EEES 406 – DATA ANALYTICS FOR IoT

FINAL REPORT
Project Title: MONITOR AND REMEDY AIR QUALITY WITH A RASPBERRY PI

Project Members:

Member #1

Name & Surname: İsmet ÖZKAN

Student ID: 2012513051

Member #2

Name & Surname: Kamil DİNLEYİCİ

Student ID: 2016513019

Abstract:

The aim of this project is to do the task of 2 sensors with a single sensor. So, we tried to estimate the value of MQ-135 according to the data from DHT-11 by comparing the data coming simultaneously from DHT-11 and MQ-135 over a long period of time. When this predicted value exceeds the pre-determined threshold value of air pollution, the ventilation system is activated. It operates until the air pollution value falls below the threshold value.

Problem Definition:

At first, our aim was just to improve the air quality in the living space, but we aimed to reduce the number of equipment to be used and to do more work with a single sensor. As a result, we created a system where the collected data and sensors can imitate each other by training a single model.

System security requirements may vary depending on the areas where the model is used. However, once the system is trained, if the internet access is cut and it starts to operate as a closed circuit, it is not possible to hack without physical access.

Weakness of the project; when comparing data from sensors, there may be incompatible values. For this reason, errors can sometimes occur in predicted values.

The strong part of the project; is about doing a lot of work with less equipment and by changing the sensors that are used in the model. If it is desired to make predictions between other sensors it provides easy estimation for different sensors.

System Design and Features:

EQUIPMENT LIST

Equipment are listed below.

-Raspberry Pi 4 2 GB / Raspberry Pi 3 B+ (We bought different Pi's)

-MQ-135 Air Quality Sensor

-DHT-11 Temperature and Humidity Sensor

-ADS1115 16 Bit I2C 4 Channel Analog to Digital Converter

-Fan

BLOCK DIAGRAMS OF SYSTEM

Figure 1 represents the scheme of the system we set up to collect data. By using the data we collect here, we reduce the system requirements as shown in Figure 2. After training our system, we no longer need MQ-135 sensor and ADS1115(ADC).

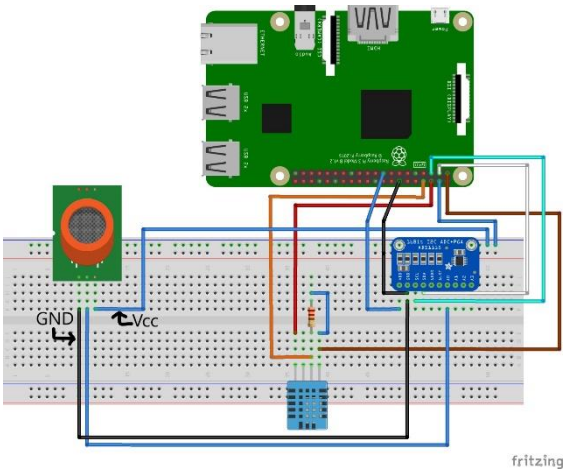

Figure 1.

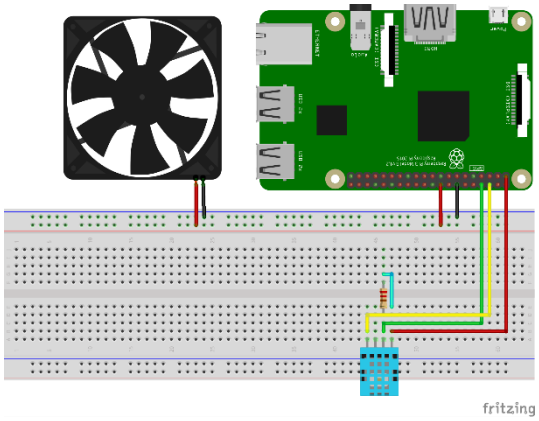

Figure 2.

COMMUNICATION AND NETWORK PROTOCOLS

We used the MQTT protocol for storage on a server by sending our data over ethernet and the I2C communication protocol is used when we have needed to utilize the functions of analog to digital converter(ADS1115).

DATA

Data we collected are “humidity ratio” and “temperature” from DHT-11,“PPM” from MQ-135.

