Name: Rohith Kumar Poshala

rposhala@buffalo.edu

Anomaly Detection

In this project, deep learning approaches that include building a sequence to sequence MLP/LSTM and also an autoencoder with the help of Dense, LSTM, Conv1D layers individually to reconstruct and detect the anomalies in the benchmark dataset.

About Dataset:

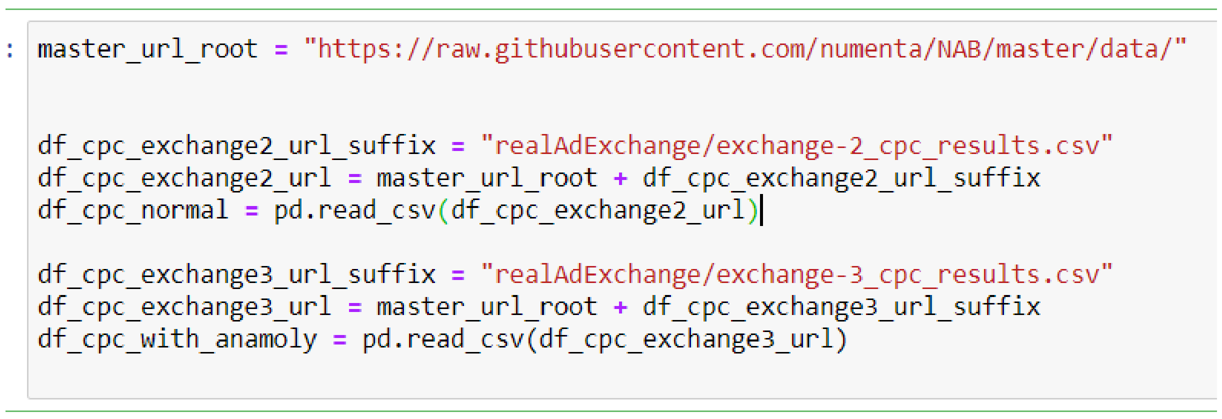
NAB (Numenta Anomaly Benchmark) is a novel benchmark for evaluating algorithms for anomaly detection in streaming, real-time applications. It is composed of over 50 data files designed to provide data for research in streaming anomaly detection. It is comprised of both real-world and artificial timeseries data containing labeled anomalous periods of behavior.

Link for NAB datasets GitHub Repo: https://github.com/numenta/NAB/tree/master/data

For this project, realAdExchange dataset (real-world data) from NAB was used. This dataset is Online Advertisement clicking rates, where the metrics are cost-per-click (CPC). One of the files is normal without anomalies and the other was with anomalies.

Loading the datasets:

Below are the commands to load a dataset from a public GitHub repository with the help of URL:

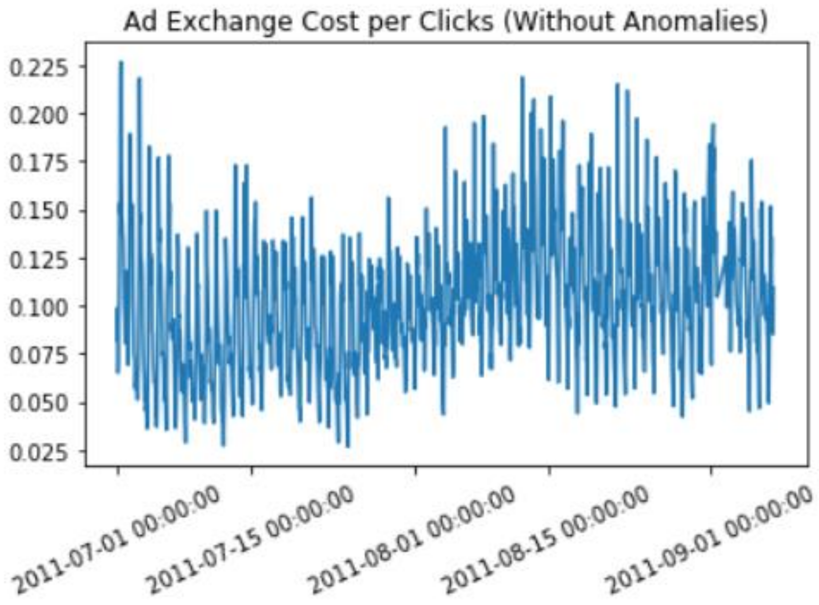


```
master_url_root = "https://raw.githubusercontent.com/numenta/NAB/master/data/"


df_cpc_exchange2_url_suffix = "realAdExchange/exchange-2_cpc_results.csv"
df_cpc_exchange2_url = master_url_root + df_cpc_exchange2_url_suffix
df_cpc_normal = pd.read_csv(df_cpc_exchange2_url)

df_cpc_exchange3_url_suffix = "realAdExchange/exchange-3_cpc_results.csv"
df_cpc_exchange3_url = master_url_root + df_cpc_exchange3_url_suffix
df_cpc_with_anamoly = pd.read_csv(df_cpc_exchange3_url)
```

