DIH Project Report

2017-18

Probabilistic Approach Based Anomaly Detection Techniques for Real Time Systems

Submitted By: -

Shikhar Gupta (15035040)

Ceramic Engineering (part IV)

Rahul (16084012)

Electrical Engineering (part III)

Supervised By: -

Dr Hari Prabhat Gupta

Assistant Professor

Dept. of Computer Science

Probabilistic Approach Based Anomaly Detection Techniques for Real Time Systems

I acknowledge that Mr. Shikhar Gupta (15035040, B.Tech-IV, Ceramic Eng.) and Mr. Rahul (16084012, IDD-III, Electrical Eng.) has completed a project on above topic during Summers 2017-18 under my supervision.

(Signature of the Supervisor)

Dr Hari Prabhat Gupta

Assistant Professor

Dept. of Computer Science

OVERVIEW: -

Anomalies are strange data points, they usually represent an unusual occurrence of an event. Anomaly detection is presented from the perspective of Wireless sensor networks. Different approaches have been taken in the past, as we will see, not only to identify anomalies, but also to establish the statistical properties of the different methods. In a data-driven culture, companies approach decision-making from a quantitative point of view and rely less on gut feeling when making major decisions. Lack of accurate, timely or relevant data from across the business is also a major concern among companies with primitive or basic capabilities.

In data mining, anomaly detection is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically, the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are also referred to as outliers, novelties, noise, deviations and exceptions etc.

In this project we are solely focused on Anomaly detection concerning with Sensor output data. Our objective will be limited to detect Sensor giving anomalous output (if any) based on past precedents of it.

ANOMALY DETECTION IN SENSORS: -

An anomaly is a relatively rare event and, hence, suffers from the accuracy paradox. Anomaly detection as a data science technique is used in many applications, from the IoT to sensor-based data with the rise of the industrial Internet and the explosion of sensor data.

With deployment of sensors in rapidly expanding IoT industry, sensors are going to be crucial for all IoT applications. Some examples include self-driving car, auto-pilot in airplanes, self-functioning household appliances etc. These all will be using outputs from sensors installed in them, to model environment around them. Entire decision making by these IoT devices will rely solely on what data their sensors are giving to them thus, reliability of these sensor becomes very crucial.

Decision making based on Sensor output data is the core of IoT devices. Any sensor failure or sensor anomaly will influence decision making & this can have moderate to severe consequences on the functioning of device. Some example can be, speed sensor malfunctioning in self-driving cars or oven overheating due to fault in the temperature measuring device.

Hence, early detection of faulty or “Anomalous Sensor” is the desired goal to ensure enhanced protection and functioning of IoT device in case of such any eventuality.

OUR APPROACH: -

For modelling a sensor anomaly detection platform, we used statistical technique called Hidden Markov Model. A Hidden Markov Model is a finite set of states, each state is linked with a probability distribution. Transitions among these states are governed by a set of probabilities called transition probabilities. In a particular state a possible outcome or observation can be generated which is associated symbol of observation of probability distribution also called emission probabilities.

It is only the outcome, not the state that is visible to an external observer and therefore states are ``hidden'' to the outside, hence the name Hidden Markov Model.

Hence, Hidden Markov Model is a perfect solution for addressing detection of individual Sensor Anomalies while working with an array of sensors used to model environments in an IoT device.

In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters, while in the hidden Markov model, the state is not directly visible, but the output or observations, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens called emission probabilities. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states.

HMM Modelling: -

HMM based on where they are applied can be classified into three types. These are: -

1. *The Evaluation Problem:*

   Given an HMM 'λ' and a sequence of observations O = $o_1, o_2, \ldots\ldots, o_T$, what is the probability that the observations are generated by the model, $P\{O|\lambda\}$?

2. *The Decoding Problem:*

   Given a model 'λ' and a sequence of observations O = $o_1, o_2, \ldots\ldots, o_T$, what is the most likely state sequence in the model that produced the observations?

