**Report on project**

**3rd December 2018**

|  |  |
| --- | --- |
| Please find all the code for this project here -  <https://github.com/jithinJE/Fuel_Analysis> | |

## 

## Setup & Connect to IOT device

First task was to install an operating system for the RK3328-CC device.

I downloaded the Ubuntu 16.04 IOT firmware available from the website [here](http://en.t-firefly.com/doc/download/34.html).

I flashed the firmware image into a micro-SD card, and used the card to boot the IOT device.

Initially, I had to connect a physical HDMI monitor, USB keyboard to use the device, then I setup the device with default IP and gateway, which allows to use the IOT device as a Headless computer, and connect to it using ethernet cable.

Connect the device to laptop using ethernet cable and use ssh to login with username - firefly and password - firefly

## Generate Random Sensor values in device

As we don’t have actual sensors with us, I have written code to simulate sensor values by generating random values within the range of each sensor.

That is, I have downloaded the test dataset to the device. The program fetches the dataset, and randomly selects 1 row, and concatenates all the columns into a single list. Example - [293.15, 100.0, 0.0268554194, 406.2675941843, 75.12]

## Publish and Subscribe using MQTT

I am using MQTT protocol, which is the most popular one for communication in IOT scenarios.

I have installed Mosquitto, which is a MQTT broker in my laptop which acts as the server. I have also installed Paho (MQTT clients) on both laptop and IOT device.

In the IOT device, I have written code to Publish the above mentioned randomly generated data to the broker to using the topic name “sensor/fuel\_data”.

In the laptop, I have created a subscriber application that listens to the broker for a topic “sensor/fuel\_data”. When the data is found, it splits the string into the 5 columns of X & Y.

Subscriber then loads the 8 order non-linear regression model file saved earlier, and uses it to predict the Y value, for the IOT sensor values.

## Online / Adaptive Learning

First, I tried searching for libraries within scikit-learn, or other similar tools to perform online learning with new data incoming from the sensor.

But, as I have not yet found such library, I decided to develop code for implementing Gradient descent algorithm.

I have used Linear regression formula -

|  |
| --- |
| Y = a \* K + b \* Psi + c \* Th + d \* SV + e    Loss function =  Differentiated above equation to get gradients -  dY/da = 1/n \* 2 \* K \* ( Y\_hat - Y )  dY/db = 1/n \* 2 \* Psi \* ( Y\_hat - Y )  dY/dc = 1/n \* 2 \* Th \* ( Y\_hat - Y )  dY/dd = 1/n \* 2 \* SV \* ( Y\_hat - Y )  dY/de = 1/n \* 2 \* ( Y\_hat - Y ) |

In my code, I performed following steps -

* Read from both Train and Test dataset.
* Use scikit-learn to train a simple Linear Regression model, using Train dataset.
* Find the weight for this model - [weight\_K, weight\_Psi, weight\_Th, weight\_SV, weight\_Intercept]
* Use these weights as the initial weights for Gradient descent algorithm

While loop( take individual row from Test dataset individually )

// This loop imitates online learning of data coming from sensor

{

Use weights to find Predicted Y\_hat

Get actual Y

Loss function = 1/N \* ( Y\_hat - Y )2

Differentiate loss function to find each gradient -

gradient\_K = 1/n \* 2K \* ( Y\_hat - Y )

gradient\_Psi = 1/n \* 2Psi \* ( Y\_hat - Y )

gradient\_Th = 1/n \* 2Th \* ( Y\_hat - Y )

gradient\_SV = 1/n \* 2SV \* ( Y\_hat - Y )

gradient\_Int = 1/n \* 2 \* ( Y\_hat - Y )

Use gradients to update weights -

weight\_K = weight\_K - learning\_rate \* gradient\_K

weight\_Psi = weight\_Psi - learning\_rate \* gradient\_Psi

weight\_Th = weight\_Th - learning\_rate \* gradient\_Th

weight\_SV = weight\_SV - learning\_rate \* gradient\_SV

weight\_Int = weight\_Int - learning\_rate \* gradient\_Int

}

Once finished with Test set...I try to confirm that I have optimal weights, by comparing the mean squared error on entire Test dataset, with both - 1) My new weights vs 2) the scikit-learn linear regression model.

|  |  |
| --- | --- |
| **MSE for Scikit-learn Linear regression** | 1.06399 |
| **MSE for weights Learned Online** | 1.74647 |

The MSE has increased for Online learning. This could be because of incorrect Loss function used earlier.

Also, the above method is feasible only for Linear regression. For 8th order Polynomial regression, there are 495 terms and weights. Performing gradient derivation for each term is not feasible.

