NATIONAL UNIVERSITY OF SINGAPORE

FACULTY OF ELECTRICAL ENGINEERING

AY 2021/2022

EE4002D PROJECT CAPSTONE

Individual Portfolio:

Odor localization via a low-power gas sensing network for Internet-of-Things application

Done by:

Liu Xiaohan
A0177828L

Table of Contents

1 Introduction

With the development of the network, IoT systems have become a popular development topic in the area of industry and manufacturing. This project will aim for developing an IoT system in odor localisation. The essence of this project is to build an Internet of Things system that uses the raw data collected from gas sensors for machine learning and presents the calculated location of the odor source to the end user.

2 Work Summary

This Portfolio describes the detail work done for the project on the tests and failures encountered to achieve the targeted success. In the previous semester report, the background and IOT system setup of this project has been introduced. This individual portfolio would focus more on the machine learning part of the project. For ease of understanding and smooth information flowing, this portfolio will explain the project for initial hardware setup and machine learning part in the following sequence:

1. Timeline and Achievement
2. Software and Hardware Tools used
3. Raspberry Pi Setup
4. Data Pre-Processing: 120 Features
5. Algorithm Development

3 Timeline and Achievement

	Timeline	Achievement
Semester 1	Week 1-5	• Project administration • Background research
	Week 6-10	• Literature paper review • Meet with supervisor to discuss future plan • Wait for equipments to arrive
	Week 11	• Raspberry pi setup
	Week 12-13	• CA1 report and presentation • Exam period
Summer Vacation		• Communication between raspberry pi and the sensors • Search information for data pre-processing
Semester 2	Week 1-3	• Generate plot for raw data with MATLAB • Data pre-processing • Feature engineering
	Week 4	• Machine learning with MATLAB, choosing the most appropriate model
	Week 5	• Generate Bagged Tree algorithm with python
	Week 6-8	• Train and modify model with different label types • Decide new features for dataset • New feature engineering

	Week 8-10	• Tuning parameters for BaggedClassifier • Research and experiment more ways to improve algorithm's accuracy • Output trained-well model for real-time predict
	Week 11-12	• Research and experiment more ways to improve algorithm's accuracy • Trained data with RNN and Gradient Boosting
	Week 12-13	• Compose final thesis and presentation

4 Software and Hardware Tools used

Hardware Tools	**Usage**
Raspberry Pi	The data transmission and processing centre of the IOT system.
Nordic Thingy52	Collect and transmit odor concentration data.
Software Tools	**Usage**
MATLAB	Visualize a large amount of raw data and assist algorithm selection through built-in functions.
Python with Jupyter Notebook	Implement the machine learning algorithm with dataset and trained out models.
HTML	Display the location of the odor source on the web page.

5 Raspberry Pi Setup

Raspberry Pi is an ARM-based microcomputer motherboard with SD card as the memory hard drive. In addition to connecting hardware devices through ports, the Raspberry Pi also supports Bluetooth and Wi-Fi transmission. In this project, the Raspberry Pi will receive the data from Nordic Thingy 52 via Wi-Fi and provide a python platform to process the data with machine learning. In addition, the final webpage result display will also be done through the Raspberry Pi.

Since the Raspberry Pi uses an SD card as the memory hard disk, the operating system needs to be written to the SD card in advance before using the Raspberry Pi. Raspberry Pi OS (Raspbian) from the official website is chosen from various operating systems. After writing the operating system into the SD card, Raspberry Pi needs to be connected with a monitor, mouse and keyboard to boot and display the graphical interface. As data may be collected and transmitted to Raspberry Pi in different environments, connecting with a monitor every time is not convenient.

Therefore, VNC is chosen to remotely operate Raspberry Pi with a laptop. To connect to VNC, Raspberry Pi should connect with Wi-fi first. To connect Wi-fi without monitor and mouse, a conf file and an empty ssh file should be written to boot in the SD card to initialize the Wi-fi setting. The conf file in Linux form is shown as Figure 1 below.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=SG