3. *The Learning Problem:*

   Given a model 'λ' and a sequence of observations O = $o_1, o_2, \ldots\ldots, o_T$, how should we adjust the model parameters $\{A,B,\pi\}$ in order to maximize $P\{O|\lambda\}$.

In this project we are using the second kind of HMM problem or the decoding problem, but in a slightly different way. Instead of finding out what the most probable underlying sequence of state is, we will simply calculate the probability of desired sequence of state, i.e. given a sequence of input sensor data our objective will be to calculate the probability of continuous anomaly in the underlying hidden state. With each sensor that is deviating from the expected pattern we will output its probability of being an anomaly.

DATASET INFORMATION: -

Time series data generated from 8 MOX gas-sensors were used for sensor anomaly detection modelling. A chemical detection platform composed of 8 chemo-resistive gas sensors was exposed to turbulent gas mixtures generated naturally in a wind tunnel. The acquired time series of the sensors were used.

The experimental setup was designed to test gas sensors in realistic environments. Wind tunnel with two independent gas sources that generate two gas plumes was utilized and gas sensors captured the spatio-temporal information contained in the gas plumes. Each measurement, which had a total duration of 300 seconds, was performed as follows: Initially no gas was released and clean air flowed along the wind tunnel. 60 seconds after, both sources started to release the corresponding volatile at the specified flow rate. The duration of the gas release was 180 s. Finally, the system acquired the recovery to the baseline for another 60 s.

This data-set have 11 attributes of which sensor data is in 3-10 (total 8 chemical sensors) attributes. Data is in time series and down-sampled to 10 instances per second for 300 seconds i.e. a total of ~2970 instances. Total of 4 similar data-sets, was used out of which 1 to base the model on & other three to test the model on.

- Initial data-sets are coded as:

000_Et_H_CO_n, 032_Et_H_CO_n, 066_Et_H_CO_n, 098_Et_H_CO_n. 000_Et_H_CO_n dataset is used to base the model on.

- Edited data-sets are coded as:

000_Et_H_CO_n_7_1.30_1950_2150, 032_Et_H_CO_n_3_1.50_1050_1230

where after _n, _7 denotes no of anomalous sensor (out of 0-7), _1.30 denotes 30% increase in attributes values of 7th sensor, _1950 & _2150 denotes increase is from instance no. 1950 to instance no. 2150.

Each file includes the acquired time series, presented in 11 columns, includes the following sensors data: -

Time (s), Temperature (°C), Relative Humidity (%), and the readings of the 8 gas sensors: TGS2600, TGS2602, TGS2602, TGS2620, TGS2612, TGS2620, TGS2611, TGS2610

The raw acquired time series were provided, and also time series down sampled at 100ms. In our project we used down-sampled version.

Dataset Preview: -

Filename: 000_Et_H_CO_n.csv

```
3.00,22.46,40.73,331.00,273.00,518.00,572.00,566.00,700.00,629.00,719.00

3.10,22.46,40.73,331.00,274.00,518.00,571.00,566.00,700.00,630.00,719.00

3.20,22.46,40.73,333.00,273.00,518.00,571.00,566.00,701.00,630.00,719.00

.    .    .    .    .    .    .    .    .    .    .    .    .

.    .    .    .    .    .    .    .    .    .    .    .    .
```

. . . .

up-to around 2950 instances.

Total of 4 types of such dataset which were formed under similar conditions were used in HMM modelling.

IMPLEMENTATION:

Sequential steps followed in the implementation is as follows:

- First of all, manipulation of dataset values was done so as to make them workable i.e. from csv file to lists etc. Data belonging to each sensor was put in 'Lists' for both base(training) data and testing data.
- Time-Series data from each sensor was assigned sequentially to a fixed no. of vertical blocks based on its absolute value distribution.
- Each instance of the dataset was considered as an observation, continuous sets of these observations were used to calculate the probability of hidden/underlying anomalous sequences using Hidden-Markov model
- Deviations in no. of blocks b/w base data & test data results in the probability of anomaly of the maximum deviation sensor for each observation to form Emission Matrix of the Hidden-Markov Model.
- For a given no. of continuous such deviations, each such sensor was categorized as anomalous unless maximum deviation sensor changes.

