A CAL Project Report

on

Weather Forecast Using LSTM And Zambretti Algorithms

to be submitted in partial fulfilling of the requirements for the course on:

CLOUD COMPUTING

Submitted By:

17BIT0156 Shubham Totla

17BIT0185 Krishna Agrawal

17BIT0216 Arun Giri

17BIT0361 Prashanta Dhar

OBJECTIVE:

We intend to use the IoT device to collect the data pertaining to the weather like temperature, humidity, pressure and then use this data to run a machine learning algorithm that will help us to predict the future values of respectively, temperature, humidity, pressure which we can then use to predict the type of weather that the area could experience.

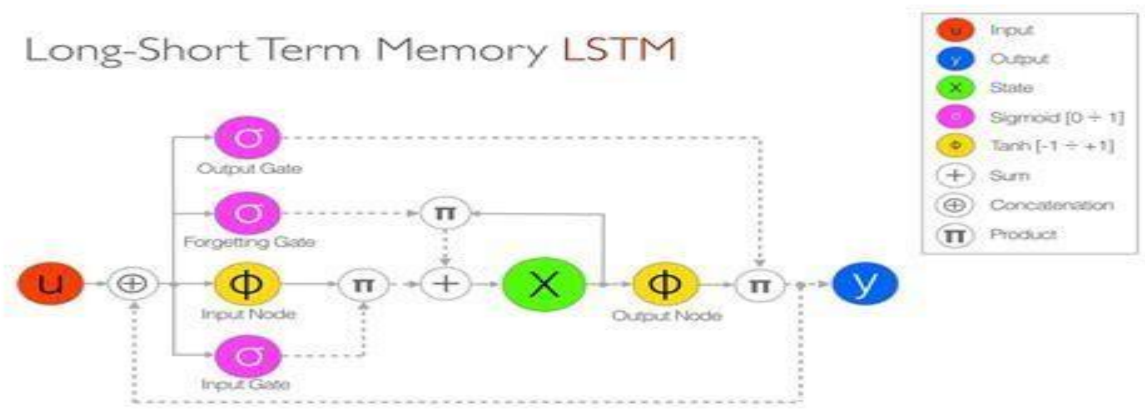
METHODOLOGY:

We used the sensors namely, BMP-180 which is a pressure sensor to get the pressure readings and DHT-22 which a temperature and humidity sensor to get the required data. We connected the IoT devices through appropriate wiring and hard coded the Arduino to collect the data after every 10 minutes. The Node MCU was connected to the Things board cloud service platform, where the data was getting stored and visualized.

Then we used the collected data to train the algorithm for weather prediction. We used LSTM algorithm which is a type of RNN with more memory capabilities such that the previous data can be used to predict the outcome of the future occurrences. We applied this algorithm individually for the temperature, humidity, pressure values. From the predicted temperature and pressure, we also calculated the sea level pressure based on the altitude of the given surface, as in this case Vellore and its altitude 213. We used ZAMBRETTI algorithm to predict the weather based on the predicted values of temperature, pressure and humidity.

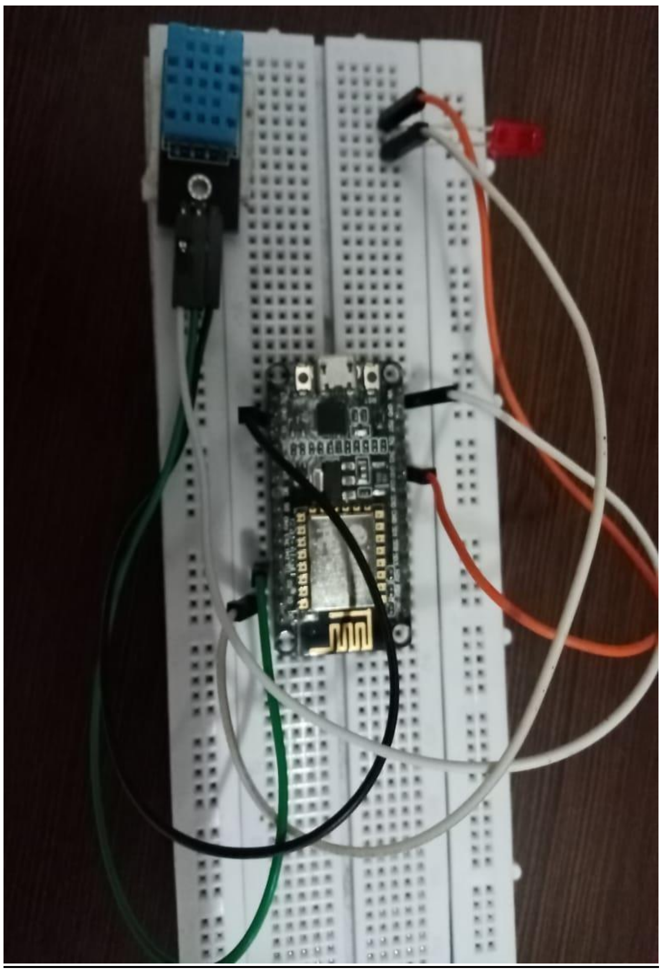
ABOUT OUR ALGORITHM:

The Long Short Term Memory (LSTM) is a second-order recurrent neural net- work architecture that excels at storing sequential short term memories and retrieving them many time-steps later. To this point, the usefulness of the LSTM training algorithm has been limited by the fact that it can only be applied to a small set of network architectures. Here we introduce the Generalized Long Short-Term Memory (LSTM-g), which provides a local, efficient, LSTM-like training algorithm that can be applied without modification to a much wider range of second-order network architectures. With LSTM-g, all units have an identical set of operating instructions for both activation and learning, sub- ject only to the configuration of their local environment in the network; this is in contrast to LSTM, where each type of unit has its own activation and training instructions. When applied to LSTM-compatible architectures with peephole connections, LSTM-g takes advantage of an additional source of back-propagated error which can enable better performance than traditional LSTM. Enabled by the broad architectural applicability of LSTM-g, we demonstrate that training recurrent networks engineered for specific tasks can produce better results. We conclude that LSTM-g has the potential to both improve the performance and broaden the applicability of the LSTM family of gradient-based training algorithms for recurrent neural networks.



IOT IMPLEMENTATION:

Sensors used: DHT22,BMP189



above figure demonstrates our IOT circuit.

CODE:

TEMPERATURE:
#RNN

```
#Data Pre-Processing

#Libraries  import  numpy  as
np import pandas as pd
import matplotlib.pyplot as plt

#Importing training set
dataset              =              pd.read_csv('C:/Users/SHUBHAM
TOTLA/Desktop/WEATHER1.csv')                train                =
dataset.iloc[:,3:4].values

#Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0,1))#All the prizes would be between 0 and 1
train_scaled = sc.fit_transform(train)

#Creating a data structre with 60 timesteps and 1 output x_train = [] y_train = []
for i in range(24,324):    x_train.append(train_scaled[i-24:i,0])
y_train.append(train_scaled[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)

#Reshaping
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

#BUILDING THE RNN

#Importing Keras
from keras.models import Sequential from
keras.layers import Dense from keras.layers import
LSTM from keras.layers import Dropout

#Initializing RNN
regressor = Sequential()

#Adding first layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True,
input_shape =  (x_train.shape[1], 1))) regressor.add(Dropout(0.2))

#Adding second layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True)) regressor.add(Dropout(0.2))

#Adding third layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True)) regressor.add(Dropout(0.2))
```

```
#Adding fourth layer and Dropout regularization regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

#Adding Output Layer regressor.add(Dense(units = 1))

#Compiling RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(x_train, y_train, epochs = 100, batch_size= 32  )

#MAKING PREDICTIONS

#Getting
data_test = pd.read_csv('C:/Users/SHUBHAM TOTLA/Desktop/weather2.csv')
real_temperature = data_test.iloc[:,3:4].values

#Getting Temperature
dataset_total = pd.concat((dataset['TEMPERATURE'], data_test['TEMPERATURE']), axis =
0) inputs = dataset_total[len(dataset_total) - len(data_test) - 24:
].values
inputs = inputs.reshape(-1 ,1)# It will put everything in one clolumn since we didn't use iloc inputs
= sc.transform(inputs)
x_test = [] for i in range(24, 52):
x_test.append(inputs[i-24:i,0]) x_test =
np.array(x_test)
x_test  =  np.reshape(x_test,  (x_test.shape[0],  x_test.shape[1],  1))  predicted_temperature  =
regressor.predict(x_test) predicted_temperature =
sc.inverse_transform(predicted_temperature)
```

# Visualising the results

```
plt.plot(real_temperature, color = 'red', label = 'Real Temperature') plt.plot(predicted_temperature,
color = 'blue', label = 'Predicted Temperature')
plt.title('Temperature Prediction')
plt.xlabel('Time') plt.ylabel('Temperature')
plt.legend() plt.show()
```

<u>HUMIDITY:</u>

```
#RNN

#Data Pre-Processing

#Libraries  import  numpy  as
np import pandas as pd
import matplotlib.pyplot as plt

#Importing training set
```

