) = \frac{1}{1 + e^{-1}}$$

sigmoid activation function picture

In Fig, this is an S-shaped curve, and the values going through the Sigmoid function will be squeezed in the range of [0, 1]. Since the probability of anything exists only between the range of 0 and 1, Sigmoid is a compatible transfer function for probability. Although the Sigmoid function is easy to understand and ready to use, it is not used frequently because it has a vanishing gradient problem. This problem is that, in some cases, the gradient gets so close to zero that it does not effectively apply change to the weight. In the worst case, this may completely stop the neural network from further training. Second, the output of this function is not zero-centered, which makes the gradient updates go far in different directions. Besides, the fact that output is in the narrow range [0, 1] makes optimization harder. In order to compensate for the shortcomings, the $tanh()$ function is an alternative option because it is a stretched version of the Sigmoid function, in which its output is zero-centred.

tanh **or hyperbolic tangent:**

it takes real-valued input and produces the results in the range[-1,1]:

$$tanh(x) = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The advantage is that the negative input values will be mapped strongly negative and zeros will be mapped near zero through this function. Therefore, this function is used when classification is needed to be performed between two classes. In practice, this function is preferred over the sigmoid function, but the gradient vanishing problem still exists. so the ReLU function is used

ReLU

Rectified Linear Unit function takes a real-valued input and replaces the negative values with zero

$$R(x) = max(0, x)$$

The ReLU() activation function is trending now in the field of neural networks. It is used in almost all convolutional neural networks or deep learning because it is a relatively simple and efficient function that avoids and rectifies the gradient vanishing problem. The problem with this activation function is that all the negative values become zeros after this activation, which in turn affects the results by not taking negative values into account. A different activation function can be used when we know what characteristics of results we expect to see. Among these three activation functions, we usually start our training process using ReLU() because it works as a general approximator for most data sets.

4.2.3 Multi-Layer Perceptron

there are two types of feed-forward neural networks:

- Single layer Perceptron: the simplest feed-forward neural network with no hidden layers.
- Multi-layer Perceptron: it has more than one layer which is useful for practical application

Multi-layer perceptron can learn not only linear functions but also non-linear functions. The feed-forward neural network in Figure is an example of the multi-layer Perceptron. For a data set containing features and results, the multi-layer Perceptron will learn the relationship between features and results from the given data set, and predict the result for a new data point. Generally in the input layer, we send n numerical inputs through n nodes:

$$x = x_1, x_2, \cdots, x_n | x \in R^n$$

then randomly assign weights for them in the first place:

$$w = w_1, w_2, \cdots, w_n$$

the values for weights will be adjusted later in the training process for more accurate approximations.
In the hidden layer, all the inputs by taking the dot product of x and w, and is known as the pre-state P

$$P = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n + b$$

$$P = \sum_{i=1}^{n} (x_i w_i) + b$$

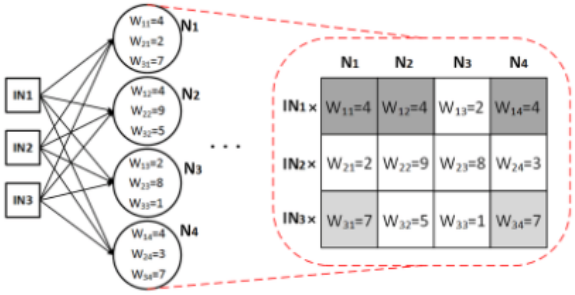

Figure 2.5: Getting Dot-products in Matrix Formatting

Figure 4.1: Dot-products in Matrix Formatting

Let's use vectors to represent them so that it can be treated as a matrix format. In Fig, the input layer has 3 nodes, and the following hidden layer has 4 nodes. Let's create a matrix of 3 rows and 4 columns and insert the values of each weight in the matrix as done above. This matrix would be called W_1. A 1×4 pre-state matrix can be obtained by matrix multiplication. In the hidden layer, four pre-states N_i, each store the dot-product of corresponding inputs and weights. These four pre-state values are ready to go through a certain activation function (). This is called the state S inside this hidden neuron.

$$S = \sigma(W_i IN_i + b)$$

There could be more hidden layers following the first hidden layer, and the state(s) S, which stores the transformed values, will be the input value(s) for the next layer. The matrix contains all the state values that will encounter the next weight matrix and produce new dot-products, and at that time, all the state values become the pre-state values of the next stage. The initial input values will go through every hidden layer in the neural net, repeat the same procedure mentioned above, and

finally arrive at the output layer. The output values we receive are the ultimate state values in the very last hidden layer. This procedure explains how we set up our neural net.

4.2.4 Model Building

A neural network learns from patterns of data and tries to make predictions as accurately as possible. Assume that we already have a set of p data pairs containing the variables and the results, $(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \cdots, (x^{(p)}, t^{(p)})$, where $x^{(i)}$ is input value and $t^{(i)}$ is the target value for i=1, 2, 3, ..., p. We would like to build a neural net F so that ideally,

$$F(x^{(i)}) = t^{(i)}$$

typically for ϵ_i. Let $y^{(i)}$ denote the output of the neural network so that

$$y^{(i)} = F(x^{(i)}) \text{ and } t^{(i)} = y^{(i)} + \vec{\epsilon_i}$$

here $y^{(i)}$ depends on parameters, which are weights and biases, then it turns out as an optimization problem. so we need to set a neural network F that minimizes the error function, which is denoted by E

$$E = \frac{1}{N} \sum_{i=1}^{p} ||t^{(i)} - y^{(i)}||^2$$

where N is the number of training patterns. If it is a two-way classification problem, then N = 2. From this equation, E is a function of the parameters in F, and we need to determine the values of weights that minimize the error by differentiating E. Let's

focus on only one term of the sum, then

$$||t - y||^2 = (t_1 - y_1)^2 + (t_2 - y_2)^2 + \cdots + (t_p - y_p)^2$$

because it is already known that the input and output values are fixed, and the only parameter here is the weight. We can differentiate both sides and get

$$\frac{\delta}{\delta W}(||t - y||^2) = -2(t - y) \cdot \frac{\delta y}{\delta W}$$

from neural network the output is $y^{(i)} = W_{ij}x^{(i)} + b$ Clearly, the output depends on the weight and if we differentiate both sides with respect to W_{ij} using the chain rule, we get

$$\frac{\delta}{\delta W_{ij}}(||t - y||^2) = -2(t_i - y_i)x_j$$

herex_j is the i^{th} coordinate position. This derivative gives us the direction to the maximum, so in order to obtain the minimum point, we follow the opposite direction of this gradient. Additionally, it is desired to see this derivative as close to 0 as possible in order to obtain the minimum error. After figuring out which direction to go, we still need to know how far we go. We do not want it to move too slowly because we would like to finish this training part in an efficient manner. On the other hand, we do not want it to move a step too far; we may face the problem of not converging. Learning rate is an important hyperparameter in gradient descent because it determines how far each step should go. Unfortunately, we cannot analytically calculate a learning rate for a certain data set; we can know it only through trial and error. Typical values for a neural network with standardized inputs (or inputs mapped to the (0,1) interval) are less than 1 and greater than 10^{-6}.

