TASK:
Implement analytic in Python to identify the regions of anomalies when the machine is RUNNING. Submit the Python code and the results.

Step 1: Importing the CSV file

I used the Google Colab platform to run my code. I was easily able to load the complete dataset and create a single DataFrame at once and had no issue. I changed the column name of time to 'TIMESTAMP.' The total number of rows in the dataset is 12661. The shape of the dataset 12661x47.

Step 2: Filling the NaN values in the dataset

I used the fillna() function in the pandas library to fill the NaN values using two methods 'ffill' and 'bfill.'

Step 3: Plotting the input speed sensor values

From the graph plotted, we see that in the first half of the dataset the input speed values are close to zero, and in the second half, the input speed is almost always above 5000 rpm with sporadic zero rpm speed. Therefore, I selected 4500 rpm as the threshold value for the running condition of the pump. In the next part, I created a separate DataFrame for the shutdown condition of the pump. Now we will only work for the running pump DataFrame (on).

The number of rows in the 'on' DataFrame is 5544. In the rest of the rows, the pump is shut down and stored in the 'shut' DataFrame. Our next task is to classify the 'on' DataFrame into normal and abnormal running conditions. These states are sufficient.

Step 4: Training Machine Learning models from the 'pyod' library

After looking up for the best anomaly detection algorithms, I found the 'pyod' library which was released recently as an opensource project. It has inbuilt Machine Learning as well as Deep Learning models for anomaly detection. Other popular approaches are also available for unsupervised learning like K-mean clustering using sklearn but the drawback of this approach is that we have to specify the number of clusters in the dataset which is unknown to us.

From the various algorithms available in the pyod library, I trained a model of ABOD(Angle-base Outlier Detection) class. This algorithm doesn't require many parameters and is fairly easy to use. The drawback of this algorithm is extremely high running time(n^3). The output of the model is a vector of binary values (0 for inliers and 1 for outliers).

As the running time for the algorithm is very high and the dataset we have has 47 features, it's best to reduce the dimensionality of the data. Using the PCA module in the sklearn library, I reduced the dimensionality to 15 features. Dimension - 5544x15

Using the describe() function on the DataFrame we get the description of the dataset according to each feature in the dataset. From the description, we see that some sensors have a maximum value of up to 6000 while others have maximum value as low as 30. Due to this large variation in the dataset, it is good practice to standardize the data. Not standardizing the data can give incorrect output as some algorithms assume the data is inherently normally distributed. StandardScaler() in sklearn is usually used for this purpose. I have used RobustScaler(), it performs the same function but is more robust to outliers.

The other model that I trained is a Deep Learning based AutoEncoder module. This algorithm builds a Neural Network (auto-encoder) and calculates the reconstruction score for the prediction of outliers. The number of neurons and layers and activation functions can be altered.

In both models, we have a parameter called 'contamination' which is the number of outliers in the dataset. This parameter is used to define the threshold for the decision function. The top 10% of the samples sorted according to the reconstruction errors will be labeled as outliers when the contamination is 0.1. I set the value to 0.1 for both the models, therefore the number of outliers is 10 percent of 5544 = 554.

The output of both the models is stored in the ab_state1 (ABOD) and ab_state2 (AutoEncoder) list. The graphs are plotted for three different ranges of values and both the models: (0-30), (0-350), (0-6000). The vertical shaded region in Yellow(ABOD) and Red(Auto Encoder) color depicts the outliers detected.

Step 5: TensorFlow RNN Encoder-Decoder

In the above two models, I reduced the number of features to 15 and it becomes impossible to determine which sensor or feature is contributing to the anomaly. To overcome this difficulty, I built an RNN Encoder-Decoder model using LSTM cells from scratch. The summary of the network can be viewed from the code.

After fitting the model on the dataset and predicting the values for each sample in the set, we can calculate the reconstruction error. I have calculated this error feature-wise and sample-wise and stored it in separate DataFrames.

To determine which sensors have more contributions in the anomaly detection, I set the threshold for sensor error to 0.431 which is the mean as well as the median of the feature-wise error values. I stored the result in Feature_contribution.csv. This CSV file gives a collective output for all anomalies. In order to determine the contribution of a sensor in each anomaly, the

error values for that individual sample can be extracted, and then depending on the values we can find the sensor contributing the most.