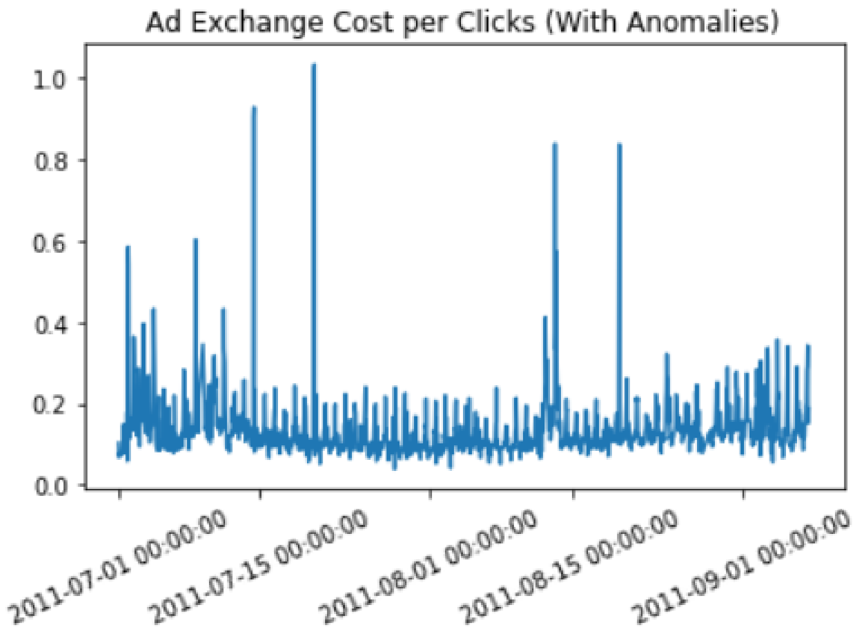
A function was defined to plot the datasets and also extract the key details on request.

Visualizing the datasets:

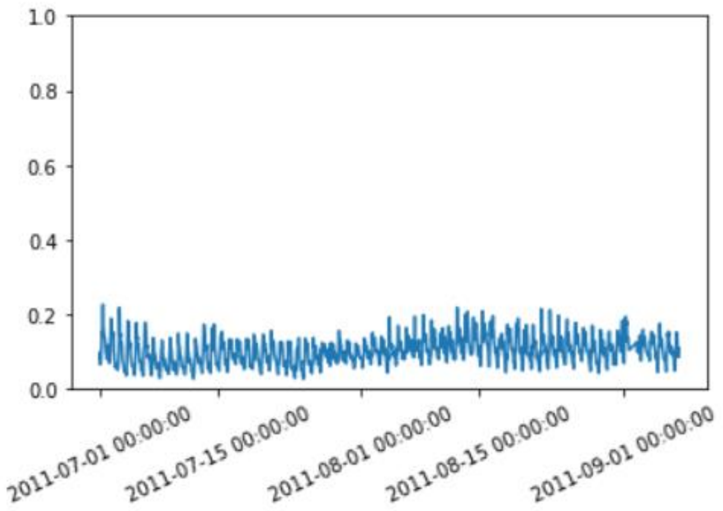
CPC dataset without anomalies:



CPC dataset with anomalies:



Plotting CPC without anomalies in the scale of above with anomalies dataset:



We can clearly see the anomalies by comparing the above two plots.