4.2.5 Back Propagation

Backward propagation or backpropagation is the process of propagating the error or loss back to the neural network and updating the weights of each neuron subsequently by adjusting the weight and bias parameters.

Back-propagation plays an important role in the Neural Network. It performs several mathematical operations to learn the patterns between the input and the target variable. The main goal of the neural network is to get the minimum error(loss). We can achieve a minimum error between an actual target value and a predicted target value if we get the correct value of the weight and bias parameters. The error keeps changing with respect to the parameters. This rate of change in error is to be found by calculating the partial derivation of the loss function with respect to each parameter.

By performing derivation, one can determine how sensitive is the loss function to each weight bias parameter. This method is also known as the Gradient Descent optimization method.

Let's start with a simple 1-1-1 neural network, which contains an input x, two stages of weights, w_1, w_2, two stages of biases, b_1, b_2, an activation function $\sigma()$ and an output y.

$$y = w_2\sigma(w_1x + b) + b_2$$

Given a target t, the error of this neural net is

$$E(w_1, w_2, b_1, b_2) = \frac{1}{2}(t - y)^2$$

$$= \frac{1}{2}(t - (w_2\sigma(w_1x + b_1) + b_2))^2$$

to minimize the error, we need to move in the opposite direction of the gradient. Through the training process, we would like to update weights/biases in order to achieve a better error. Suppose we let u denote a generic parameter (either a weight or a bias). Using the gradient descent, u is updated by:

$$u_{new} = u_{old} - \alpha\frac{\delta E}{\delta u} = u_{old} + \alpha\Delta u$$

Where α is called the learning rate, and the change in u is computed via the chain rule on the error. Notice that we incorporated the negative sign into u, because the derivative of the (ty) term will always be negative ty. In particular,

$$\Delta u = -\frac{\delta E}{\delta y} \cdot \frac{\delta y}{\delta u}$$

$$= -(t - y) \cdot -\frac{\delta y}{\delta u}$$

$$= (t - y)\frac{\delta y}{\delta u}$$

Recall the pre-state P (pre-state) and state S mentioned in the prior section,

$$P = w_1x + b_1S = \sigma(P)$$

Now, Let us compute these partial derivatives for all different parameters: for $y = w_2S + b_2$,

$$\frac{\delta y}{\delta w_2} = S \ , \ \frac{\delta y}{\delta b_2} = 1$$

$$\Delta w_2 = (t - y)S \ , \ \Delta b_2 = (t - y)$$

,

for $y = w_2\sigma(P) + b_2$,

$$\frac{\delta y}{\delta w_1} = w_2\sigma'(P) \cdot x \ , \ \frac{\delta y}{\delta b_1} = w_2\sigma'(P)$$

$$\Delta w_1 = (t-y)w_2\sigma'(P) \ , \ \Delta b_1 = (t-y)w_2\sigma'(P)$$

, From the example of a 1-1-1 neural net, we can generalize this to a three-layer neural network in the form of nkm with the activation function σ. Although we could define a different for every neuron, we typically use the same activation function for all the neurons in a single layer. Once that is done, we have to find matrices W_1, W_2 (and more, if we use more layers) and the bias vectors b_1, b_2.

Ideally, much more data is needed in order to get good estimates. The key idea is that once we have the derivative of the sum of squares error with respect to the weights, we can adjust the weights accordingly through the training process.

4.3 stochastic gradient descent

Stochastic Gradient Descent Method [9]

4.3.1 Gradient Descent

The gradient of a differentiable function $f : R^d \to R$ at w, denoted $\nabla f(\mathrm{W})$, is the vector of partial derivatives of f.

$$\nabla f(w) = \left(\frac{\partial f(w)}{\partial w[1]}, \cdots, \frac{\partial f(w)}{\partial w[1]} \right)$$

.

A gradient is an iterative algorithm. let's start with initial value of w, $w^1 = 0$. then, at each iteration, take a step in the direction of the negative of the gradient at the current point. that will be the updated step

$$w^{t+1} = w^t - \eta \nabla f(w^t)$$

, $\eta > 0$ will be discussed later. Since the gradient points in the direction of the direction of the greatest rate of increase of around w^t, the algorithm needs to make a small step in the opposite direction, thus reducing the value of the function. After T iteration the algorithm outputs the averaged vector,

$$\bar{w} = \frac{1}{T} \sum_{t=1}^{T} w^t$$

The output could also be the last vector, w^T, or the best-performing vector. However, taking the average would be better, especially when generalising gradient descent to non-differentiable functions and the stochastic case.

4.4 Neural Network Classification Model

These are the Steps followed here to train the neural model for better accuracy

Input: X, y (features and labels), X_{train}, X_{test}, y_{train}, y_{test} (training and testing splits) Initialize neural network model *model* Add layers to *model*: 64-neuron dense layer with ReLU activation, 64-neuron dense layer with ReLU activation, 4-neuron dense layer with softmax activation function Compile *model* with Adam optimizer and sparse categorical crossentropy loss Train *model* on X_{train} and y_{train} for 50 epochs with batch size 32 and 10% validation split Evaluate *model* on X_{test} and y_{test} to calculate loss and accuracy

- **Data Preparation** Splitting dataset into features (X) and target (y), then further splitting them into training and testing sets using train_test_split.
- **Model Definition**Defining a neural network model using the Sequential API. This model consists of three Dense layers. The first two layers have 64 neurons, each with ReLU activation function, and the final layer has 4 neurons with softmax activation function, suitable for multiclass classification.
- **Model Compilation**Compiling the model using the Adam optimizer and sparse categorical crossentropy as the loss function and specifying accuracy as the metric to monitor during training.
- **Model Training**Training the model on the training data (*X_train, y_train*) for 50 epochs with a batch size of 32. Additionally, using 10% of the training data as a validation set during training.
- **Model Evaluation**Finally, evaluate the trained model on the test data (X_test, y_test) and obtain the loss and accuracy scores.