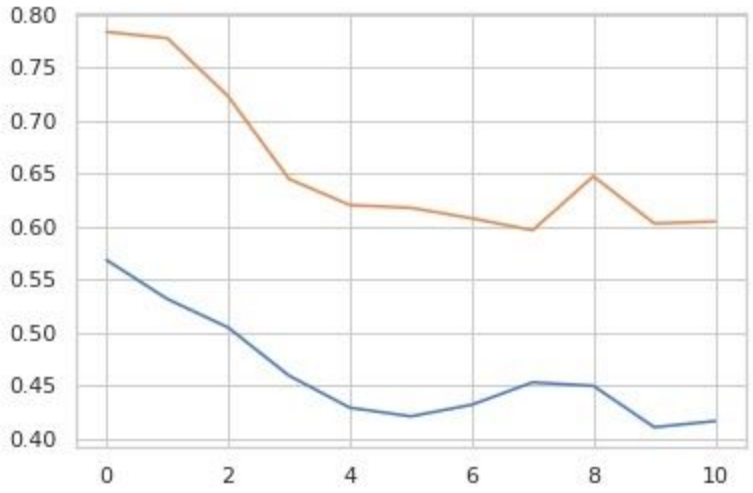
In the next step, I plotted the distribution of sample-wise errors. Using the distribution plot, we can select the appropriate threshold for our model. The graphs are plotted with blue colored shaded regions.

Finally, I merged all the DataFrame in one and exported it to a CSV file. This file has the timestamp and the condition of the pump - shutdown, true for an anomaly, false otherwise.

Overfitting / Underfitting:

In the ABOD model, the parameter n_neighbors can be tuned to avoid overfitting. This algorithm works in the same way as K-Nearest Neighbors. Therefore, increasing the number of neighbors decreases overfitting but increases the runtime. Although, increasing the number of neighbors can introduce underfitting.

The other models are Neural Networks, therefore, we can add the Dropout layer or change the number of neurons and layers to avoid underfitting and overfitting. Given below is the plot of validation loss and training loss for the RNN network with the number of epochs.



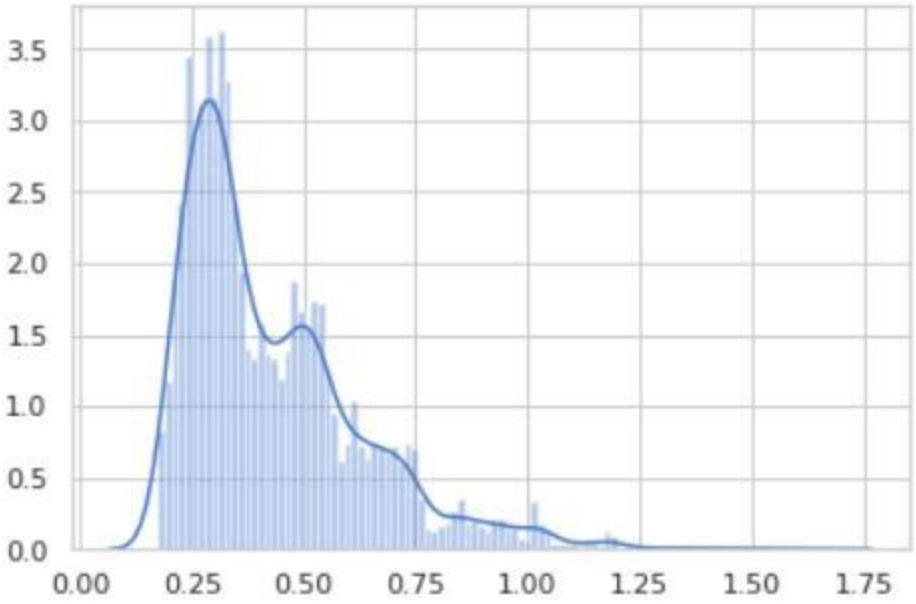
The number of anomalies:

In pyod models, we have a parameter called ‘contamination’ which is the number of outliers in the dataset. This parameter is used to define the threshold for the decision function. The top 10% of the samples sorted according to the reconstruction errors will be labeled as outliers when the contamination is 0.1. I set the value to 0.1 for both the models, therefore the number of outliers is 10 percent of 5544 = 554. For the RNN model, the threshold is 0.75 for

reconstruction error. All the samples above this value are labeled as an anomaly. The total number turns out to be 396.

Avoiding False Positive and Noise:

Given below is the distribution of the sample-wise error on the training set predicted using RNN. From the graph, we can select the appropriate value for a threshold which will decrease the chances of the anomaly being a noise. As the threshold is increased, only more prominent anomalies are detected. Similarly, for pyod models, decreasing the value of contamination is similar to increase the threshold.



Another way is to detect the anomalies which occur in a series and not just individual spikes in the data. Noise is sparsely distributed and usually occur in a single timestep, an anomaly, on the other hand, might last longer than that. Therefore, we can confidently detect anomalies.