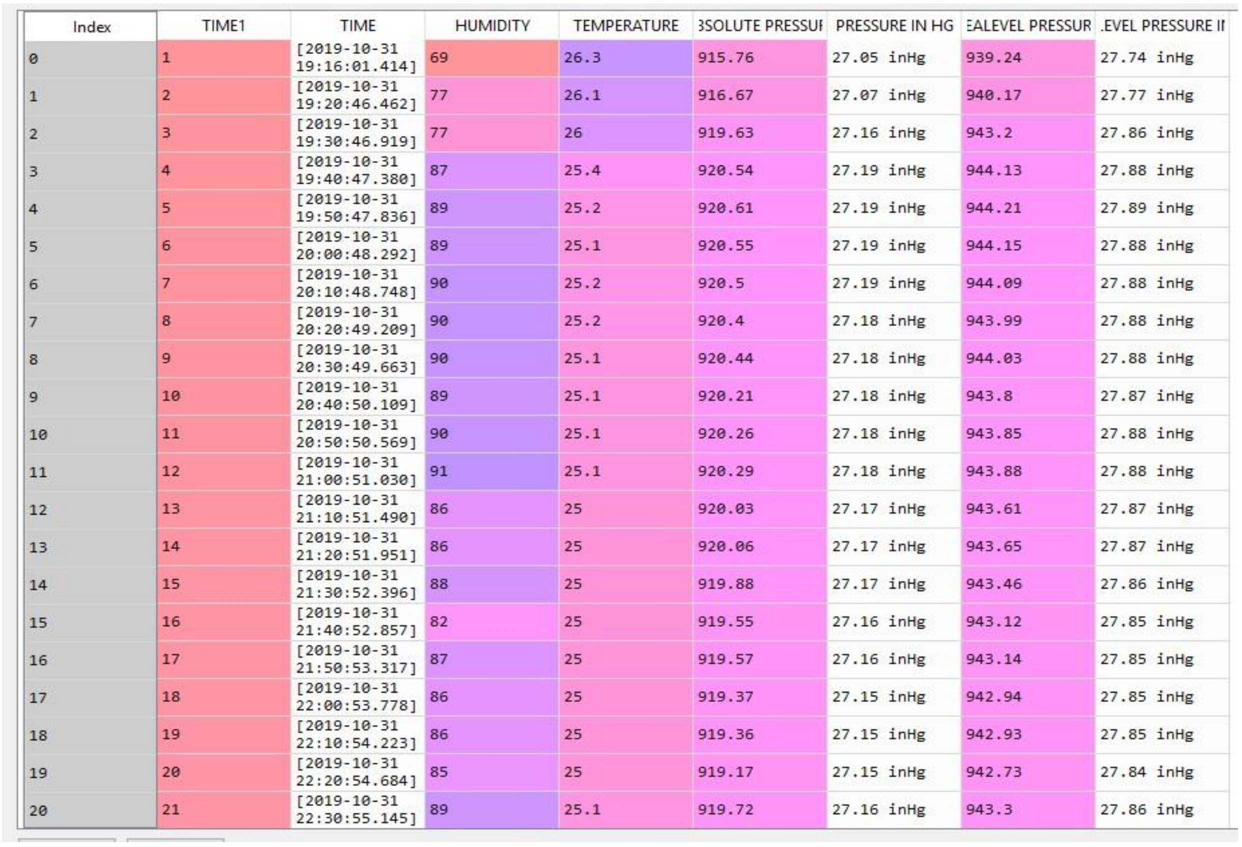
```
dataset = pd.read_csv('C:/Users/SHUBHAM
TOTLA/Desktop/WEATHER1.csv') train =
dataset.iloc[:,2:3].values

#Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0,1))#All the prizes would be between 0 and 1
train_scaled = sc.fit_transform(train)

#Creating a data structre with 60 timesteps and 1 output x_train = [] y_train = []
for i in range(24,324):     x_train.append(train_scaled[i-24:i,0])
y_train.append(train_scaled[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)

#Reshaping
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

#BUILDING THE RNN

#Importing Keras from keras.models import
Sequential from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

#Initializing RNN
regressor = Sequential()

#Adding first layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True,
input_shape =  (x_train.shape[1], 1)))
regressor.add(Dropout(0.2))

#Adding second layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True)) regressor.add(Dropout(0.2))

#Adding third layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True)) regressor.add(Dropout(0.2))

#Adding fourth layer and Dropout regularization regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

#Adding Output Layer regressor.add(Dense(units = 1))

#Compiling RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(x_train, y_train, epochs = 100, batch_size= 32  )
```

```
#MAKING PREDICTIONS

data_test = pd.read_csv('C:/Users/SHUBHAM TOTLA/Desktop/weather2.csv')
real_humidity = data_test.iloc[:,2:3].values

#Getting Humidity
dataset_total = pd.concat((dataset['HUMIDITY'], data_test['HUMIDITY']), axis = 0) inputs =
dataset_total[len(dataset_total) - len(data_test) - 24:
].values
inputs = inputs.reshape(-1 ,1)# It will put everything in one clolumn since we didn't use iloc inputs
= sc.transform(inputs)
x_test = [] for i in range(24, 52):
x_test.append(inputs[i-24:i,0]) x_test =
np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1)) predicted_humidity =
regressor.predict(x_test)
predicted_humidity = sc.inverse_transform(predicted_humidity)



# Visualising the results plt.plot(real_humidity, color = 'red', label = 'Real Humidity')
plt.plot(predicted_humidity, color = 'blue', label = 'Predicted Humidity')
plt.title('Humidity Prediction')
plt.xlabel('Time') plt.ylabel('Humidity')
plt.legend() plt.show()
```
PRESSURE:

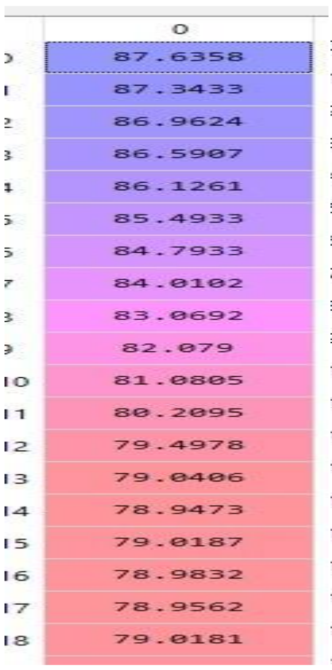
```
#RNN

#Data Pre-Processing

#Libraries   import   numpy   as
np import pandas as pd
import matplotlib.pyplot as plt

#Importing training set
dataset                =                pd.read_csv('C:/Users/SHUBHAM
TOTLA/Desktop/WEATHER1.csv')                  train                  =
dataset.iloc[:,4:5].values

#Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0,1))#All the prizes would be between 0 and 1
train_scaled = sc.fit_transform(train)

#Creating a data structre with 60 timesteps and 1 output x_train = [] y_train = []
for i in range(24,324):
   x_train.append(train_scaled[i-24:i,0])     y_train.append(train_scaled[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)

#Reshaping
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

#BUILDING THE RNN

#Importing Keras from keras.models import
Sequential from keras.layers import Dense from
keras.layers import LSTM
from keras.layers import Dropout

#Initializing RNN regressor =
Sequential()

#Adding first layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True,
input_shape =  (x_train.shape[1], 1)))
regressor.add(Dropout(0.2))

#Adding second layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True)) regressor.add(Dropout(0.2))

#Adding third layer and Dropout regularization regressor.add(LSTM(units = 50, return_sequences =
True)) regressor.add(Dropout(0.2))

#Adding fourth layer and Dropout regularization regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

#Adding Output Layer
regressor.add(Dense(units = 1))

#Compiling RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')#Stochastic Gradient descent

# Fitting the RNN to the Training set
regressor.fit(x_train, y_train, epochs = 100, batch_size= 32  )

#MAKING PREDICTIONS

data_test = pd.read_csv('C:/Users/SHUBHAM TOTLA/Desktop/weather2.csv')
real_pressure = data_test.iloc[:,4:5].values

#Getting Pressure
dataset_total = pd.concat((dataset['ABSOLUTE PRESSURE'], data_test['ATM PRESSURE']),
axis = 0) inputs = dataset_total[len(dataset_total) - len(data_test) - 24:
].values
```

```
inputs = inputs.reshape(-1 ,1)# It will put everything in one clolumn since we didn't use iloc inputs
= sc.transform(inputs)
x_test = [] for i in range(24, 52):
x_test.append(inputs[i-24:i,0]) x_test =
np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1)) predicted_pressure =
regressor.predict(x_test)
predicted_pressure = sc.inverse_transform(predicted_pressure)

# Visualising the results
plt.plot(real_pressure, color = 'red', label = 'Real Pressure') plt.plot(predicted_pressure, color = 'blue',
label = 'Predicted Pressure')
plt.title('Pressure Prediction')
plt.xlabel('Time') plt.ylabel('Pressure')
plt.legend() plt.show()
```

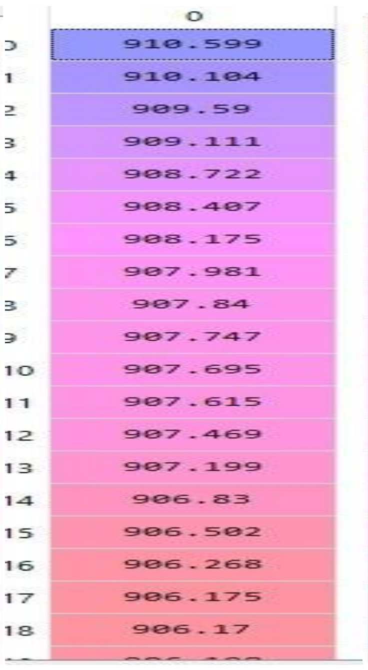
RESULT: Showcasing Our results sets.