Table 4.1: Training Neural Network Model

epoch	**Batch size**	time	validation split	Loss	Accuracy
500	256	7m	0.1	0.139866	0.9316594
500	64	140m	0.1	0.12724	0.937797
100	128	14m	0.1	0.13216	0.9351769
100	64	28m	0.1	0.131524	0.935428
100	16	110m	0.1	0.181495	0.9097726
50	128	7m	0.1	0.15372	0.922419
50	32	28m	0.1	0.1405129	0.933256
50	32	27m	0.2	0.1404415	0.932207
50	32	23m	0.1	0.134115	0.93225824

4.5 Neural Networks Results

For all the features converted to 0 to 1 as input and target variable hot encoded, different neural network parameters were tweaked to get a better model. See Table 4.1. it shows 32 batch size and 50 epochs will be enough with 10% validation split. The confusion matrix explains about the true negative, true positive, false negative and false positive prediction for the completely unknown data set with around four lac data points.

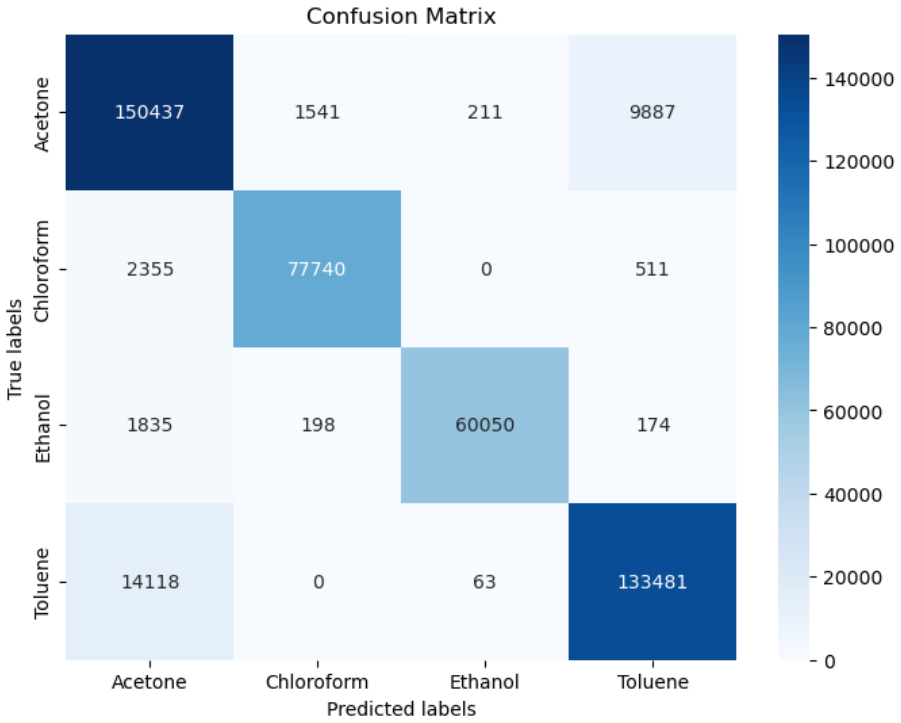

Figure 4.2: Confusion Matrix for trained neural networks on New Data set

Chapter 5

Multi-Sensor Array

The ML analysis was recorded for three sensors, and each sensor was maintained at a sufficiently high temperature in order to achieve a decent sensor response. However, each sensor material (oxides in this case) may have its optimum temperature at which the response would be highest, and this optimum response temperature need not be the same for all materials used. Therefore, compared to the previous method of measurement, where all the sensors are kept at the same temperature (See Appendix C), a new setup was designed and built wherein a sizable number of sensor arrays (6 nodes) may be accommodated. The temperature of each can be controlled individually such that each of them is at its optimal temperature. Besides, one can also program the heater power supply to the desired waveform to tune the sensor response dynamically, and that could be used as an effective control feature of ML data.

The new setup built is shown in Fig 5.1. It can accommodate up to six sensors simultaneously, as seen in the figure. A metallic chamber containing two layers where

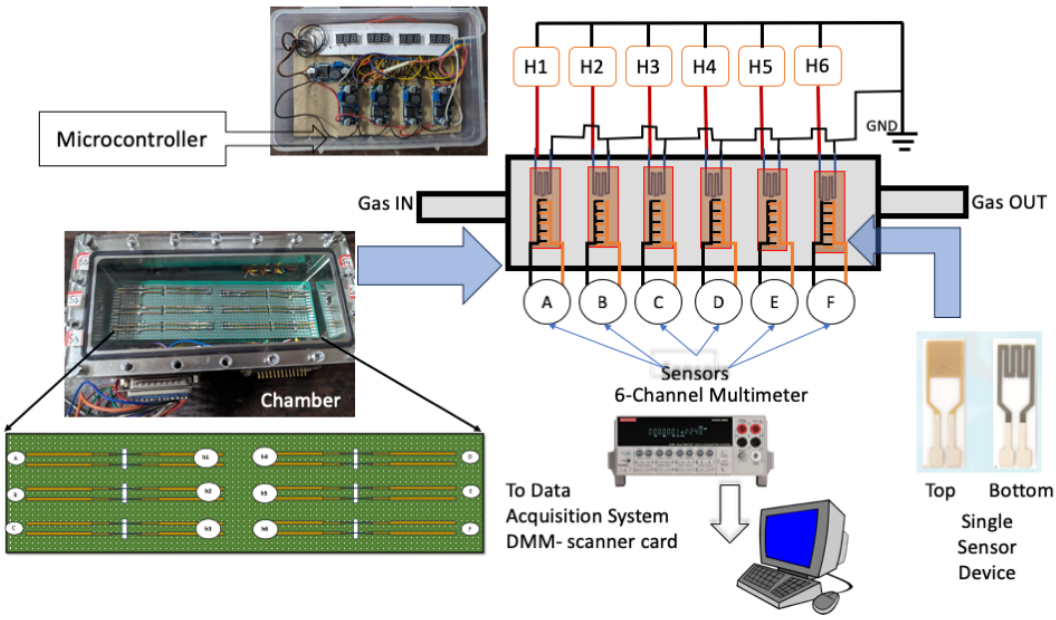

Figure 5.1: Schematics of six sensor arrays with individual heater control using microcontroller and the sensor output leads (Interdigitated gold electrodes) to simultaneously measure sensor response of 6 nodes. The typical single-sensor device is shown with top electrodes and a bottom heater.