Statistics about the time points in the data:

```
Number of time points in the data:  1624
Number of days in the data 69
We can see below that there are only 15 records instead of 24 on the last da
1606    2011-09-06 22:00:01
1607    2011-09-06 23:00:01
1608    2011-09-07 00:00:01
1609    2011-09-07 01:00:01
1610    2011-09-07 02:00:01
1611    2011-09-07 03:00:01
1612    2011-09-07 04:00:01
1613    2011-09-07 05:00:01
1614    2011-09-07 06:00:01
1615    2011-09-07 07:00:01
1616    2011-09-07 08:00:01
1617    2011-09-07 09:00:01
1618    2011-09-07 10:00:01
1619    2011-09-07 11:00:01
1620    2011-09-07 12:00:01
1621    2011-09-07 13:00:01
1622    2011-09-07 14:00:01
1623    2011-09-07 15:00:01
Name: timestamp, dtype: object
```

There are two parts in this project.

Part 1: MLP model is built for predicting a sequence of values. Different setups of the window size of the output sequence are explored.

Part 2: Building an Autoencoder model for predicting a sequence of values. Autoencoder setups like Dense, LSTM, Conv1D layers were explored and their performance is visualized.

Part 1: Anomaly Detection task with MLP is very much similar to Time Series Analysis. We take a window size and sequence size. We start creating training datasets which are formed in a similar way of sliding window.

For Example: We had a data like [1,2,3,4,5,6,7,8,9,10] and the window size is 5 and sequence size is 2.

Training data and its corresponding labels would be like this:

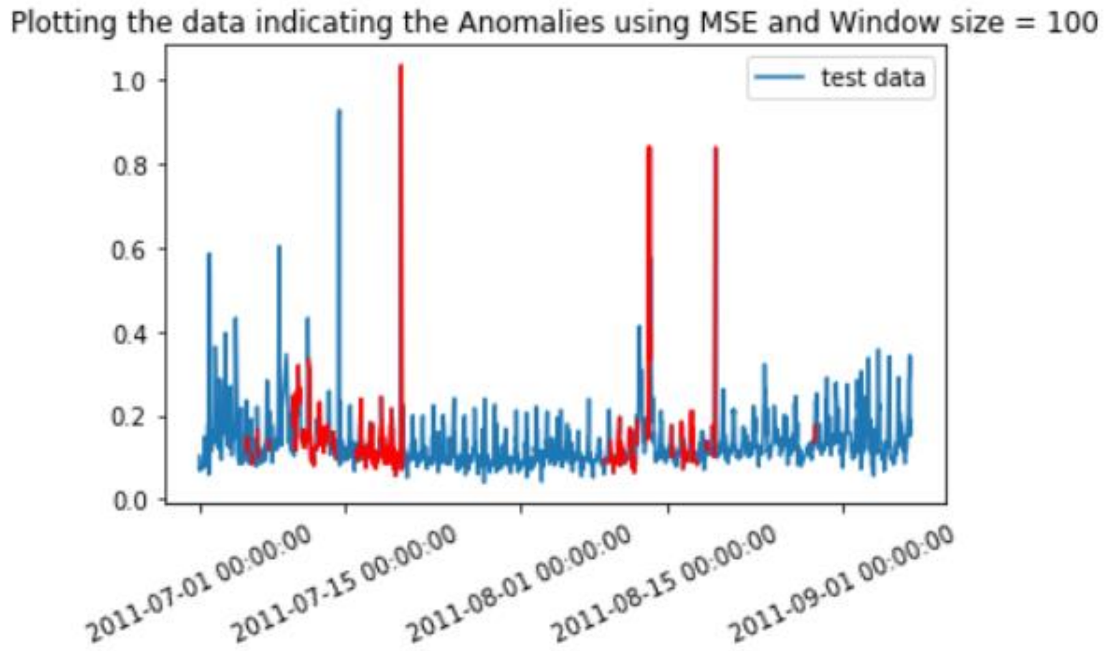
[1,2,3,4,5]	[6,7]
[2,3,4,5,6]	[7,8]
[3,4,5,6,7]	[8,9]
[4,5,6,7,8]	[9,10]