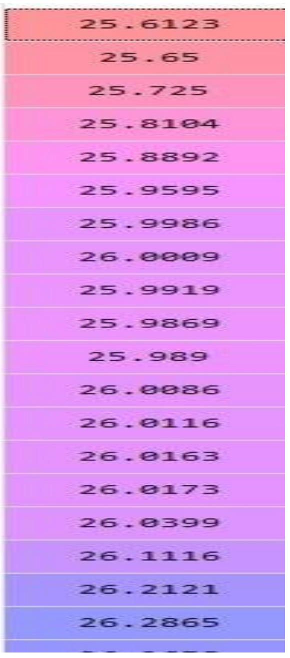
Index	TIME1	TIME	HUMIDITY	TEMPERATURE	3SOLUTE PRESSUI	PRESSURE IN HG	EALEVEL PRESSUR	.EVEL PRESSURE II
0	1	[2019-10-31 19:16:01.414]	69	26.3	915.76	27.05 inHg	939.24	27.74 inHg
1	2	[2019-10-31 19:20:46.462]	77	26.1	916.67	27.07 inHg	940.17	27.77 inHg
2	3	[2019-10-31 19:30:46.919]	77	26	919.63	27.16 inHg	943.2	27.86 inHg
3	4	[2019-10-31 19:40:47.380]	87	25.4	920.54	27.19 inHg	944.13	27.88 inHg
4	5	[2019-10-31 19:50:47.836]	89	25.2	920.61	27.19 inHg	944.21	27.89 inHg
5	6	[2019-10-31 20:00:48.292]	89	25.1	920.55	27.19 inHg	944.15	27.88 inHg
6	7	[2019-10-31 20:10:48.748]	90	25.2	920.5	27.19 inHg	944.09	27.88 inHg
7	8	[2019-10-31 20:20:49.209]	90	25.2	920.4	27.18 inHg	943.99	27.88 inHg
8	9	[2019-10-31 20:30:49.663]	90	25.1	920.44	27.18 inHg	944.03	27.88 inHg
9	10	[2019-10-31 20:40:50.109]	89	25.1	920.21	27.18 inHg	943.8	27.87 inHg
10	11	[2019-10-31 20:50:50.569]	90	25.1	920.26	27.18 inHg	943.85	27.88 inHg
11	12	[2019-10-31 21:00:51.030]	91	25.1	920.29	27.18 inHg	943.88	27.88 inHg
12	13	[2019-10-31 21:10:51.490]	86	25	920.03	27.17 inHg	943.61	27.87 inHg
13	14	[2019-10-31 21:20:51.951]	86	25	920.06	27.17 inHg	943.65	27.87 inHg
14	15	[2019-10-31 21:30:52.396]	88	25	919.88	27.17 inHg	943.46	27.86 inHg
15	16	[2019-10-31 21:40:52.857]	82	25	919.55	27.16 inHg	943.12	27.85 inHg
16	17	[2019-10-31 21:50:53.317]	87	25	919.57	27.16 inHg	943.14	27.85 inHg
17	18	[2019-10-31 22:00:53.778]	86	25	919.37	27.15 inHg	942.94	27.85 inHg
18	19	[2019-10-31 22:10:54.223]	86	25	919.36	27.15 inHg	942.93	27.85 inHg
19	20	[2019-10-31 22:20:54.684]	85	25	919.17	27.15 inHg	942.73	27.84 inHg
20	21	[2019-10-31 22:30:55.145]	89	25.1	919.72	27.16 inHg	943.3	27.86 inHg



	0
0	87.6358
1	87.3433
2	86.9624
3	86.5907
4	86.1261
5	85.4933
6	84.7933
7	84.0102
8	83.0692
9	82.079
10	81.0805
11	80.2095
12	79.4978
13	79.0406
14	78.9473
15	79.0187
16	78.9832
17	78.9562
18	79.0181



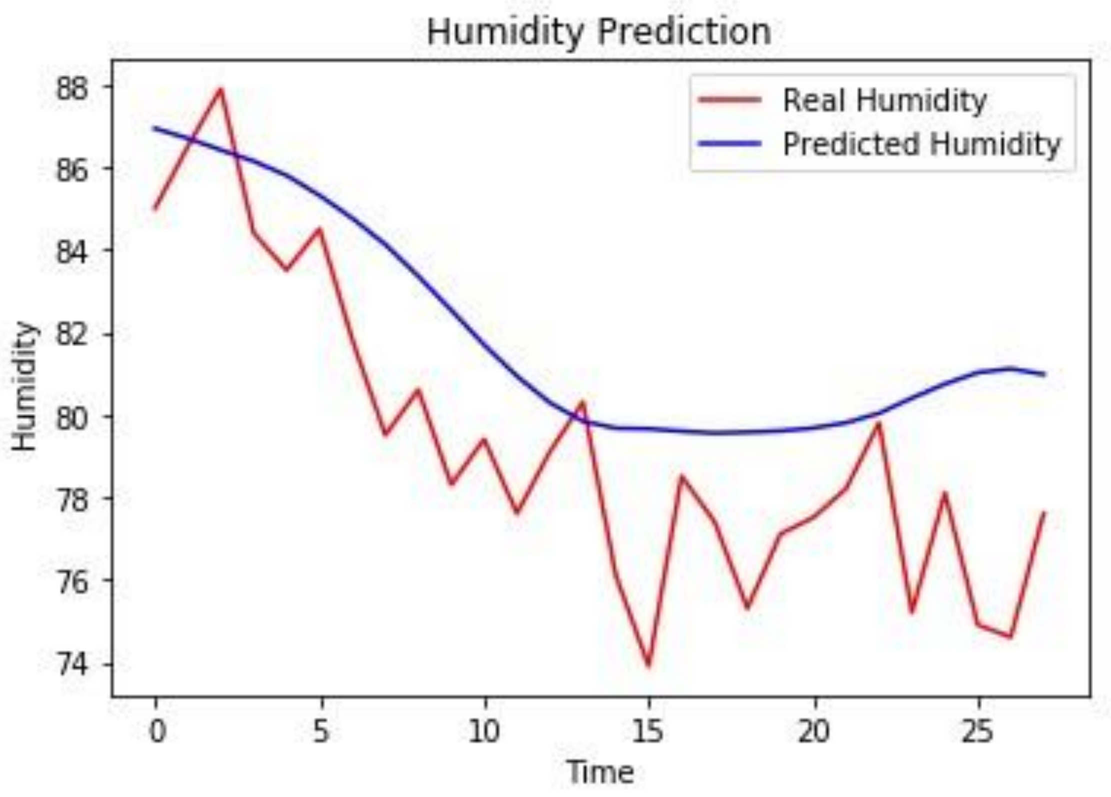
	0
0	910.599
1	910.104
2	909.59
3	909.111
4	908.722
5	908.407
6	908.175
7	907.981
8	907.84
9	907.747
10	907.695
11	907.615
12	907.469
13	907.199
14	906.83
15	906.502
16	906.268
17	906.175
18	906.17