one PCB Snaps on the other and holds the top PCB consist of six-hole slits where alumina samples with deposited heaters can be slid. On each side horizontal to the PCB slit, two pogo pins were used to make contact with the alumina sample electrode or heater, exploiting the spring-loaded mechanism of pogo pins. Similarly, six samples may be fitted in a single top PCB. The other side of the PCB contains 2.54 mm pins, which simply snap onto the bottom PCB. The Bottom PCB is fixed inside the chamber with all the wiring and connection to a 25-pin connector side to the wall of the chamber, as shown in the figure. A transparent lid is used to enclose the chamber from the top. The metal chamber has inlet and outlet pipes to achieve gas flow.

The whole assembly consist of six sensors and six heaters. The Pt deposition on the alumina substrate on one side acts as a heater with 6Ω resistance, ensuring precise and local heating. On the other hand, the opposite side of the alumina provides the surface for the sensor substrate. A multi-channel voltage supply was also constructed using a parallel six-buck converter to drive the heater at a constant temperature individually. The main voltage source for all the buck converters was a 24 V-5 A SMPS (Switch Mode Power Supply). The buck converter gives the freedom to adjust the constant voltage individually by changing its potentiometer. Hence temperature of the each sensor can be controlled, for better utilization a micro controller can be used to set the temperature simultaneously for each sensor by changing the voltage on buck converter. The heater can go up to $450°C$ at 12 V. So the Bottom PCB inside the chamber provides the interface to connect all the sensors to the voltage supply source and multi-meter to measure the resistance.

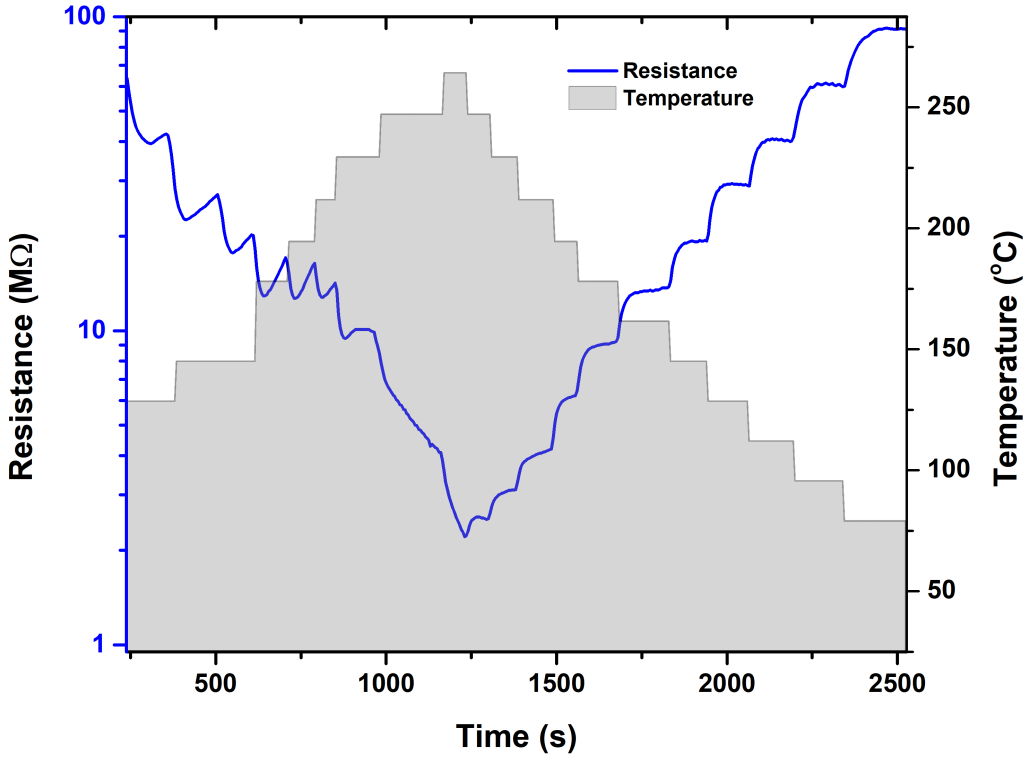

Figure 5.2: The data recorded with the assembly described in Fig 5.1 for a single sensor using a controller made. (measurement courtesy Ms. Niharika)

Chapter 6

Conclusion

- The aim of the project involves multi-sensor data analysis and designing a set-up to characterise devices for the identification of gases.
- The data collected with 3 sensor arrays was analysed using machine learning techniques.
- The main method for analysis relied on the dimensional reduction method to visualise data and ML methods for the classification of the data.
- The comparison of the classification results using various methods like Decision tree, Random forest and NN was performed and the same is listed in Figure 6.1.
- Almost all training data could be predicted with an accuracy of above 90%. However, the decision tree model gave the fastest result compared to other models.