Its basically like forecasting the next 2 steps based on previous 5 steps if the defined window and sequence sizes are 5 and 2.

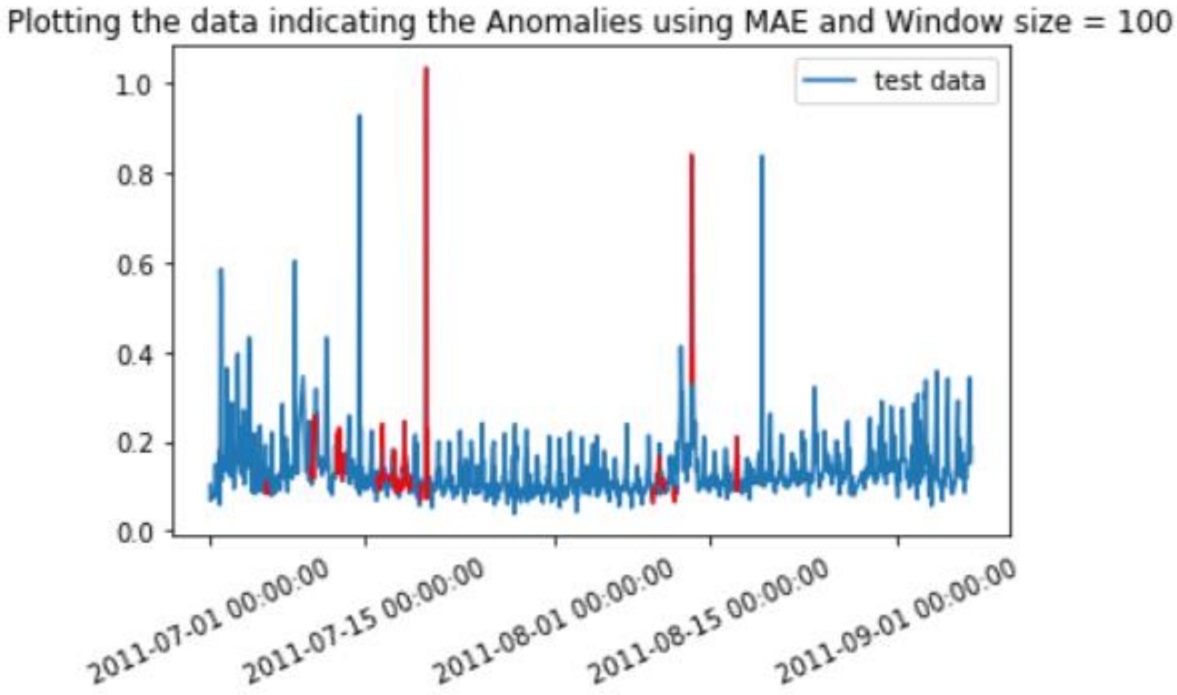
In our project, we design the training data and their labels based on the size defined in the above manner and the errors captured by our model and stored and a threshold is defined based on the nature if the anomaly. In this case, it is anomalies are upwards, the anomalous spikes are only found upwards through the comparisons done from visualized plots.

Error metrics used for this project are: Mean Squared Error, Mean Absolute Error and Cosine Distance.