The Python code of implementation is: -

Filename: Anomaly_Detection.py

```
*********************************          BEGIN          *****************************************

import matplotlib.pyplot as plt
from pandas import read_csv

########################   CONTROL PARAMETERS   ##############################
zz = 2970    # length of dataset
ds = 1       # downsampling
div = 20     # no. of vertical divisions
Catch = 100  # no of least countinuous max sensors instance to be caught as an anomaly
Cont = 5    # sequences of data to be used to calculate hidden markov model probability
##################    training data       ############################
train_filename = "000_Et_H_CO_n.csv"

#################  testing data with 1.25 times   ############################
#test_filename = "000_Et_H_CO_n_7_1.30_1950_2150.csv"
#test_filename = "000_Et_H_CO_n_7_1.50_1950_2150.csv"
#test_filename = "032_Et_H_CO_n_3_1.25_1050_1230.csv"
#test_filename = "032_Et_H_CO_n_3_1.50_1050_1230.csv"
test_filename = "066_Et_H_CO_n_3_1.25_2200_2450.csv"
#test_filename = "066_Et_H_CO_n_3_1.50_2200_2450.csv"
#test_filename = "098_Et_H_CO_n_2_1.30_1350_1550.csv"
#test_filename = "098_Et_H_CO_n_2_1.50_1350_1550.csv"

# train sensors : collectionn of individual sensor inputs into packets of ds data each
train_dataset = read_csv(train_filename)
train_array = train_dataset.values

a,b,c,d,e,f,g,h = [],[],[],[],[],[],[],[]
train_sensors = [a,b,c,d,e,f,g,h]
count = 3
sum =0
n=1
for i in train_sensors:
   while(n<zz and count<11):
      for k in range(ds*(n-1),ds*n):
         sum += train_array[k][count]
      i.append(sum/ds)
      sum=0
      n+=1
   count+=1
   n=1
```

```
plt.plot(a)
plt.plot(b)
plt.plot(c)
plt.plot(d)
plt.plot(e)
plt.plot(f)
plt.plot(g)
plt.plot(h)
plt.legend(['1', '2', '3', '4','5','6','7','8'], loc='upper left')
plt.show()
print("Training data plot")

# test_sensors : collectionn of individual sensor inputs into packets of ds data each
test_dataset = read_csv(test_filename)
test_array = test_dataset.values

aa,bb,cc,dd,ee,ff,gg,hh = [],[],[],[],[],[],[],[]
test_sensors = [aa,bb,cc,dd,ee,ff,gg,hh]
count = 3
sum =0
n=1
for i in test_sensors:
  while(n<zz and count<11):
    for k in range(ds*(n-1),ds*n):
      sum += test_array[k][count]
    i.append(sum/ds)
    sum=0
    n+=1
  count+=1
  n=1

plt.plot(aa)
plt.plot(bb)
plt.plot(cc)
plt.plot(dd)
plt.plot(ee)
plt.plot(ff)
plt.plot(gg)
plt.plot(hh)
plt.legend(['1', '2', '3', '4','5','6','7','8'], loc='upper left')
plt.show()
print("Testing data plot")

######################          min max fitting          ###################
minimum,maximum,width = [],[],[]
for i in range(8):
  j = train_sensors[i]
```