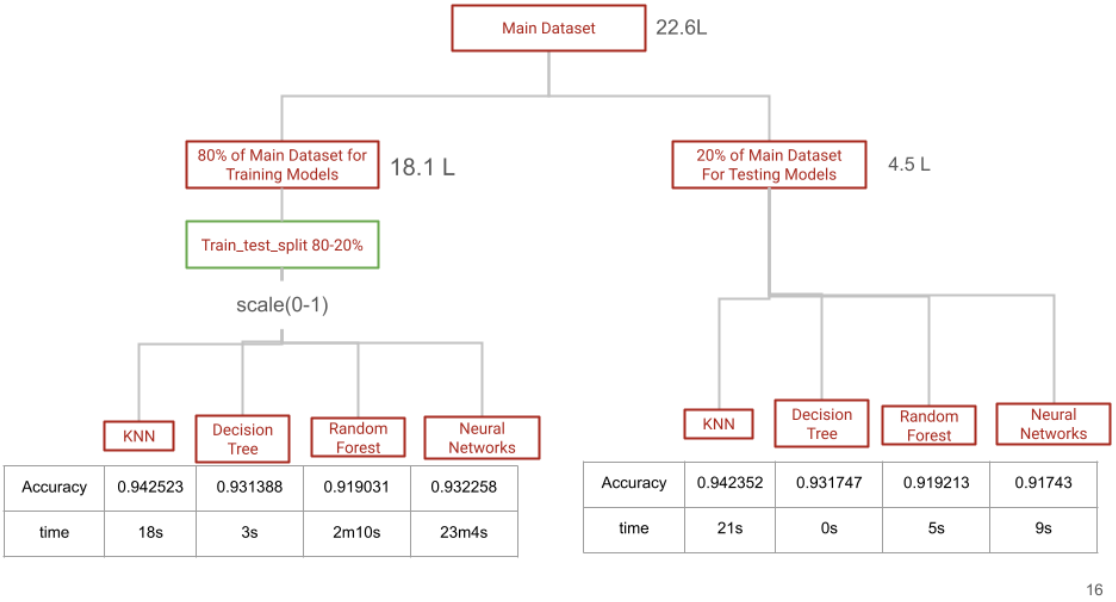

Figure 6.1: Model Comparison on training and unknown data set

- A new setup that can record the response data for six sensors simultaneously, with individual heater control for each sensor, has been designed and developed.

Appendices

Appendix A

Data collection of three simultaneous sensors

The electrical resistance of the thin films was monitored in reaction to different volatile organic compounds (VOCs) at constant operating temperatures in order to conduct gas-sensing studies. Under dynamic flow conditions set by mass flow controllers with different capacities, the sample gases were infused. The in-house gas sensing system employed in this study is shown in Figure 1. In order to assess the gas sensing characteristic in a detecting chamber, the films were placed on a sample holder that could achieve 400°C utilizing a heater underneath the sample holder (manufactured by Excel Instruments, India). The temperature of the sensor was determined using a sample holder with a type-K thermocouple installed. For measuring sensor resistance, an alumina substrate with interdigitated gold electrodes is used. The sensor resistance was measured using a Keithley 6517B electrometer connected to a workstation by providing two probes with a constant bias voltage of

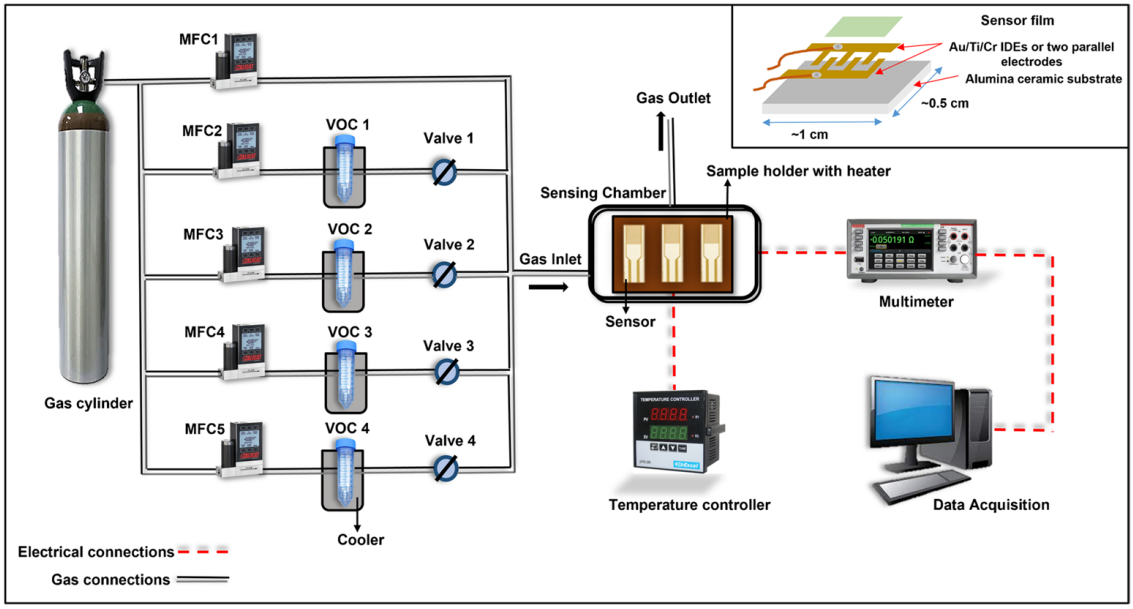

Figure A.1: The schematic diagram of the gas sensing setup used for the experiment [11]

10 V. With a tolerance of 1 fA, it is a high-resistance analyzer that could contribute meaningfully to 1015 ohms. To evaluate the sensor sensitivity to ethanol and other volatile organic compounds, the deposited films need to be exposed to the relevant vapours diluted in the air. [11]

Appendix B

Building Thermal Expansion Measurement System and its interfacing

B.1 Building Thermal Expansion Measurement System and its interfacing

B.1.1 The principle of measurement system

The coefficient of linear Thermal expansion (α) denotes the change in length of the given material when heated. It is a material property and the same may be written as,

$$\alpha = \frac{\frac{L_f - L_0}{L_0}}{T_f - T_0}$$

The quantity is of high significance for studying the thermal properties of materials for fundamental as well as applied areas. A couple of methods have been reported for the measurement of TEC. Those can be broadly classified as-

- Interferometry - light interference based
- Diletometry
- Capacitance based
- LVDT-based

In the case of interferometric methods, the sample surface is shone with a monochromatic light beam, and the interference fringes are observed from the rays reflected on the surface. Although this method has high precision, it is applicable to only a limited range of α values and depends on the sample surface optical properties.
On the other hand, in dilatometric methods, the change in length of the material is measured by means of a linear translation rod/arm assembly that changes the signal on the capacitance bridge or an LVDT, respectively. We have developed a system based on this method using a Linear Variable Differential Transformer (LVDT) as a linear transducer.