network={
	ssid="SINGTEL-GJC3"
	psk="fxfhkc8aax"
	key_mgmt=WPA-PSK
}

network={
	ssid="SINGTEL-GJC3(5G)"
	psk="fxfhkc8aax"
	key_mgmt=WPA-PSK
}

network={
	ssid="Lxh"
	psk="20000323"
	key_mgmt=WPA-PSK
}
```

Fig. 1. Config file for Wi-fi initializing

Moreover, after getting the IP address of Raspberry Pi from the router terminal website, the IP address can be set in Putty to use the VNC viewer.

A total of three Raspberry Pi is used in this project. After setting up each Raspberry Pi, the odor concentration is sent from the corresponding Nordic Thingy52, and raw data is generated.

6 Data Pre-Processing: 120 Features

Raw data needs to be pre-processed before being used as actual training data because of problems like missing data or outliers. Data pre-processing is necessary for providing meaningful data as machine learning model inputs [1].

Feature engineering in machine learning is more than selecting the appropriate features and transforming them. Not only does feature engineering prepare the dataset to be compatible with the algorithm, but it also improves the performance of the machine learning models.

Due to the large amount of data, experimental equipment, and environmental errors, some of the data will have unreasonable values. For example, in the figure2 below, the concentration curve is so unsmooth that it will affect training.

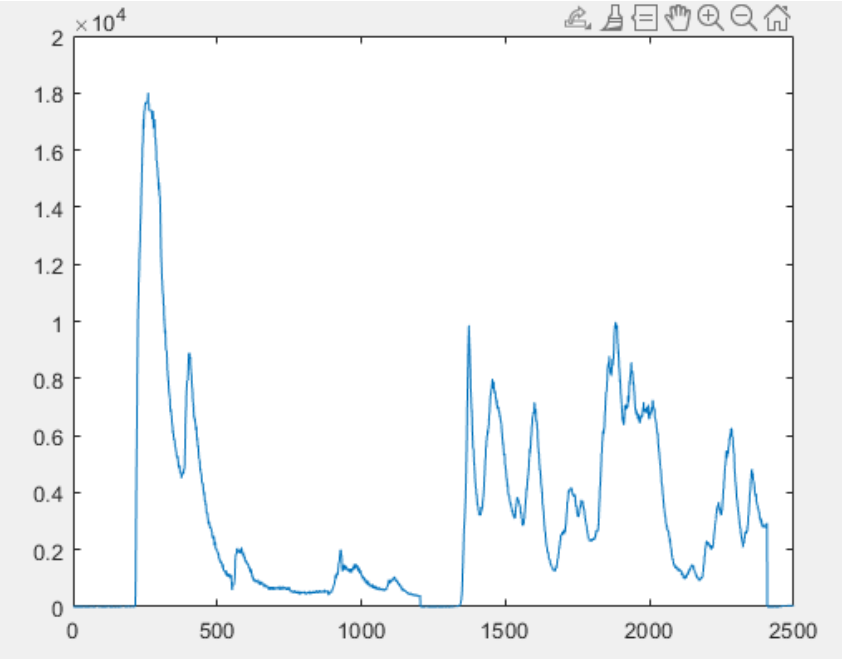

Fig.2. Example of Plot for Raw Data

From the plot generated by MATLAB, the 20-minute data generally has a trend of increasing first and then decreasing, so the data of different time periods have different characteristics. Since this is a time series data, we initially thought that the bagged tree could not make judgments on continuous time data. Therefore, the data is manually determined to be 5 minutes due to a set of original concentration plots generated by MATLAB as figure2.

However, simply using the concentration within five minutes as all features may cause the number of features to be too large to be not able to train and to be too inefficient. Therefore, at the beginning, the original concentration data was not directly used for training, but a total of 120 features are used. The features not only include some basic characteristic for data series: max, min, average, median and variance, but also peaks time, peaks value and prominence.

Since there are a lot of peaks in variety height and width in raw data series, shown as figure2, the position, width, gradient, and height of peaks may be important features, however, the built-in parameter prominence of the scipy.signal package covers the gradient and width information of peaks, so it is adopted. After calculating the four sets of characteristics every 20 minutes, these four sets of characteristics have been added and subtracted to obtain new compared features.

The following table1 shows our consideration when deciding on the features.

• maximum, minimum, average and variance
The maximum, minimum, average, variance and their corresponding time are calculated every 5 minutes regardless of any missing data within the 5 minutes. If data missing is more than 5 minutes, the set of collected data is discarded.
• highest peak value and prominence
The highest peak value and its corresponding time and prominence within every 5 minute is computed using find_peaks function from scipy.signal package. Here prominence is a measure of how much the peak stands out due to its intrinsic height and its location relative to other peaks.
• maximum difference and average difference