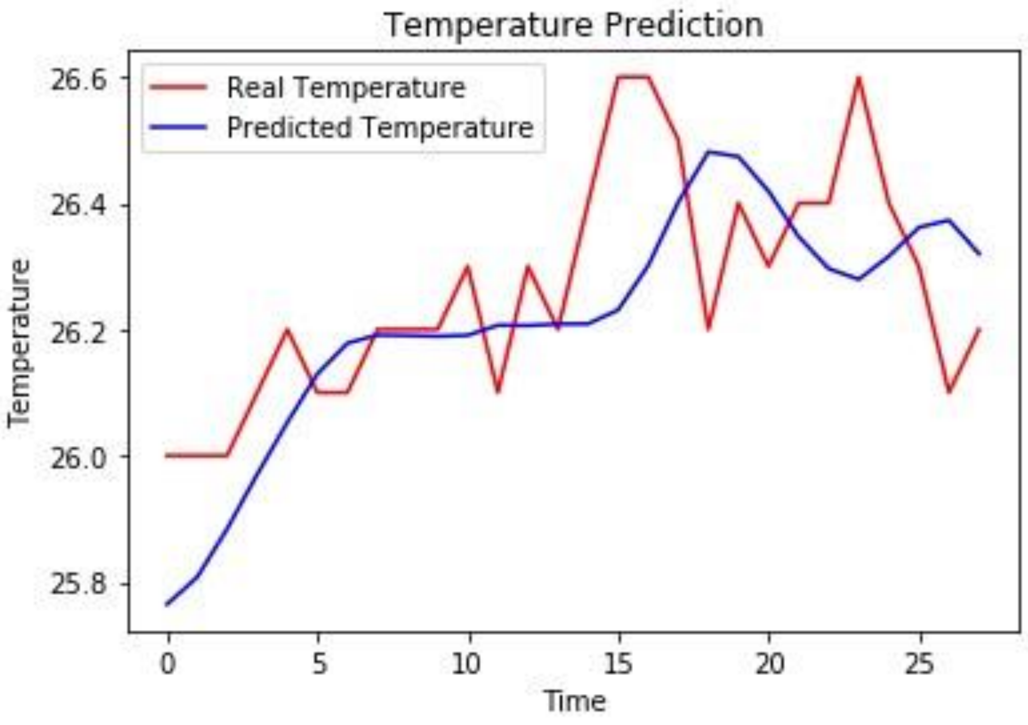


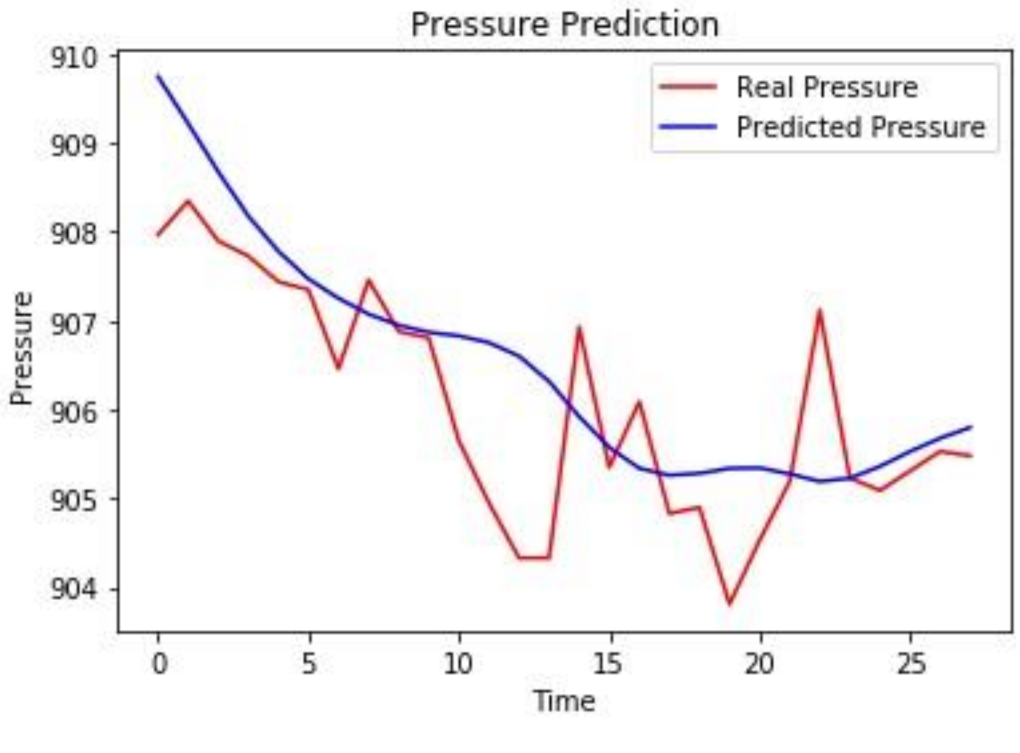
25.6123
25.65
25.725
25.8104
25.8892
25.9595
25.9986
26.0009
25.9919
25.9869
25.989
26.0086
26.0116
26.0163
26.0173
26.0399
26.1116
26.2121
26.2865