B.1.2 Instrumentation

The entire system is home built and the schematic diagram of the system is shown in Fig 2.8. The sample (1) to be measured is sandwiched between two silver metal blocks (2 and 3) for to hold in place, as well as silver helps in better temperature

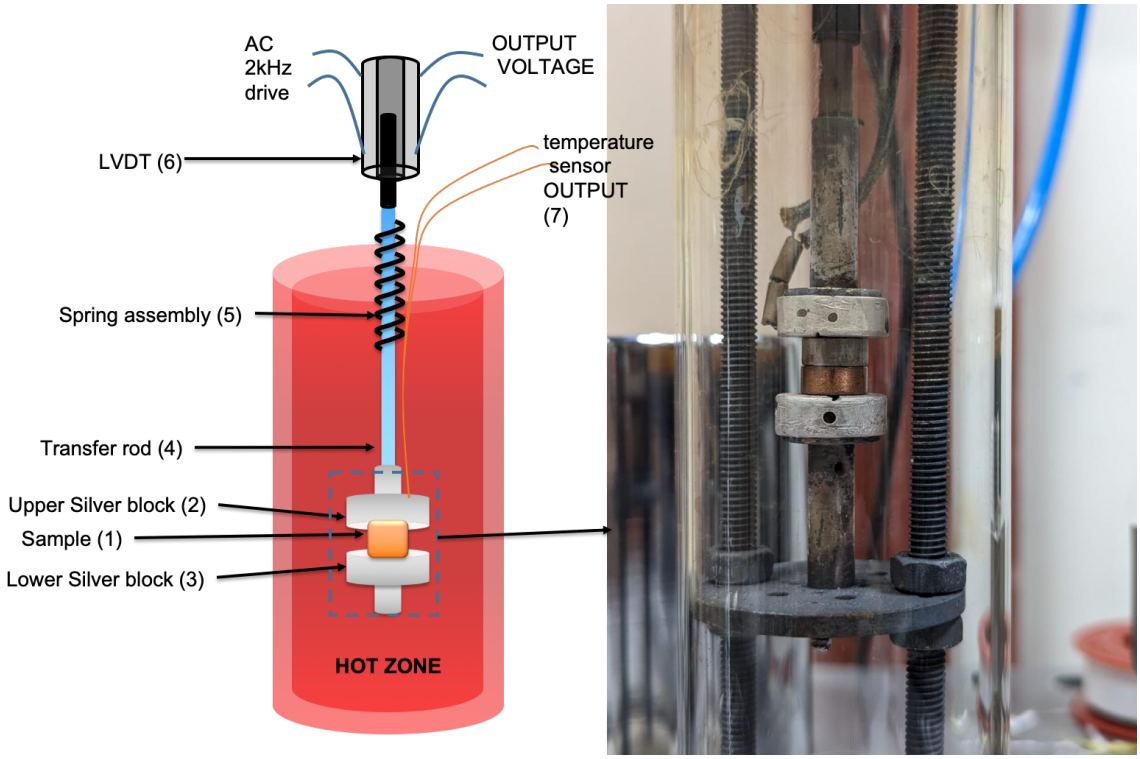

Figure B.1: The schematic diagram of the home built Thermal expansion measurement system and its digital photograph.

equilibration. The lower silver block is fixed while the upper silver block is mounted on a transfer rod/arm (4) which is made up of steel (not modified to Quartz). For this top silver block to always press against the sample, it is equipped with a spring assembly (5) that always pushes it against the sample. This ensures that any change is sample length is directly translated to the other end of the rod where the transducer is place. As mentioned above, LVDT is used here as a transducer as shown in the figure. The enlarged view of sample holder assembly is also shown in Fig 1. The entire sample holder goes inside a vacuum chamber made of the quartz tube with suitable couplers and flanges. The Quartz tube is inserted into the furnace (hot zone) to raise the sample temperature uniformly. A k-type thermocouple is inserted in upper silver block to measure the sample temperature. The electronic components are place away from the hot zone. If the sample is placed, it is expected to show a certain signal from the LVDT arm. If its temperature is raised and it expands the change is length of the sample caused the quartz rod to push the core of LVDT further inside producing a change in signal.

Design and working of LVDT

An AC power source of 1.2 V at 2 kHz sine wave was used to power the LVDT, and a Keithley 2700 Multi-meter cum data acquisition system was used to read its analog outputs. A hollow cylinder of insulating material serves as its main component. This insulating cylinder has one main winding P and two secondary winding A and B looped around its circumference. In the middle of the insulating cylinder lies the main winding P, and on either side of it are two secondary winding, A and B, coiled in complete opposition to one another as shown in Fig 2.9 In other words, A and B are moving in opposite directions. A magnetic or armature core is put within the

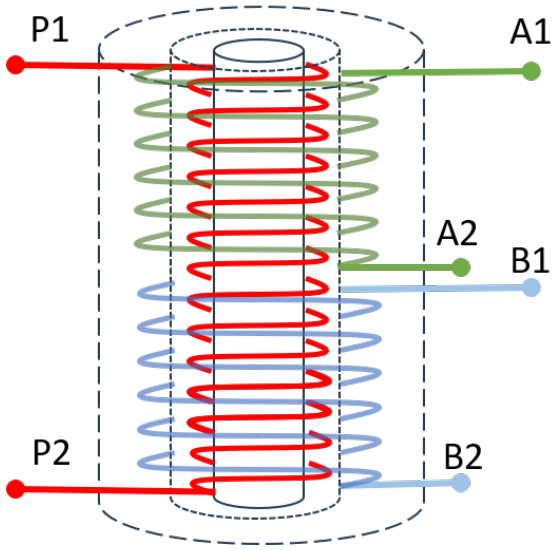

Figure B.2: Circuit diagram of LVDT primary winding and Secondary winding.

insulating hollow cylinder and can move freely in both directions. Attaching the soft iron core to the object under study allows for the measurement of displacement. Nickel is commonly used in soft-core because of its great sensitivity. [9]

From Faraday's law of electromagnetic induction, an EMF is produced in the secondary winding. When the primary winding is supplied with an alternating current, a magnetic flux is generated that travels through both A and B. Primary winding flux is proportional to secondary winding conductor count. The primary coil is powered using an AC signal (here we used 2 kHz and 1.2 V). The magnetic induction into the secondary develops due to magnetic core placed in between (like a transformer). When the core of LVDT may slide to shifts to top, bottom or to the null position. There will be an increase or decrease in the output difference. In this case, the LVDT's output voltage is a linear function of core displacement up to a certain point (5 mm from the LVDT's null position limit). This graph displays the variation in output voltage versus displacement. (See Fig 2.10).