The maximum difference and average differences are difference between any 2 of the 3 sensors.
• Regions

Besides the coordinates of odor source, the 2.1m*2.1m room is categorized into twenty-five 42cm*42cm square regions with labels (region 1, region 2…) for clearer learning process.

Table1. Consideration of Features

The dataset after data preprocessing is a n*126 table which contains both input/features and output/regions & coordinates as illustrated table2 below:

data set no.	sensor 1	0-5 min		6-10 min		11-15min		16-20min		sensor 2	…	sensor 3	…	x_coordinate	y_coordinate	region
		average	…	average	…	average	…	average	…							
1																
2																
3																
4																
5																
…																

Table2. Data Structure for 120 Features

As such, the preprocessing processes and the ideology can be illustrated in the figure3 below:

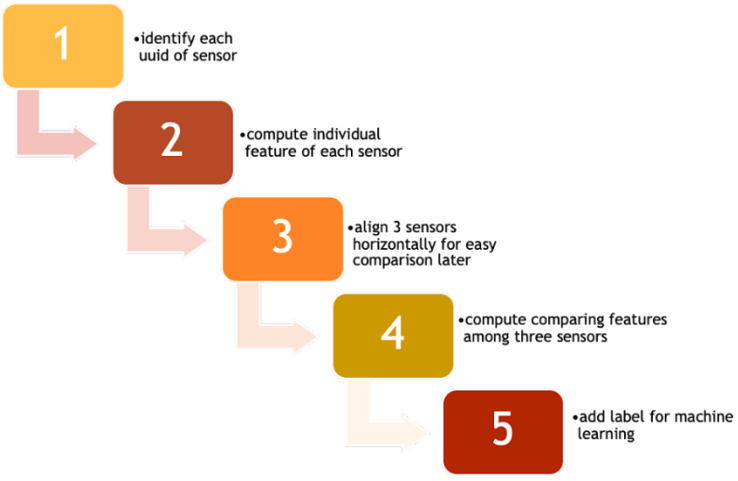

Fig.3. Preprocessing Processes of Data

The detailed 120 feature processing code is shown in appendix1.

7 Algorithm Development

7.1 Algorithm Choose

7.1.1 Bagged Tree

MATLAB provides a fast and convenient way to find suitable machine learning models. The above dataset is imported into MATLAB as a table. By using Regression Learner and Classifier in MATLAB, the dataset is split into features and target, the performance of different models is tested and compared.

Among all the models, the bagged tree model is chosen as it gives the best performance in terms of lowest root mean square error (RMSE) and the predicted location distribution.

The bagged tree is an ensemble algorithm based on a decision tree. Ordinary decision trees are easily changed by small amounts of data changes, so that overfitting is prone to occur. The bagged tree is to randomly select the samples in the training set multiple times and take the average from the multiple training decision trees, which reduces the overfitting impact brought by the decision tree. However, in this project, overfitting not only comes from the decision tree algorithm itself, but also from too little data and too many features, which will be explained later in part Number of Feature. The below figure4 is a clear illustration of the mechanisms that a bagged tree uses.

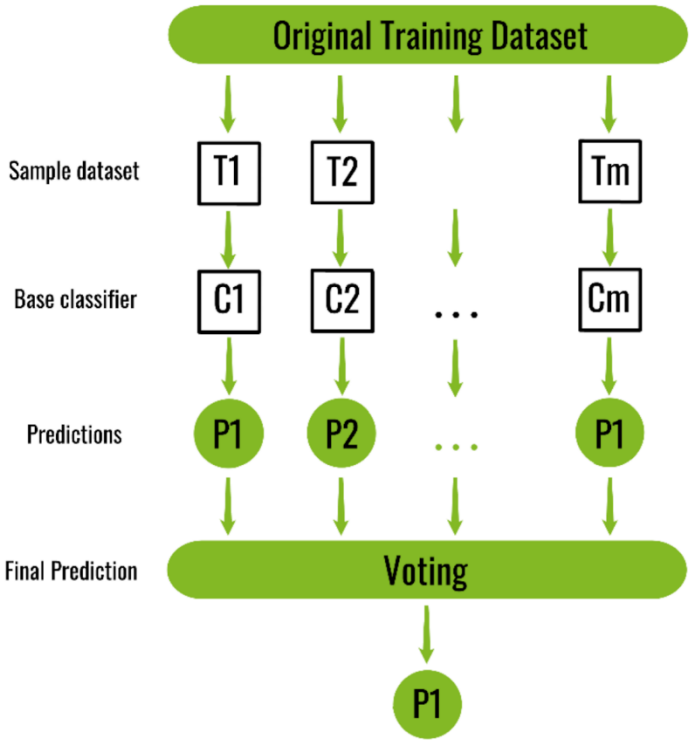

Fig.4. Mechanisms of Bagged Tree [2]

Due to such working mechanisms of bagged trees, the decision trees are sensitive to the data. If the training data is changed, the resulting decision tree can be quite different, so that when the number of training data sets increases, the final model as well as the accuracy of the prediction is varying.

7.1.2 Library Package in Python

Since the focus of this project is not to develop the model algorithm itself from the foundation, the existing model algorithm used from the rich python library package. Common machine learning function libraries include pandas, tensorflow, scikit-learning, etc. Scikit-learn is developed by a professional team of experts, it covers most machine learning models, and a

large number of parameters of each model can be adjusted and have detailed document explanations. Scikit-learn is selected as the model training library because it provides high-quality model algorithms and easy-to-understand operating procedures.

Since there is no special requirement for the division of training and testing data, here choosing a common splitting of training and test set into 70%/30%.

The general training and testing process by Scikit-learn is shown as follows:

```
import function and libraries needed
file = read ( 'dataset.csv')
#split training and testing data as 70%/30%
X = according feature columns in file
Y = according feature columns in file #three kind of y in total
X_train = 70% random samples in X
Y_train = according labels for X_train
X_test = X - X_train
Y_Test = Y - Y_train
#training by scikits-learn models
Model = bagged_trees.fit(X_train,Y_train)
#testing
Y_pred_test = Model.predict(X_test)
Y_pred_train = Model.predict(X_train)
#Calculate Accuracy and RSME
Test Accuracy and RSME = Compare Y_Test and Y_pred_test
Train Accuracy and RSME = Compare Y_Train and Y_pred_train
```

7.2 Output Labels

In the dataset, there are three columns which can represent the location of sources: x_coordinate, y_coordinate and region. There are two kinds of bagged tree functions in scikits-learn which are BaggingRegressor and BaggingClassifier. The BaggingRegressor is used for regression problems and gives continuous output. The BaggingClassifier is used for classification problems and gives discrete label output. For regression prediction problems, the output variable is numerical, while the output variable is categorical for classification problems.[3] However, before the end of the training, there is not enough information to judge which regression or classification is better. Therefore, both BaggingRegressor and BaggingClassifier are used, and then the one with better test result is selected.

7.2.1 Classification Label

In the dataset, the region is tags based on location which are discrete categories. Therefore, region is suitable for BaggingClassifier.

For BaggingClassifier, in raw data, the experimental area is divided to 25 regions which are according to 25 categories in the output label, shown as figure5.

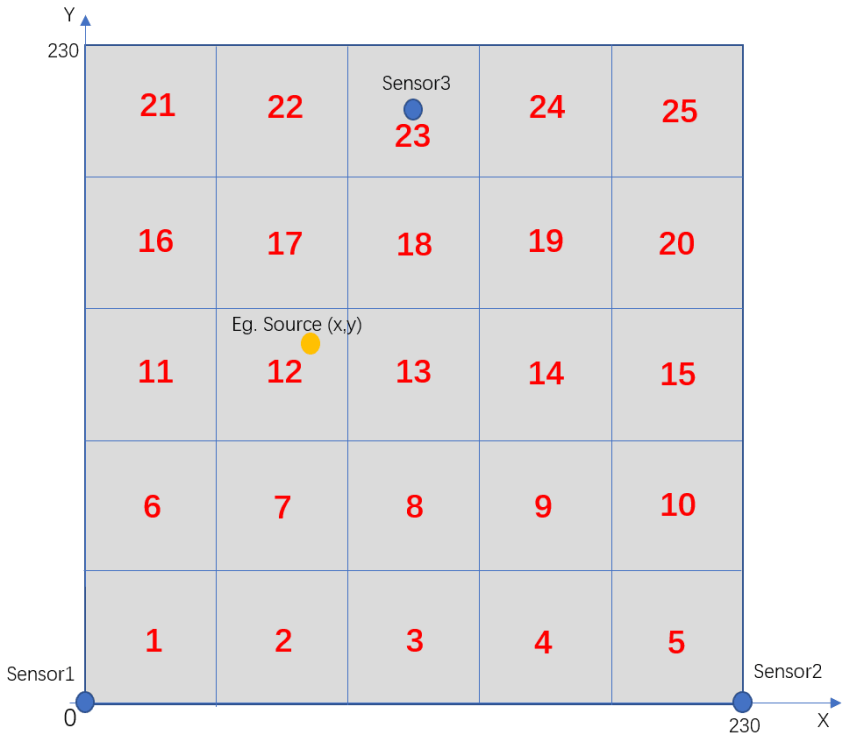

Fig.5.Standered 25 Regions Division

However, since the training results of 25 regions are not ideal, which will be mentioned later in Tuning Parameters and Final Model Results part, the experimental region is re-divided into 4 regions to obtain better results. The redivided regions are shown as figure6 below:

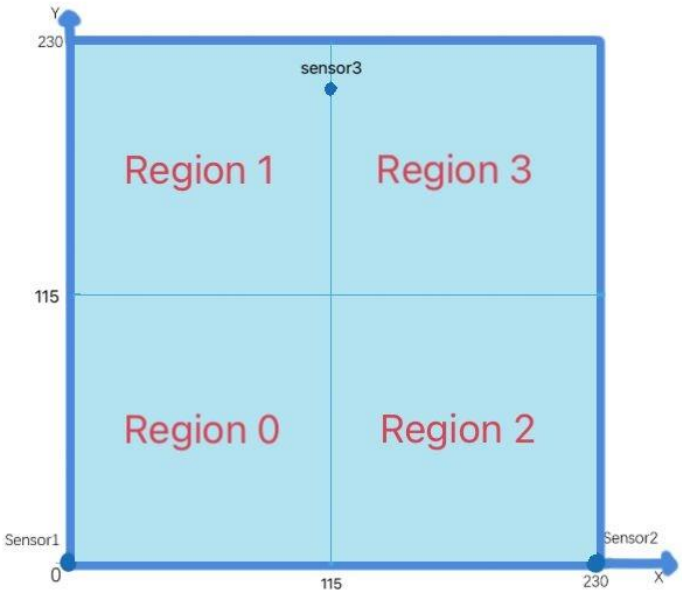

Fig.6.Redivided 4 Regions Division

Therefore, there are two kinds of labels in BaggingClassifier: 4-Region label and 25-Region label.

7.2.2 Regression Label

In the dataset, x and y coordinates are numerical values for location in range (0,230) and can be continuously predicted by non-linear regression function. Therefore, x and y coordinates are suitable for BaggingRegressor.

For BaggingRegressor, x and y coordinates are processed to two kinds. The first type, set [x,y] as the output label to train with one model. The second type, regarding the x and y coordinate as the output label separately: [x], [y], after training with two different models, the predict x and predict y are combined to form a complete coordinate.

7.3 Number of Feature

7.3.1 120 Features