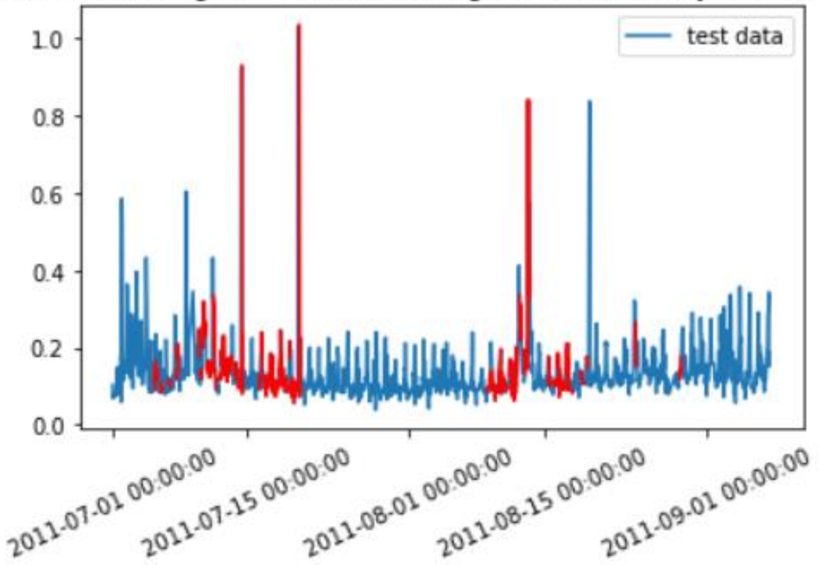
Results observed by applying the model trained with a defined threshold on the data with anomalies: The values for which the error captured is more than the defined threshold are marked in red color.



Applying Mean Absolute error instead of Mean Squared Error for the same model and window size:

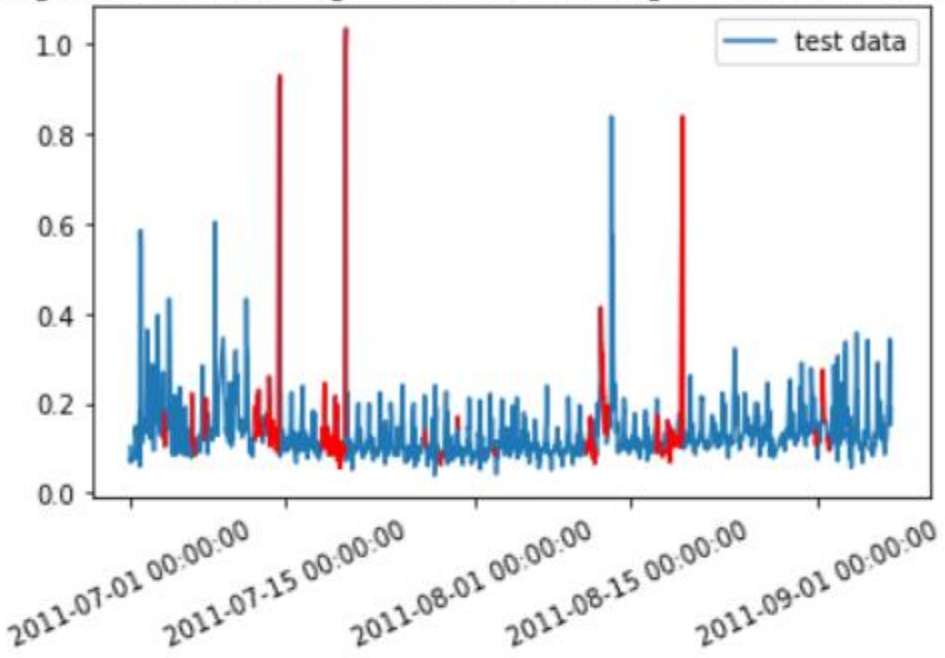


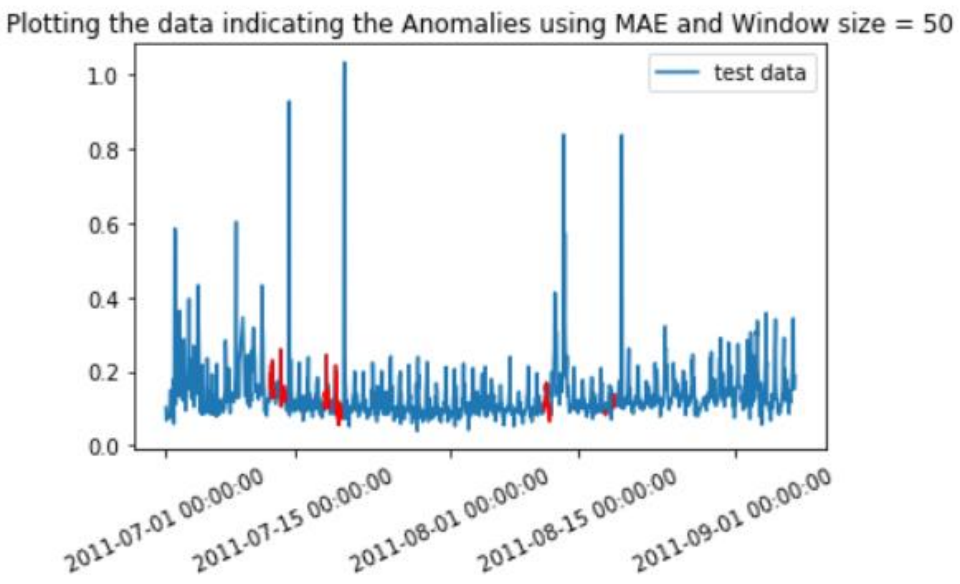
Plotting the data indicating the Anomalies using Cosine Similarity and Window size = 100

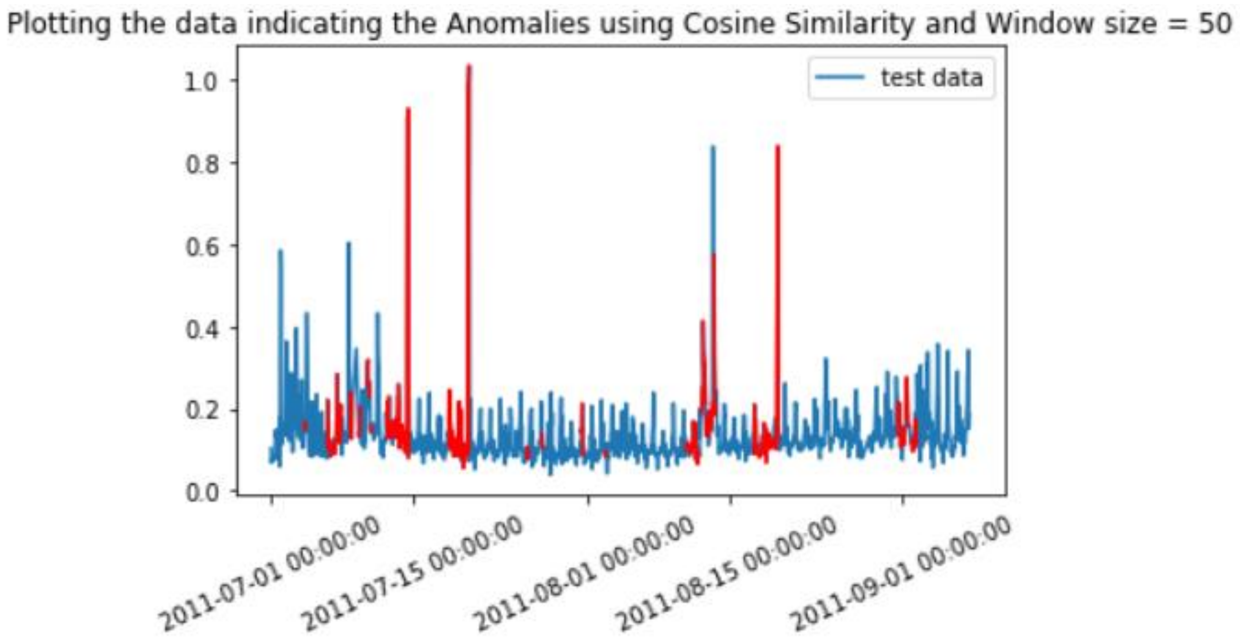


Creating the training data with the window size of 50. Below are the plots for Errors: Mean Squared Error, Mean Absolute Error and Cosine Distance:

Plotting the data indicating the Anomalies using MSE and Window size = 50

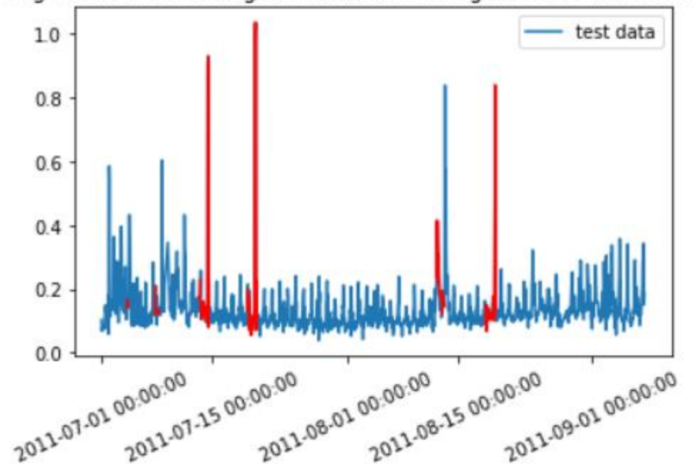




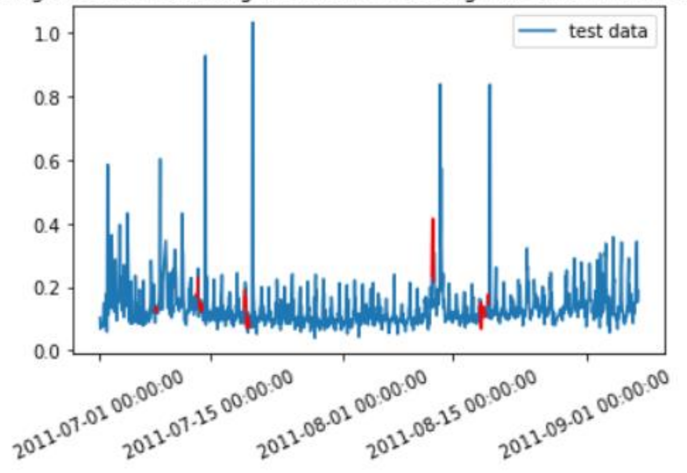


Creating the training data with the window size of 25. Below are the plots for Errors: Mean Squared Error, Mean Absolute Error and Cosine Distance:

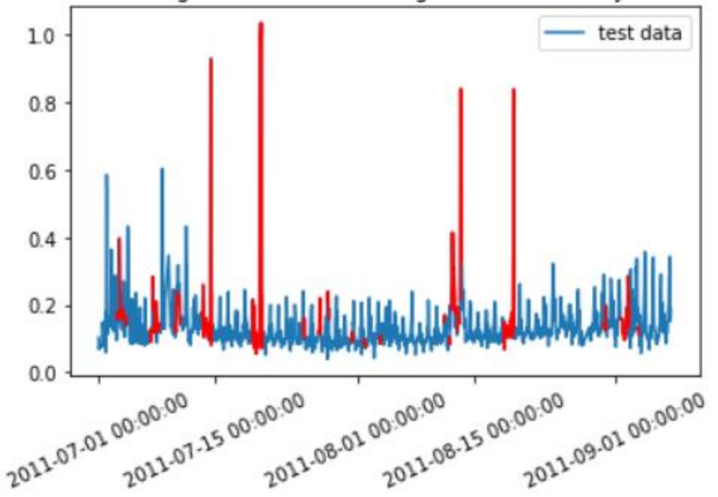
Plotting the data indicating the Anomalies using MSE and Window size = 25



Plotting the data indicating the Anomalies using MAE and Window size = 25



Plotting the data indicating the Anomalies using Cosine Similarity and Window size = 25



Observations:

In all the plots we can see some non-anomalous points to classified as anomalies, this might be due to the model trained on a similar data without anomalies but not exactly the same data, so the model might predict the values away from real trend at some points and hence their errors might be greater than the threshold, even though they are not anomalies. From the above results indicating number of anomalies detected for each window size given a distance measure type, that the model with 50 and 25 window size has least number of anomalies, while the model with 100 window size are higher.