	Temperature	Humidity	Air Quality
0	29.6	63.0	321.1
1	29.6	63.0	320.0
2	29.6	63.0	319.2
3	29.5	62.0	318.6
4	29.6	62.0	317.5
...	...	...	...
12812	30.4	28.0	613.3
12813	30.5	28.0	613.7
12814	30.5	28.0	614.0
12815	30.5	28.0	612.5
12816	30.6	27.0	611.4

12817 rows × 3 columns

Figure 3. The dataset

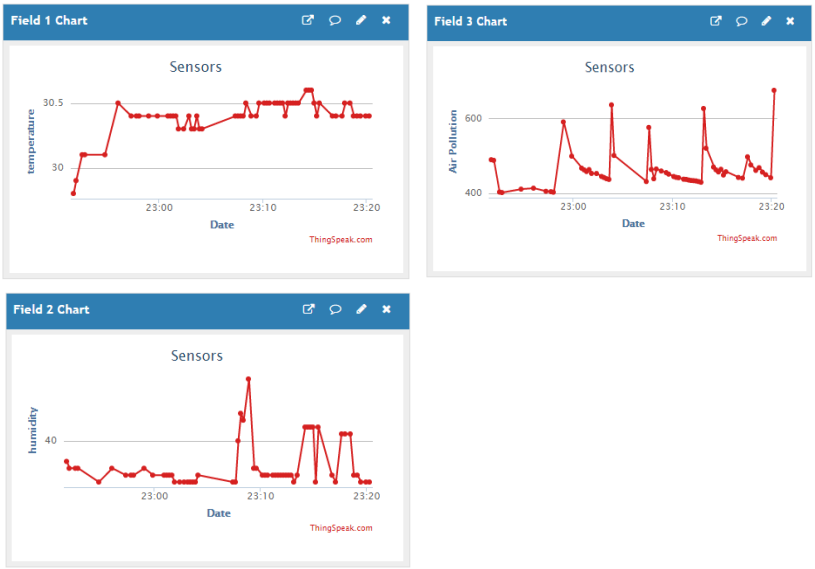

Figure 4. Demonstration of data on cloud server

DATA PROCESS & MACHINE LEARNING

We thought that the Multiple Linear Regression algorithm will fit dataset. After we prepared the data we plotted the graph between independent variables and dependent variables. It was obvious that there was no linear connection between features and target value.

"matplotlib.pyplot" library was added to observe these graphs.

```
ax1 = df.plot.scatter(x="Temperature", y ="Air Quality",c='DarkBlue')
```

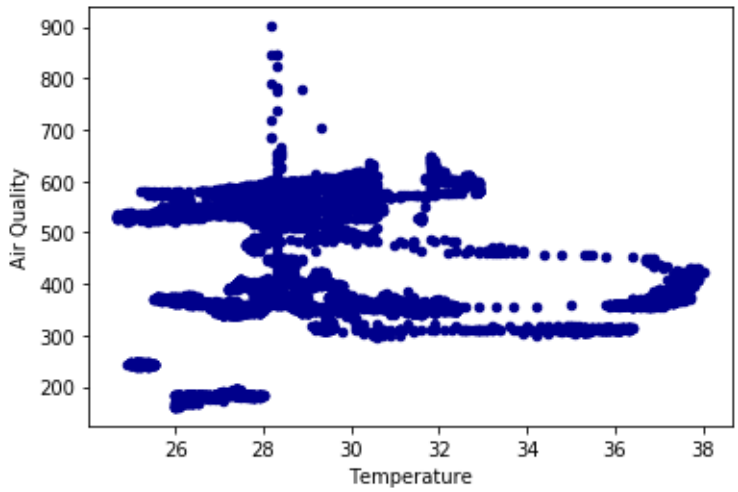

Figure 5. Relationship between "Temperature" and "Air Quality"

```
ax1 = df.plot.scatter(x="Humidity", y ="Air Quality",c='DarkBlue')
```