As mentioned in Data Pre-processing, the dataset is generated with 120 features initially. When the number of features is 120, the number of samples is around 200. After 120 features were put into the bagged tree as X, no matter how to adjust the parameters of the model, the models corresponding to the four outputs mentioned above all appeared in the same situation that the training accuracy is much higher than the testing accuracy. This means that severe overfitting has occurred. According to J. Chem, overfitting will appear when the order of magnitude of the training set is smaller than the model complexity, too many unimportant

features exist, the feature distribution of the training set and the test set are inconsistent, etc.[4] Since every 20 minutes of data is processed into a single sample, there are only about 200 training samples here. No matter how to limit the number of features used in the model and the depth of the decision tree, 200 samples are too few for this algorithm. As mentioned in 7.1.1 Bagged Tree, if the number of samples is too small, the performance of bagged tree may be affected.

7.3.2 24 Features

In order to increase the number of samples and reduce the number of features, the characteristics every five minutes are treated as a single sample and the compared characteristics are no longer considered as a feature. In this way, the number of samples has quadrupled, because every 20 minutes of data is divided into four samples. Therefore, a total of about 800 pieces of data were put into the bagged tree model.

The dataset after data preprocessing is a n*30 table which contains both input/features and output/regions & coordinates as illustrated table3 below:

data set no.	sensor1	max	min	ave	median	peak_value	peak_time	prominence	var	sensor2	...	sensor3	...	x_coordinate	y_coordinate	region
1																
2																
3																
4																
5																
…																

Table3. Data Structure for 24 Features

Unfortunately, the testing accuracy for all four-type-label models is still very low and much smaller than the training accuracy. The problem of overfitting remains unsolved. Since the number of features has been greatly reduced, the problem of overfitting seems to have hardly been improved, another cause of overfitting may have happened: too large or wrong hypothesis space. Since the original data is artificially processed into eight characteristics, we assume that these eight characteristics selected by us are the key to get the position from the concentration. In fact, in the process of selecting these eight characteristics as features, perhaps more important characteristics were omitted, and unimportant characteristics were added to the feature as noise.

The detailed 24 feature processing code is shown in appendix2.

7.3.3 Concentration

Therefore, the original concentration data is used for training. Since only the concentration of three sensors are used as features, millions of raw data became training samples. In this way, the problems of too many hypothetical features and too little sample size have been solved. After putting the original concentration data into the bagged tree model, test accuracy has been significantly improved, and the gap between test accuracy and train accuracy has also been reduced to a reasonable range. But the test accuracy is still not high enough, because in the bagged tree model, the timing order of the data is ignored.