PLOTTING OUR RESULT:

1)HUMIDITY







ZEMBRETTI ALGORITHM:

Such simple algorithms are empirical and for the same reason very local, so depending on your location they can have very bad performance. In the other hand, some people argue that properly calibrated they can be over 90% accurate.
Most of them are based in the Zambretti algorithm or some variant of it. This algorithm was originally implemented in a forecaster device produced by the firm Negretti and Zambra in the early XX century.

It considers the absolute value of the pressure, the trend of the pressure, the season and the wind direction (although wind direction and season have a small impact in the output). The algorithm is well described and translated to formulas in this article, but in summary it works calculating a tabulated forecast number ZZ as follows:

1. From your measured pressure PP, temperature in Celsius TT and the altitude in meters hh compute the atmospheric pressure reduced to sea level P0P0. There are many formulas for this, a common one is:

P0=P(1−0.0065hT+0.0065h+273.15)−5.257P0=P(1−0.0065hT+0.0065 h+273.15)−5.257

2. Compute the pressure trend and
   - If the pressure is **falling** compute the forecast number as Z=130−P081Z=130−P081
   - If the pressure is **steady** compute the forecast number as Z=147−5P0376Z=147−5P0376
   - If the pressure is **rising** compute the forecast number as Z=179−2P0129Z=179−2P0129
3. Adjust ZZ for wind direction:
   - For Northerly winds adjust Z=Z+1Z=Z+1
   - For Southerly winds adjust Z=Z−2Z=Z−2
4. Adjust ZZ for the season:
   - If Winter adjust Z=Z−1Z=Z−1
   - If Summer adjust Z=Z+1Z=Z+1
5. Look up for the forecast from the following table:

THE FORECAST LETTER SCALE

The Forecast shown on the reverse of the instrument is a table of descriptions of short term weather changes against the letters of the alphabet. Fine weather has letters early in the alphabet and poor weather, late. The descriptions against each letter are as follows:

A	Settled Fine
B	Fine Weather
C	Becoming Fine
D	Fine Becoming Less Settled
E	Fine, Possibly showers
F	Fairly Fine, Improving
G	Fairly Fine, Possibly showers, early
H	Fairly Fine Showery Later
I	Showery Early, Improving
J	Changeable Mending
K	Fairly Fine , Showers likely
L	Rather Unsettled Clearing Later

M Unsettled, Probably Improving

N Showery Bright Intervals

O Showery Becoming more unsettled

P Changeable some rain

Q Unsettled, short fine Intervals

R Unsettled, Rain later

S Unsettled, rain at times T Very Unsettled, Finer at times U Rain at times, worse later.