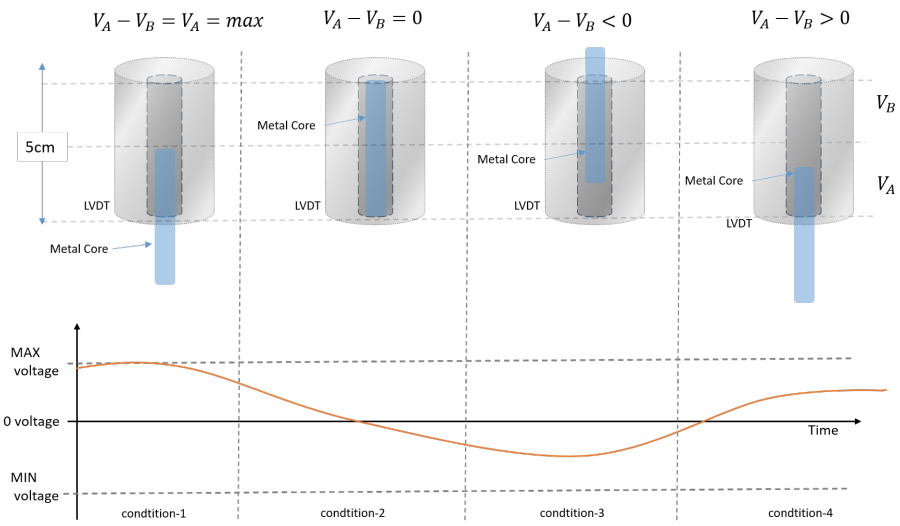

Figure B.3: Different placements of metal core inside LVDT and corresponding voltage signal being generated.

Measuring TEC requires monitoring the output voltage of the LVDT that changes when the temperature of the sample under consideration varies. With increment in temperature, material expands and so the connected shaft to the metal core inside LVDT moves, which gives change in voltage proportional to the change in position of the metal core. Therefore, LVDTs can be consider as length or position sensors that detect changes in length or position.

In order to measure the TEC a change in length of material with change of temperature is to be measured, and initial length of the sample should be known. With a good sensitive LVDT even a small expansion (sub mm) can also be measured accurately. Thus, the final system measurement should give the plot between change in length vs temperature. Change in length becomes change in Voltage by using LVDT's sensitivity factor (typically given in mV/mm).

Let V_A and V_B be the voltages produced by the A and B coil, respectively. Since

A and B are connected in opposite directions, they must be connected in series to create a single voltage with a phase difference of 180 degrees. Because of this, the secondary coil's output will be the difference between the two voltages produced by the device.

$$V = V_A - V_B$$

The secondary coil's voltage output is linear for small displacements, as seen in Fig 12. There are no discrete steps in the voltage output, and the resolution is more a function of the testing apparatus than the transducer itself. No further intermediary amplifiers are required since the output voltage is large and easily measurable. The transducer can withstand significant vibration and stress because of its high sensitivity. Most importantly, the measuring system is insensitive to changes in temperature and suffers no loss due to friction. This attribute is necessary because of its proximity to a high-temperature furnace.

B.1.3 Calibration and Measurement

Fig13— The preliminary measurements performed with the system are shown in Fig. The system background data was measured without any sample and shown in Fig (a). because of the component used for sample holder there were non-zero background signal measured. Because of its low value there is a significant noise in the background. Similarly, the data was also measured by placing a sufficiently long Cu sample(s) between the two silver holders. This yielded about one order of magnitude large change in the voltage. Cu is used here for standardization for its known and high value of TEC. Nevertheless, there are several non-monotonous changes in the voltage that are not expected.

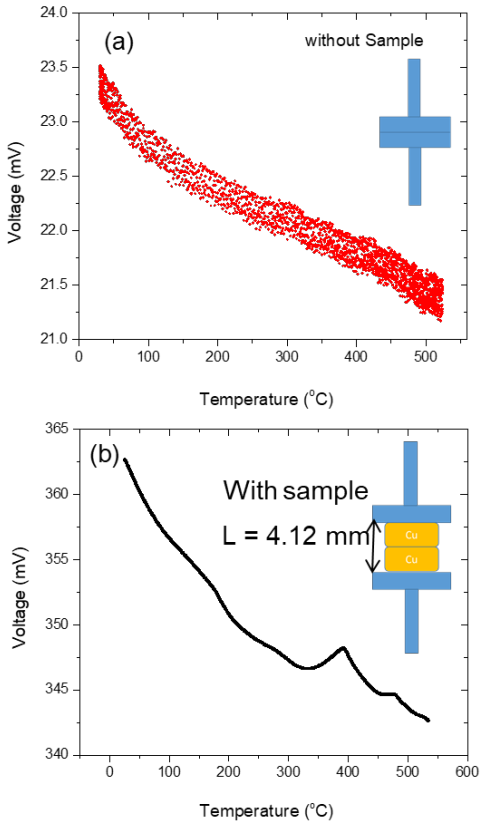

Figure B.4: (a) Without and (b) With sample thermal expansion measurement data. Two Cu samples in the stack (each with L = 4.12mm) were used as samples.

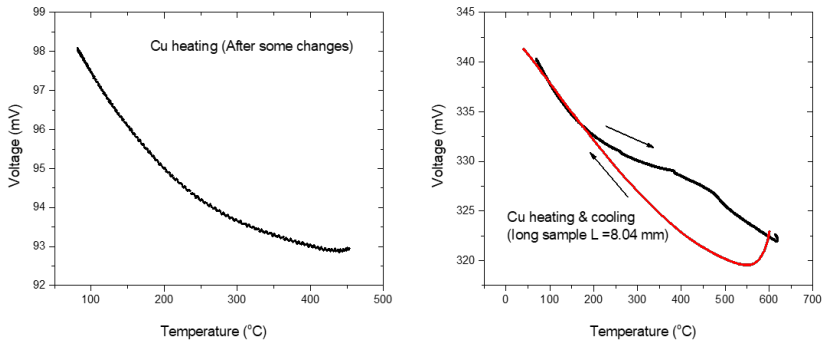

Figure B.5: Measurements performed on the (a) same Cu sample, L= 4.12mm and (b) longer sample (8.04 mm length Cu sample)

Further, optimization of the system needs dynamic measurements. Also, the final data is to be represented as L/L. Here, copper has very high thermal expansion coefficient. However, for insulating samples like ceramics etc. the TEC value is low and hence the is of the order of background. In that case, it will be challenging to analyze the result when signal to noise ratio is low.