The final dataset after data preprocessing is a n*6 table which contains both input/concentrations and output/regions & coordinates as illustrated table4 below:

Dataset no.	concentration_sensor1	concentration_sensor2	concentration_sensor3	x_coordinate	y_coordinate	region
1						
2						
3						
4						
...						

Table4. Data Structure for Concentration

The detailed 3 concentration processing code is shown in appendix3.

7.3.4 Performance with Different Features

The following table5 shows the values of test and train accuracy under different numbers of features and output label choices.

	25 Regions	4 Regions	[X, Y] trained	[X]trained and [Y]trained
120 Features	Test: 23.53	Test:50.98%	Test:35.29%	Test:47.06%
	Train:100%	Train:100.00%	Train:95.73%	Train:95.73%
24 Features	Test:27.72%	Test:57.43%	Test:27.23%	Test:32.18%
	Train:98.94%	Train:99.15%	Train:85.11%	Train:90.00%
3 Concentrations	Test:50.05%	Test:66.71%	Test:49.08%	Test:52.83%
	Train:69.50%	Train:79.80%	Train:64.63%	Train:68.29%

Table5. Performance with Different Features

As shown in the table, the 3 concentration features have the best performance. Therefore, the 3 concentration features are chosen here.

Code of models corresponding to different types of features and labels shown in appendix4.

7.4 Tuning Parameters and Final Model Results

According to the table5 above, from a longitudinal comparison, grouped by the four different labels, the concentration features always have the highest test accuracy and the smallest test and train accuracy difference. This means that in the case of 3 concentration, the test performs best, and the problem of overfitting is also solved best. In horizontal comparison, 4 regions have t e best performance, followed by X trained and Y trained, 25 Regions, and the worst is [X, Y] trained. However, different labels have different precision for regional division. When using the X and Y coordinates as labels, the test accuracy error tolerance rate is set to 40 to obtain the best results. Threshold = 40 means that when |predict_y-y| <= 40, the prediction is judged to be accurate. This is equivalent to a sliding window with a size of 80. The precision of this judgment is much less than that of dividing the area into 25 distinct regions. In addition, although the precision of the 4 region is not as high as that of the 25 regions, the test accuracy of the 4 region is much higher than that of the 25 regions. Therefore, both the 4 region and 25 region models of the concentration feature have been decided to remain， shown as the dark yellow part in table5 above.

After deciding on the types of features and label, the parameters of the model itself also need to be debugged to get the best performance. For BaggingClassifier, there are 11 types of parameters that need to be decided which are shown as below[5]:

- base_estimator

The base_estimator is used to identify the base estimator to fit on random subsets of the dataset. Since decision tree is used here, base_estimator = None.

- max_samples

The max_samples is the max number of samples drawn from dataset to each base estimator. When the limit is large, the accuracy is high. Therefore, set the number equal to length of dataset. max_samples = 1.0.