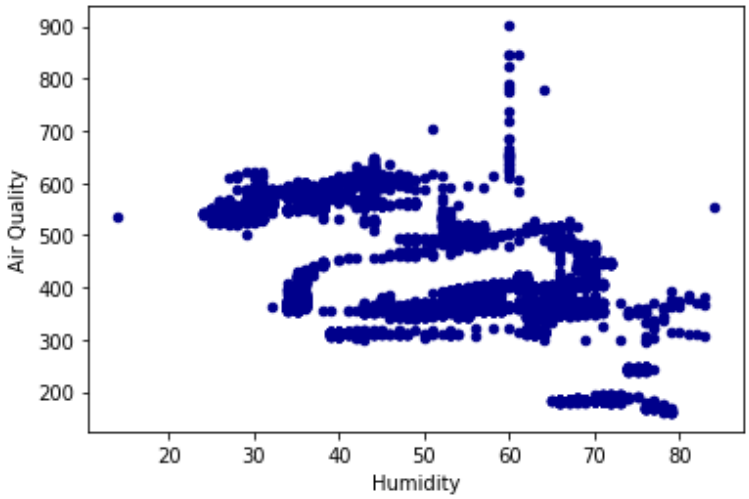

Figure 6. Relationship between "Humidity" and "Air Quality"

Although we could not find any linear relationship, we wanted to see the results. The whole data splitted as test and train data.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,random_state =42)
```

Scikit-Learn library was added to build the Linear Regression model and model fitted.

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
model = lm.fit(x_train,y_train)
```

The intercept and coefficient value for the equation is calculated.

```
model.intercept_
```

```
1071.8982355097528
```

```
model.coef_
```

```
array([-12.57406131,  -5.59241707])
```

Using test dataset the predicted target value is calculated.

```
y_pred = model.predict(x_test)
```

After calculating the predicted value we compared this value with test dataset.

Actual Value	Predicted value	Difference
571.1	474.320960	96.779040
602.2	449.172838	153.027162
313.1	405.594595	-92.494595
549.9	519.192118	30.707882
316.9	352.976163	-36.076163
525.6	600.959025	-75.359025
549.1	567.009059	-17.909059
352.3	353.371626	-1.071626
372.3	360.353270	11.946730
370.4	340.102951	30.297049
244.9	334.343204	-89.443204
397.1	416.576591	-19.476591
398.9	356.748381	42.151619

The difference was oscillating and eventually, we calculated R^2 score and RMSE(Root Mean Square Error). These two metrics indicate how good the model is.

R^2 score is defined at Wikipedia as follows: *" ...the proportion of the variance in the dependent variable that is predictable from the independent variable(s)."* It changes between 0 and 1. A low value would show a low level of correlation and how much it is closed to 1 it means the independent and dependent variables are more correlated.

RMSE(Root Mean Square Error) is the root of the average of the square of the errors.

```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

```
0.5334742690620904
```

```
rmse = np.sqrt(sklearn.metrics.mean_squared_error(y_test,y_pred))
print(rmse)
```

```
73.11586576514566
```

The R^2 score was low and RMSE was high for the Linear Regression model. Considering these values, we switched the model to the other models to observe higher R^2 score and lower RMSE score.

We were building the model on Jupyter Notebook(Windows PC) then were transferring the model through GitHub to Raspberry Pi. Due to the discrepancy of model building structures(Windows PC-64 bit and Raspberry Pi-32 bit), we could not transfer the rest of the models except the Linear Regression model. The models which are not compatible with Raspberry Pi are built and they are used in Raspberry Pi.

We started calculating K-Neighbors Regressor. The cross-validation method was used to get an exact number of neighbors and founded 7. After we finished calculations on KNN we tried other models. We observed relatively higher and lower R^2 score and RMSE values and decided to use **Random Forest Regression**. It has the highest R^2 score and lowest RMSE score among the other models.

The comparison is listed in Figure 7.

MODEL	R^2 score	RMSE (Root Mean Square Error)
Support Vector Regression(SVR)	0.5843	69.0136
Multi-layer Perceptron Regression	0.6248	65.5663
K-Neighbors Regression(KNN)	0.9309	28.1322
Decision Tree Regression	0.9368	26.8948
Random Forest Regression	0.9391	26.4274

Figure 7. The comparison of metrics between models

THE PROGRAM FOR RANDOM FOREST REGRESSION MODEL

First of all, the needed libraries are added. Then, the collected dataset is imported to the program. The redundant features are extracted from the dataset. The train and test dataset is created with a ratio of 20% for test and 80% for training.

Using RandomForestRegressor from the Scikit-Learn library we created a "random_forest_model" object and fitted the model with training data. Prediction is made using predict function of sklearn. R2 and RMSE score is printed and the model is exported to use in our main program. All code is shared below section.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import model_selection
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor

the_data = pd.read_csv('iot_project_setdata.csv')

del the_data['latitude']
del the_data['longitude']
del the_data['elevation']
del the_data['status']
del the_data['field4']
del the_data['entry_id']
del the_data['created_at']

df = the_data.rename(columns =
{"field1":"Temperature","field2":"Humidity","field3":"Air Quality"})

x = df.drop("Air Quality", axis =1)
y = df["Air Quality"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.2,random_state =42)

random_forest_model = RandomForestRegressor()

random_forest_model.fit(x_train, y_train)

y_pred = random_forest_model.predict(x_test)

y_test = np.array(y_test)

#R2 Score
r2_score = r2_score(y_test, y_pred)
print("R2 score for Random Forest model:" , r2_score)

#RMSE
RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE for Random Forest model:" , RMSE)

#DUMP THE MODEL
import joblib
filename = 'rf_model.sav'
joblib.dump(random_forest_model, filename)
```