B.2 Thermal Expansion Temperature v/s Voltage

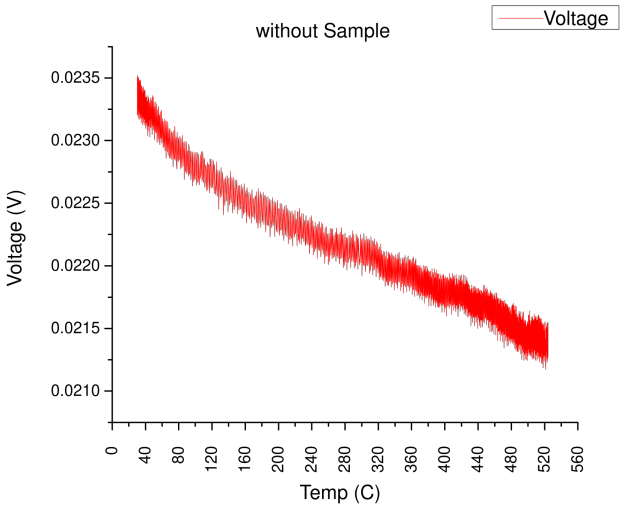

Figure B.6: without sample thermal expansion measurement

Bibliography

[1] B. Acock and G. W. Wall. "A Simple Conductimetric CO2 Analyzer With Automatic Recalibration: I. Design, Implementation, and Functionality". In: *Agron. J.* 87 (1995), pp. 70–75. DOI: 10.2134/agronj1995.00021962008700010014x. URL: https://scholar.google.com/scholar?q=Acock%2C+B.%3B+Wall%2C+G.W.+A+Simple+Conductimetric+CO2+Analyzer+With+Automatic+Recalibration%3A+I.+Design%2C+Implementation%2C+and+Functionality.+Agron.+J.+1995%2C+87%2C+70%E2%80%9375&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[2] Gareth James et al. *An Introduction to Statistical Learning*. Vol. 112. New York: Springer, 2013.

[3] J. D. James et al. "A Review of Measurement Techniques for the Thermal Expansion Coefficient of Metals and Alloys at Elevated Temperatures". In: *Measurement Science and Technology* 12.3 (2001), R1.

[4] Rappal Sangameswara Krishnan, Ramachandran Srinivasan, and S. Devanarayanan. *Thermal Expansion of Crystals: International Series in the Science of the Solid State*. Elsevier, 2013.

[5] Caren Marzban, Ulvi Yurtsever, and Michael Richman. *Principal Component Analysis for Equation Discovery*. 2024. arXiv: 2401.04797 [stat.ME].

[6] Sidharth Prasad Mishra et al. "Multivariate Statistical Data Analysis-Principal Component Analysis (PCA)". In: *International Journal of Livestock Research* 7.5 (2017), pp. 60–78.

[7] Cari Prodi and Try Out. *Eigenvalues and Eigenvectors*.

[8] J. W. Severinghaus and P. B. Astrup. "History of Blood Gas Analysis. III. Carbon Dioxide Tension". In: *J. Clin. Monit.* 2 (1986), pp. 60–73. DOI: 10.1007/BF00759191. URL: https://scholar.google.com/scholar?q=Severinghaus%2C+J.W.%3B+Astrup%2C+P.B.+History+of+Blood+Gas+Analysis.+III.+Carbon+Dioxide+Tension.+J.+Clin.+Monit.+1986%2C+2%2C+60%E2%80%9373&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[9] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning*. Cambridge University Press, May 2014. ISBN: 978-1-107-05713-5. URL: http://books.google.ie/books?id=Hf6QAwAAQBAJ&pg=PR4&dq=ISBN978-1-107-05713-5&hl=&cd=1&source=gbs_api.

[10] K. Shitashima. "Evolution of Compact Electrochemical In-Situ pH-pCO_2 Sensor Using ISFET-pH Electrode". In: *Proceedings of the Oceans 2010 MTS/IEEE Seattle*. Seattle, WA, USA, Sept. 2010. DOI: 10.1109/OCEANS.2010.5664545. URL: https://scholar.google.com/scholar?q=Shitashima%2C+K.+Evolution+of+Compact+Electrochemical+In-Situ+pH-pCO2+Sensor+Using+ISFET-pH+Electrode.+In+Proceedings+of+the+Oceans+2010+MTS%2FIEEE+Seattle%2C+Seattle%2C+WA%2C+USA%2C+20%E2%80%9323+September+2010&hl=en&as_sdt=0&as_vis=1&oi=scholart.

[11] S. Singh et al. “Metal oxide-based gas sensor array for VOCs determination in complex mixtures using machine learning”. In: *Microchimica Acta* 191.4 (2024). Published online: March 13, 2024, pp. 1–10. DOI: 10.1007/s00604-024-06258-8. URL: https://doi.org/10.1007/s00604-024-06258-8.

[12] Sachin Ashok Sonawane and M. L. Kulkarni. “Optimization of Machining Parameters of WEDM for Nimonic-75 Alloy using Principal Component Analysis integrated with Taguchi Method”. In: *Journal of King Saud University-Engineering Sciences* 30.3 (2018), pp. 250–258.

[13] Muhsang Sudadama. *Advanced Linear Algebra (Third Edition) by Steven Roman*. Nov. 2023. URL: URL.

[14] Yeram Sarkis Touloukian et al. *Thermal Expansion: Metallic Elements and Alloys*. 1975.

[15] Paul Wilmott. “Machine Learning: An Applied Mathematics Introduction”. In: *Machine Learning and the City: Applications in Architecture and Urban Design*. 2022, pp. 217–248.