- max_features

The max_ features is the max number of features drawn from dataset to each base estimator. Since there are 3 features in total for concentration dataset, max_features = 3.

- bootstrap

The bootstrap determines whether samples are drawn with replacement. Here default value is chosen, bootstrap = True.

- bootstrap_features

The bootstrap_features determines whether features are drawn with replacement. Here default value is chosen, bootstrap_features = false.

- oob_score

The oob_score is related to built-in generation error calculation, which is not used here.

- warm_start

The warm_start is to decide whether trained based on model before. Since only one time trained here, warm_start = False.

- n_jobs

The n_jobs means number of jobs to run in parallel. Here default None is chosen.

- random_state

The random state controls random resampling of the original dataset. Since any int number is just a seed which will not affect performance, a seed 0 is chosen here.

- Verbose

The verbose controls the verbosity, the default value 0 is chosen.

- n_estimators

The n_estimator is the number of estimators to generate average model, which is quite important for final performance.

During tuning, num_estimators sampled at (1, 150) at intervals of 5, and the following plot was generated together with train and test accuracy. For 4 region model, the plot is shown as below in figure7:

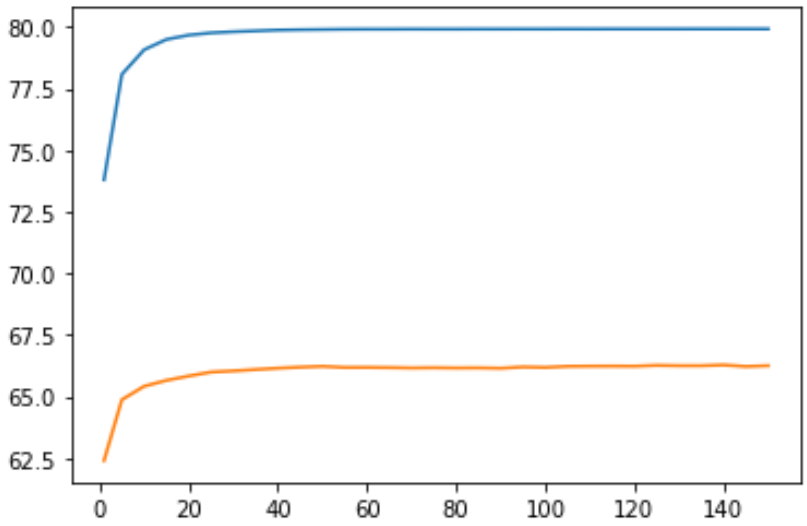

Fig.7.Accuracy of 4-region vs Num_estimators

For 25 region model, the plot is shown as below in figure8:

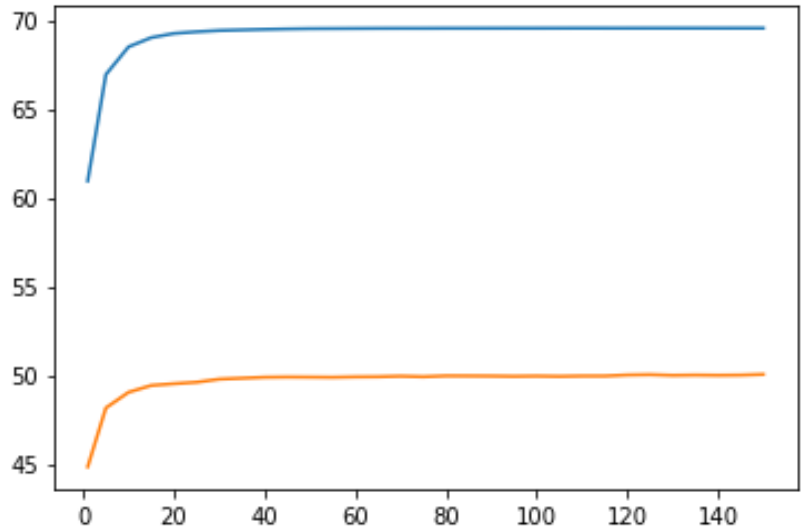

Fig.8.Accuracy of 4-region vs Num_estimators

As can be seen from the figure, after the num_estimator is greater than 20 in region4 and greater than 30 in region25, both the train and test accuracy tend to be flat and no longer change significantly. Due to Michael J. Kearns, In the case where the performance improvement is not obvious, a model with lower complexity and consumption should be chosen.[6] Since the model body is small and time-consuming, we tend to choose a more stable model. Therefore, num_estimator = 40 for region4 and num_estimator = 50 for region25 are chosen.

In general, two BaggingClassifier are decided with parameters(base_estimator=None, n_estimators=N, max_samples=1.0, max_features=3, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=0, verbose=0). N = 40 in 4region model and N = 50 in 25region model. The final accuracy is shown in the dark yellow part in the table5 above.

The code of final trained model is shown in appendix5 and the trained-well model for 4-region and 25-region is shown in appendix6.

7.5 Output Model

After getting the trained model, the model should be used in real time prediction. A prediction function is generated. The input data should be processed to the same structure as input in training: [sensor1 concentration, sensor2 concentration, sensor3 concentration]. And regional information should also be inputted into this function. When region = 4, the result will follow the 4 region division figure above, and when region = 25, it will follow the Standardized environment division. The real-time model is shown as below:

```
def predict(region,input_data):
    if region == 25:
        out_region = model_25.predict(input_data)
    if region == 4:
        out_region = model_4.predict(input_data)
    return out_region
```

Code of output model is shown in appendix7.

7.6 Other Algorithms

7.6.1 RNN

Since bagged trees cannot consider the timing order of the data during training, the final test accuracy is not very ideal. Recurrent Neural Network (RNN) is a type of neural network dedicated to processing time series data samples. Each layer of it not only outputs to the next layer, but also outputs a hidden state for the current layer to use when processing the next

sample. RNN can capture the relation between each set of data, theoretically it can improve the accuracy. The mechanism of RNN is shown as figure9 below:

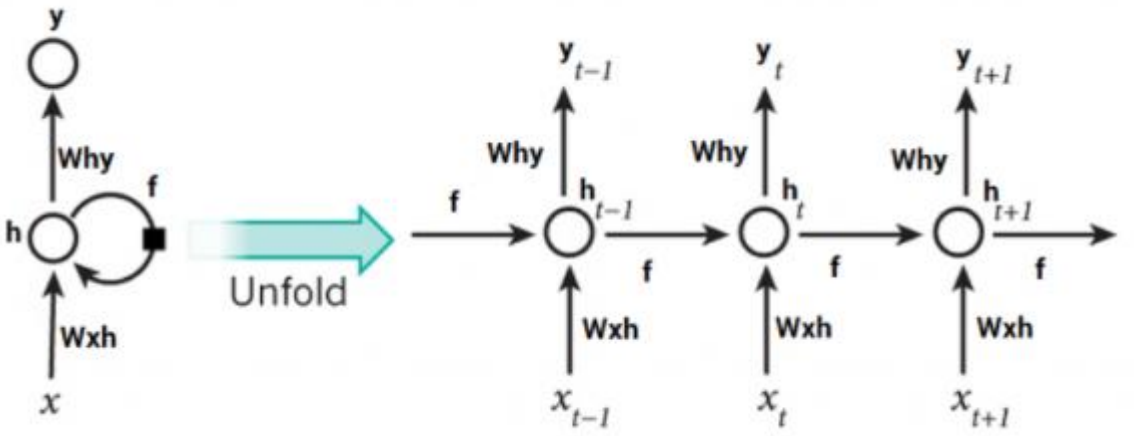

Fig.9. Mechanism of RNN [7]

The neutral network receives x_t at time t to get hidden layer value h_t and output y_t. The key point is that the value of ht not only depends on xt, but also depends on ht-1. Therefore, RNN can handle timing series dataset well.

The basic RNN classification algorithm is tried, but because our data needs to calculate the association between hundreds of data, this will cause the complexity to be too great for our computer's computing ability. If the time step is too small, the result is quite bad, while the time step is too large, so that run time error occurs.

In theory, the RNN algorithm that can include timing features can improve the performance of the results. However, for this project, this requires better hardware or better optimized algorithms.

In the future, if there is better hardware support or more optimized algorithms, the RNN algorithm should be a method worth trying.

8 Conclusion

For the first semester, Zheng Gerui, Li Danni and I worked together to study background information and Theoretical knowledge of this project. We have developed a clear project timeline and divided the work for everyone. However, due to the emergence of the COVID-19, our plan has been disrupted. The Nordic Thingy-52 ordered from the United States did not arrive on time, which delayed our plan. So, in addition to theoretical knowledge, I assisted in setting up the Raspberry Pi in the first semester.

Due to the COVID-19, Danni and I returned to China during the summer vacation. Gerui was mainly responsible for the experimental data collection. After conducting research on feature engineering in the summer vacation, I was mainly responsible for data pre-processing and algorithms in the second semester. Due to the incomplete theoretical knowledge, we made too many assumptions in the early stage of feature engineering, so that serious overfitting appeared later. After selecting the appropriate label and feature, the performance of the bagged tree is still not good enough. In fact, a lot of time was spent on the implementation of the RNN algorithm, but due to time and hardware limitations, the RNN algorithm did not get better results in the end. I am very interested in the research of algorithms. I am willing to make further improvements to this project if the hardware conditions permit or there are better algorithms in the future.

In conclusion, through the final year project, I learned a lot about machine learning algorithms and the Internet of Things. I want to express my appreciation to Associate Professor Ang Kah Wee, Dr Tan Wee Chong and my teammates for their help.

9 Bibliography

[1] A. Famili et al, "Data Preprocessing and Intelligent Data Analysis," Intelligent Data Analysis, vol. 1, (1), pp. 3-23, 1997.

[2] D. Dey, “ML: Bagging classifier,” GeeksforGeeks, 20-May-2019. [Online]. Available: https://www.geeksforgeeks.org/ml-bagging-classifier/.

[3] D. M. J. Garbade, “Regression versus classification machine learning: What's the difference?” Medium, 11-Aug-2018. [Online]. Available: https://medium.com/quick-code/regression-versus-classification-machine-learning-whats-the-difference-345c56dd15f7.

[4] J. Chem. Inf. Comput. Sci.” The Problem of Overfitting” 2-Dec-2003. [Online]. Available: https://pubs.acs.org/doi/full/10.1021/ci0342472

[5] “Sklearn.ensemble.BaggingClassifier,” scikit. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

[6] M. J. Kearns, The computational complexity of Machine Learning. Cambridge, MA: MIT Press, 1990.

[7] P. Srivastava, Mechanism of RNN. 2017. [Online]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.analyticsvidhya.com%2Fblog%2F2017%2F12%2Ffundamentals-of-deep-learning-introduction-to-lstm%2F&psig=AOvVaw1EFs95Lw1KPnGELmpN1FUQ&ust=1635334135961000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCIiVxND85_MCFQAAAAAdAAAAABAD

10 Appendix

1. Compute_features_120.py.

2. Compute_features_24.ipynb

3. concentration_feature.py

4. train_4region_24feature.ipynb
   train_4region_120feature.ipynb
   train_4region_concentration.ipynb
   train_25region_120feature.ipynb
   train_25region_24feature.ipynb
   train_25region_concentration.ipynb

5. Final_Trained.ipynb

6. Bagged_region_4.model
   Bagged_region_25.model

7. model.ipynb

Since only PDF document can be uploaded, for appendix document above, please contact

e0261881@u.nus.edu.