I need to find some library which will automate the process for me.

## Utilize GPU or parallel compuation using scikit-learn

Scikit-learn is not capable of utilizing GPU. For GPU we need to use tensorflow, which is capable of neural networks based regression.

I have managed to get access to aries.ecs.fullerton.edu which is a GPU server, provided by our department, and have setup remote Jupyter notebook on the server.

## Utilize Spark

I have installed Spark on my machine, and I am currently going through a Lynda.com tutorial on MLlib with Spark.

To utilize the RDD based spark on a cluster for parallel processing, I will need to work in the Cloud computing lab, where I can utilize the Hadoop Distributed File System.

## Components required for Sensors

|  |  |
| --- | --- |
| K Temperature (in deg C) | DHT11 sensor (available)  Jumper Wires (not available)  Breadboard (not available) |
| Psi Pressure (in Pa) | Not in kit  Some sensors found online -  BMP 180 (air pressure sensor) |
| Th Thermal Conductivity | Not found anything for Thermal Conductivity…  There are sensors for Electrical conductivity |
| Sv Sound Velocity | Not found any sensors |

As discussed above, due to non-availability of actual sensors, I have simulated the sensor value generation in the IOT device using random row selection from the Test dataset.

## Find Optimal Regression Model

I used Sckit-learn to develop following models, and found Non-linear Polynomial regression with order 8 to be optimal, within feasible time.

**Polynomial Regression (order = 8)**

|  |  |
| --- | --- |
| Parameteres ( degree – 8 )        Dataset size = **420,000** | Root Mean Squared Error = 0.211474802  Accuracy Measure - |

**EPSILON Support Vector Regression – RBF KERNEL**

|  |  |
| --- | --- |
| Epsilon SVR    Kernel = rbf  Parameters (C = 1, gamma = 0.001, epsilon = 0.2)    Dataset size = **420,000** | Root Mean Squared Error = 0.324    Accuracy Measure - (~88% for 0.5, ~99% for 1.0, 100% for 2+)    TIME – 12 hours Training, 1.5 hours Prediction |
| Epsilon SVR    Kernel = rbf  Parameters (C = 1, gamma = 0.001, epsilon = 0.2)    Dataset size = **100** | Root Mean Squared Error = 1.353097    Accuracy Measure - |
| Epsilon SVR    Kernel = rbf  Parameters (C = 1, gamma = 0.001, epsilon = 0.2)    Dataset size = **1000** | Root Mean Squared Error = 1.847696    Accuracy Measure - |
| Epsilon SVR    Kernel = rbf  Parameters (C = 1, gamma = 0.001, epsilon = 0.2)    Dataset size = **10,000** | Root Mean Squared Error = 0.44030782    Accuracy Measure - |
| Epsilon SVR    Kernel = rbf  Parameters (C = 1, gamma = 0.001, epsilon = 0.2)    Dataset size = **80,000** | Root Mean Squared Error = 0.33204623    Accuracy Measure - |
| Epsilon SVR    Kernel = rbf  Parameters (C = 1, gamma = 0.001, epsilon = 0.2)  Dataset size = **150,000** | Root Mean Squared Error = 0.3279102  Accuracy Measure -      TIME = 1.5 hours |

**EPSILON Support Vector Regression – LINEAR KERNEL**

|  |  |
| --- | --- |
| Epsilon SVR    Kernel = linear  Parameters (C = 1, gamma = 0.001, epsilon = 0.2)    Dataset size = **FULL** | Root Mean Squared Error = 0.    Accuracy Measure -    TIME = hours |

**DECISION TREES**

|  |  |
| --- | --- |
| Parameters (random\_state = 0)    Dataset size = **FULL** | Root Mean Squared Error = 0.26003824  Accuracy Measure -        TIME - IMMEDIATE ON GOOGLE COLAB |

**RANDOM FOREST**

|  |  |
| --- | --- |
| Parameters (n\_estimators = 10, random\_state = 0)    Dataset size = **FULL** | Root Mean Squared Error = 0.20716353    Accuracy Measure -    TIME - IMMEDIATE ON GOOGLE COLAB |
| Parameters (n\_estimators = 100, random\_state = 0)    Dataset size = **FULL** | Root Mean Squared Error = 0.200835872    Accuracy Measure -    TIME – 3 mins ON GOOGLE COLAB |

## Find Polynomial terms generated by Scikit-learn

I am able to generate the Polynomial terms for Non-Linear order 8, i.e. 495 terms. Please find in Github the text file Polynomials\_8th\_order.txt for list of terms. We can directly use this as a formula in C++.