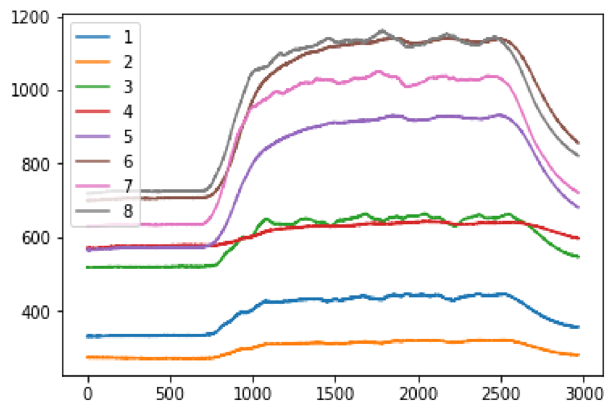
```
    k = test_sensors[i]
    if min(j)<min(k):
        m = min(j)
    else:
        m = min(k)
    if max(j)>max(k):
        n = max(j)
    else:
        n = max(k)
    minimum.append(m)
    maximum.append(n)
    w = (n - m)/div
    width.append(w)

###################          Vertical disribution        #####################

m,n,o,p,q,r,s,t = [],[],[],[],[],[],[],[]
train_values = [m,n,o,p,q,r,s,t]

mm,nn,oo,pp,qr,rr,ss,tt = [],[],[],[],[],[],[],[]
test_values = [mm,nn,oo,pp,qr,rr,ss,tt]

for i in range(8):
    j = train_values[i]
    k = test_values[i]
    w = width[i]
    m = minimum[i]
    for l in train_sensors[i]:
        for n in range(div):
            if l >= (m + n*w) and l < (m + (n+1)*w):
                j.append(n)
            elif l == (m +div*w):
                j.append(div-1)
    for l in test_sensors[i]:
        for n in range(div):
            if l >= (m + n*w) and l < (m + (n+1)*w):
                k.append(n)
            elif l == (m + div*w):
                k.append(div-1)

plt.plot(train_values[0])
plt.plot(train_values[1])
plt.plot(train_values[2])
plt.plot(train_values[3])
plt.plot(train_values[4])
plt.plot(train_values[5])
plt.plot(train_values[6])
plt.plot(train_values[7])
plt.legend(['1', '2', '3', '4','5','6','7','8'], loc='upper left')
plt.show()
print("Vertical Assignment of Training data plot")

plt.plot(test_values[0])
plt.plot(test_values[1])
plt.plot(test_values[2])
plt.plot(test_values[3])
plt.plot(test_values[4])
plt.plot(test_values[5])
plt.plot(test_values[6])
plt.plot(test_values[7])
plt.legend(['1', '2', '3', '4','5','6','7','8'], loc='upper left')
plt.show()
print("Vertical Assignment of Testing data plot")

####################      Variation in data streams       ######################
var_stream = []

for l in range(2960):
   diff_array = []  # difference of sensor values b/w training & testing per instance
   for i in range(8):
      j = train_values[i]
      k = test_values[i]
      if (j[l] - k[l]) >= (k[l] - j[l]):
         diff_array += [ j[l] - k[l] ]
      else:
         diff_array += [ k[l] - j[l] ]
   var_stream += [diff_array]

ga,gb,gc,gd,ge,gf,gg,gh = [],[],[],[],[],[],[],[]
gaps = [ga,gb,gc,gd,ge,gf,gg,gh]

for i in range(8):
   for j in var_stream:
      gaps[i].append(j[i])

plt.plot(ga)
plt.plot(gb)
plt.plot(gc)
plt.plot(gd)
plt.plot(ge)
plt.plot(gf)
plt.plot(gg)
plt.plot(gh)
plt.legend(['1', '2', '3', '4','5','6','7','8'], loc='upper left')
plt.show()
print("Training And Testing Differnce")

###################    *********    Markov Model   *********   ######################