V Rain at times, becoming very unsettled

W Rain at Frequent Intervals

X Very Unsettled, Rain

Y Stormy, possibly improving Z Stormy, much rain

CODE FOR ZAMBRETTI ALGORITHM:

```
package com.PrimesPackage;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
                    // write your code here
        //ZAMBRETI ALGORITHM

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter you measured pressure");        double P=sc.nextDouble();
        System.out.println("Enter your measured Temperature in
Celcius");
        double T=sc.nextDouble();
        System.out.println("Altitude in meters");;        double h =
sc.nextDouble();

        double Pf = P*(1-((0.0065*h)/(T+0.0065*h+273.15)));

        double Pfinal=Math.pow(Pf,-5.27);  //atmospheric pressure reduced to sea level
double Z=0.0;        if(Pfinal>P)
        {
            System.out.println("Pressure is rising");
            Z=179-((2*Pfinal)/129);
        }
        else if(Pfinal<P)
        {
          System.out.println("Pressure is falling");
          Z=130-(Pfinal/81);
        }
        else
          Z=147-(5*Pfinal/376);
        int choice1;

        System.out.println("Adjustment of value of Z based on Wind
Direction");
        System.out.println("1. Northerly Winds");        System.out.println("2.Southerly Winds");
        choice1=sc.nextInt();
        if(choice1==1)
        {
          Z=Z+1;
        }
          else if(choice1==2)
        {
          Z=Z-2;
        }       else
          System.out.println("Wrong choice");

        System.out.println("Adjustment For Season");
        System.out.println("1..Winter");
System.out.println("2..Summer");        int
choice2=sc.nextInt();        if(choice2==1)            Z=Z-1;
else if(choice2==2)
          Z=Z+1;

        if(Z>32)            Z=Z%32;


        if(Z==1)
        {
          System.out.println("Settled Fine");
        }
        else if(Z==2)
        {
          System.out.println("Fine Weather");
        }
        else if(Z==3)
        {

          System.out.println("Fine Becoming Less Setled");
        }
        else if(Z==4)
        {
```

```
   System.out.println("Fairly fine Showery Later");

}
else if(Z==5)
{

   System.out.println("Showering Becoming More unsettled");
}
else if(Z==6)
{

   System.out.println("Unsettled, Rain Later");
}
else if(Z==7)
{
   System.out.println("Rain at times, Worse Later");

}
else if(Z==8)
{
   System.out.println("Rain at times ,Becoming Very unsettled");

}else if(Z==9)
{
   System.out.println("Very Unsettled, Rain");
}
else if(Z==10)
{
   System.out.println("Settled Fine");
}
else if(Z==11)
{
   System.out.println("Fine Weather");
}
else if(Z==12)
   {
   System.out.println("Fine,Possibly Showers");
}
else if(Z==13)
{
   System.out.println("Fairly Fine, Showers Likely");
}
else if(Z==14)
{
   System.out.println("Showery,Bright Intervals");
}
else if(Z==15)
```

```
{
   System.out.println("Changeble,Some Rain");
}
else if(Z==16)
{
   System.out.println("Unsettled, Rain at times");
}
else if(Z==17)
{
   System.out.println("Rain at frequent Intervals");
}
else if(Z==18)
{
   System.out.println("Very Unsettled,Rain");
}
else if(Z==19)
{
   System.out.println("Stormy, Much Rain");
}
else if(Z==20)
{
   System.out.println("Settled,Fine");
}
   else if(Z==21)
{
   System.out.println("Fine Weather");
}
else if(Z==22)
{
   System.out.println("Becoming Fine");
}
else if(Z==23)
{
   System.out.println("Fairly Fine, Improving");
}
else if(Z==24)
{
   System.out.println("Fairly Fine, Possibly Showers early");
}
else if(Z==25)
{
   System.out.println("Showery Early,Improving");
}
else if(Z==26)
{
   System.out.println("Changeble,Mending");
}
```

```
else if(Z==27)
{
  System.out.println("Rather Unsettled, Clearing Later");
}
else if(Z==28)
{
  System.out.println("Unsettled,Probably Improving");
}
else if(Z==29)
{
  System.out.println("Unsettled, Short Fine Intervals");
  }
else if(Z==30)
{
  System.out.println("Very Unsetttled, Finer At Times");
}
else if(Z==31)
{
  System.out.println("Stormy, Possibly Improving");
}
else if(Z==32)
{
  System.out.println("Stormy, Much Rain");
}      else
  System.out.println("Unsettled,Changeble");
```

FORECASTED WEATHER:

SW to NW, 30.10 to 30.20 and steady - Fair with slight temperature change for 1 to 2 days.

* SW to NW, 30.10 to 30.20 and rising rapidly - Fair, followed within 2 days by rain.

* SW to NW, 30.20 and above and stationary - Continued fair, with no decided temperature change.

* SW to NW, 30.20 and above and falling slowly - Slowly rising temperature and fair for 2 days.

* S to SE, 30.10 to 30.20 and falling slowly - Rain within 24 hours.

* S to SE, 30.10 to 30.20 and falling rapidly - Wind increasing in force, with rain within 12 to 24 hours.

Forecast number ;	Forecast letter ;	Forecast Text ;	Pressure
1;	A;	Settled Fine;	1033
2;	B;	Fine Weather;	1023
3;	E;	Fine, Possibly showers;	1014
4;	K;	Fairly Fine , Showers likely;	1008
5;	N;	Showery Bright Intervals;	1000
6;	P;	Changeable some rain;	994
7;	S;	Unsettled, rain at times;	989
8;	W;	Rain at Frequent Intervals;	981
9;	X;	Very Unsettled, Rain;	974
10;	Z;	Stormy, much rain;	960

Conclusions

To obtain an on-demand weather forecast from an electronic IOT barometer based upon the Zambretti Forecaster, the following steps must be built into the embedded firmware:

A) The determination of the time-derivative of the pressure.

B) On the basis of A), select the appropriate equation (4), (5) or (6)

C) Measure the current pressure

D) Apply the appropriate equation from B) to obtain a value of Z

E) Modify Z according to wind direction data if available,

F) Modify Z according to the Season,

G) Use the modified Z to look up forecast in the storage and print the forecasted result.