MAIN PROGRAM

```
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import dht11
import time
import datetime
import sklearn
from sklearn import *
import board
import busio
import joblib
import pandas as pd

#Load model
model = joblib.load("/home/pi/Desktop/knn_files/rf_model.sav")

#Threshold value for air quality ratio
ppmThreshold = 600

# initialize GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

# read data using pin 14
instance = dht11.DHT11(pin=14)

client = mqtt.Client()
client.connect("mqtt.thingspeak.com", 1883, 60)

channelId = "1334074" #kendi channel ID'ni gir
apiKey =  "WZO06O50VBD33BWN" #kendi write api key'ini gir

try:
    while True:
        result = instance.read()
        if result.is_valid():
            temp = result.temperature
            humidity = result.humidity
            new_dataset = [[result.temperature],[result.humidity]]
            new_dataset = pd.DataFrame(new_dataset).T
            estimated_apr =
("{:.2f}".format(float(model.predict(new_dataset))))
            publish_path = "channels/" + channelId + "/publish/" + apiKey
            publish_data = "field1=" + str(temp) + "&field2=" +
str(humidity) +"&field3=" + str(estimated_apr)
            client.publish(publish_path,publish_data)
            print("Last valid input: " + str(datetime.datetime.now()))
            print("Temperature: %-3.1f C" % result.temperature)
            print("Humidity: %-3.1f %%" % result.humidity)
            print("Estimated APR is: ",  estimated_apr)
        time.sleep(15)
        if estimated_apr > ppmThreshold:
              GPIO.output(24, 1)
              time.sleep(15)
        else:
              GPIO.output(24, 0)
```

```
            time.sleep(15)

except KeyboardInterrupt:
    print("Cleanup")
    GPIO.cleanup()
```

OUTPUT FOR EACH 15 SECONDS (it changes with corresponding values):

```
Last valid input: 2021-05-19 17:00:46.705755
Temperature: 27.9 C
Humidity: 25.0 %
Estimated APR is:  547.06
```

Important libraries are added at first then the model that was created at above program is imported using Joblib.

Now, we are able to see the temperature, humidity and the Air Pollution Ratio value on the ThingSpeak.com. The models are created by the model. Prediction is made by using temperature and humidity values.

We searched and found that 600 PPM value is risky for environment and assigned this value as “ppmThreshold”. If the estimated PPM value which is created by the our Machine Learning model exceed the assigned threshold value the fan will be “ON” state and try to decrease these value under threshold value. Unless the estimated PPM value exceed threshold value the fan will be “OFF” state.

RESULTS AND CONCLUSION

We completed all the goals that we presented at the beginning. Actually, we believed that we made more job more than we presented. We thought that Linear Regression will be sufficient to process data but it could not help us and we searched and developed new models. It was time-consuming but we learned lots of things and worths it.

There are models that predict what the current data might be when systems or sensors malfunction, but we have not seen a project that correlates sensor values with each other as in our project.

If we increase the accuracy of the predictions by using better quality sensors and if we can prove that the system works with more different sensors, the possibility of selling the model will increase. Besides how accurately the model works, people care about visuals and interfaces. We think that a panel where the working status of the system can be monitored and intervened will increase the commercial success rate. In addition, maybe if a more cost-effective computer that will work only for this task is designed instead of a multi-functional single board computer such as Raspberry Pi, it can find a place in the market.