###################    Emmision matrix probability assignment    ###################
emm_prob = []
sensor_max_list = []
for i in var_stream:
    g_sum=0
    for j in i:
        g_sum += j
    emm_prob.append(8*max(i) - g_sum)
    for k in range(8):
        if max(i) == i[k]:
            sensor_max_list.append(k)
            break
#### normalisation
z = max(emm_prob)
for i in range(len(emm_prob)):
    emm_prob[i] = emm_prob[i]/z

##############  Observation data with "sensor_max_list" and "emm_prob" combined
sequence = []
Observation = []
temp = []
t = []
for i in range(len(sensor_max_list)):
    if i+1 == len(sensor_max_list):
        temp.append([sensor_max_list[i],emm_prob[i]])
        Observation += [temp]
        t.append(i)
        sequence += [t]
    else:
        if sensor_max_list[i] == sensor_max_list[i+1]:
            temp.append([sensor_max_list[i],emm_prob[i]])
            t.append(i)

        else:
            temp.append([sensor_max_list[i],emm_prob[i]])
            Observation += [temp]
            temp = []
            t.append(i)
            sequence += [t]
            t = []

##############    transition matrix and initial state matrix
```

```
pi = 0.5
A = [1,0.5]

###############    probability of observation for each block(cont instance each) of data
P = []
for i in Observation:
   if len(i)>Catch:
      Ps = []
      n = len(i)//Cont   # n = len(i)-Catch
      Ps.append( i[0][0] )
      for c in range(n):
         p = ( i[c*Cont][1] * pi )
         for j in range(1,Cont-1):
            p = p * ( i[c*Cont+j][1]) * A[0]
         Ps.append( p )
      P += [Ps]

###################   ************  Final Selection  ***********   ###################
count        = []     # Location Track of Anomaly
intensity    = []     # No of Countinious Anomalous Blocks
A_sensor     = []     # Anomalous Sensor Number (0-7)
A_prob       = []     # Probability of anomalous sequence
for i in P:
   A_sensor.append(i[0])
   intensity.append(len(i)-1)
   c = len(i)
   s = 0
   for j in range(1,c):
      s += i[j]
   A_prob += [ s/(c-1) ]

for i in range(len(sequence)):
   k = sequence[i][0]
   if len(sequence[i])>Catch:
      count += [k]

######################          Print it out      ##################################

print("Seq.   Track   Sensor no.   Intensity   Probability of Anomaly")
for i in range(len(A_sensor)):
   print(" %-5d %-7d %-12d %-14d %-12.10f"%(i+1,count[i],A_sensor[i],intensity[i],A_prob[i]))

####################################      END     #########################################
```

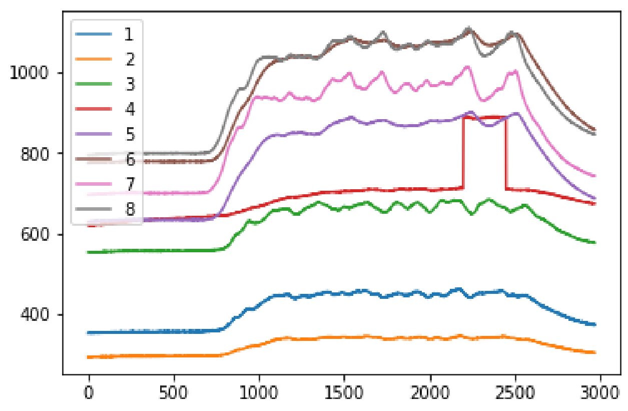
CODE OUTPUT: -

Code output gave visualisation of training data and testing data. Visualisation of sensor reading based vertical data distribution of both training and testing data was yielded on code execution. A line chart with one chart showing progression of time series of all 8 gas sensors was used. Difference b/w these vertical distributions of each sensors which were used to calculate emission probabilities were also drawn on a line chart.

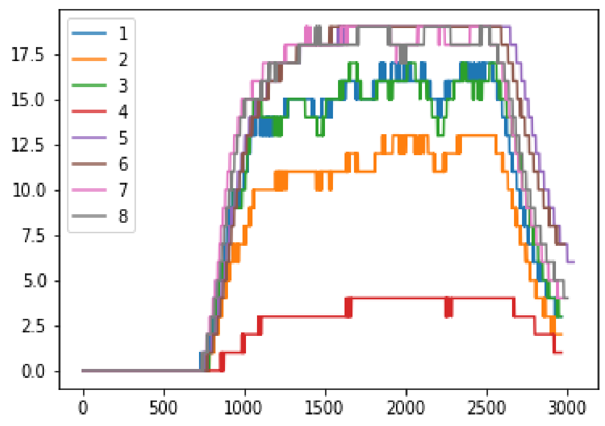
Ex: - Visualisation Output: -



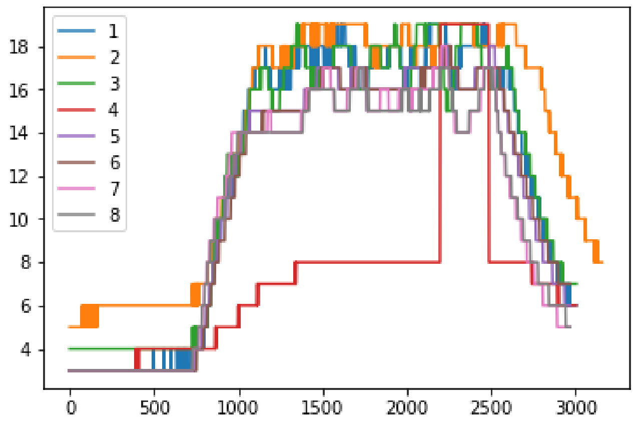
Training data plot



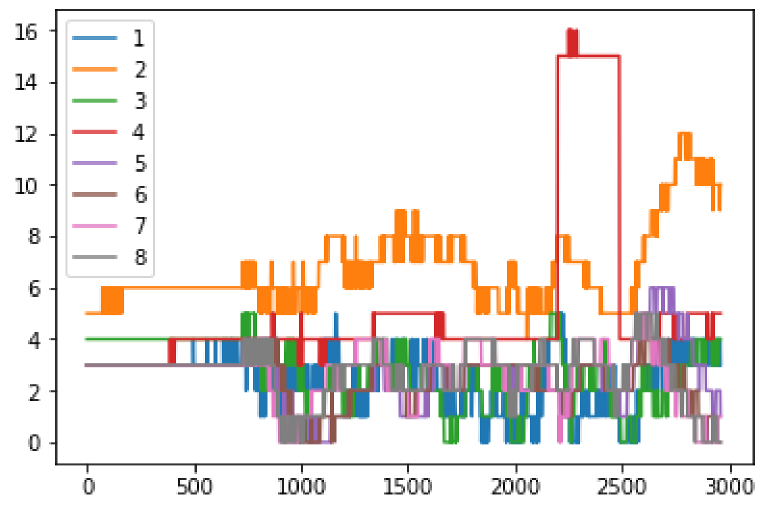
Testing data plot



Vertical Assignment of Training data plot



Vertical Assignment of Testing data plot



Training and Testing Difference Plot

Finally, sequential output carrying the following items was yielded as a result of the code execution: -

- Anomalous Sensors (b/w 0 to 7) caught by the Model.
- Their respective probability of anomaly.
- Sequential location of the anomaly.
- Intensity describing, for how long does this anomaly remains.

Ex: -

Seq.	Track	Sensor no.	Intensity	Probability of Anomaly
1	0	1	219	0.0000242600
2	2198	3	28	0.1480821534
3	2487	1	47	0.0024076676

RESULT: -

Following observations were obtained out of numerous test cases that the algorithm was put through:

- Anomaly detection becomes easier & its probability increases as % change in attributes values is increased (say from 30% to 50%).
- Anomaly detection is easier at the start & medium but becomes difficult at the end of the dataset.
- Sensors whose output value range is larger (sensor no. 5,6 & 7) are detected at a higher % increase in their values.