In all the models we can see that with MSE or Cosine similarity as Measure, most of the anomalies are detected properly even though there are many False Positives (or Negatives this isn't a classification type), while MAE as measure it predicted only some of the anomalies but has lower False Positives compared to the other measures.

We can also see that the model with window size 25 has better anomalies identified compared to the higher window size

Note- in All the Models using different metrics the initial anomalies are not identified that is because those data points are not predicted because of the initial window size.

Part 2: Autoencoders were built with Dense, LSTM, Conv1D layers.

We basically decrease the dimension of the input data to extract latent features and later build the information to the size of original data from latent features.

The layers are built to in such a way the dimensions are reduced and later increased to input size. This is a basic structure of an Autoencoder.

Preparing training data for this part:

Data has been normalized. And this data has its value for every 5 minutes for 14 days.

24 timesteps per day. Data is from 2011-07-01 to 2011-09-07 = 69 days.

1624 data points in total.

Defining Time Steps: 24

Similar to the previous part, we need to create training data but here we create with respect to the time steps we defined.

We take the sequences and try to predict a sequence of input size. So, we are reconstructing the data from the original data. The concept behind this is, we try to recreate the same sequence and capture the errors occurred for every sequence and we set a threshold to detect anomalies.

For Example: We had a data like [1,2,3,4,5,6,7,8] with time steps = 5.

Training data would look like below.

[1,2,3,4,5]

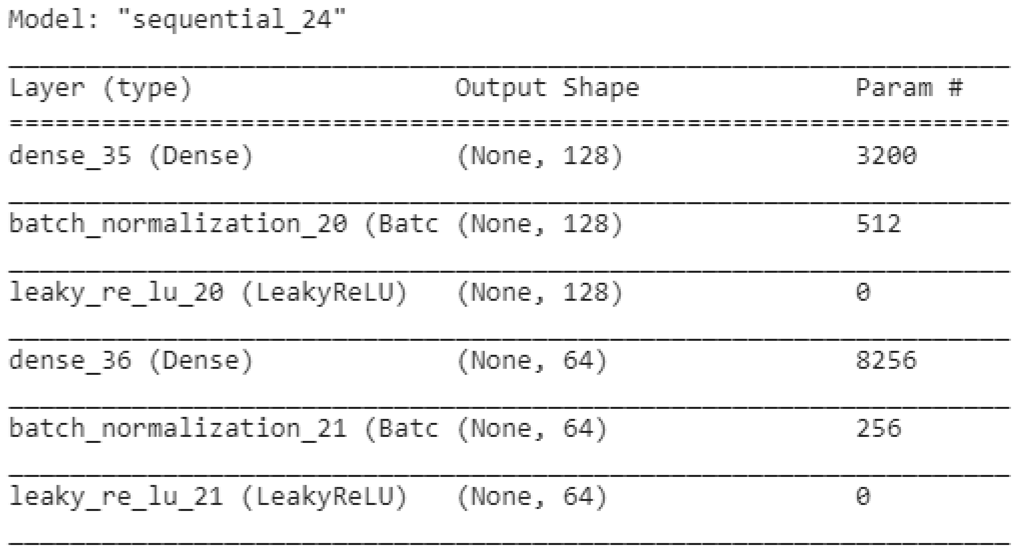
[2,3,4,5,6]

[3,4,5,6,7]

[4,5,6,7,8]

Building and training MLP with dense layers to form an autoencoders:

Model summary:

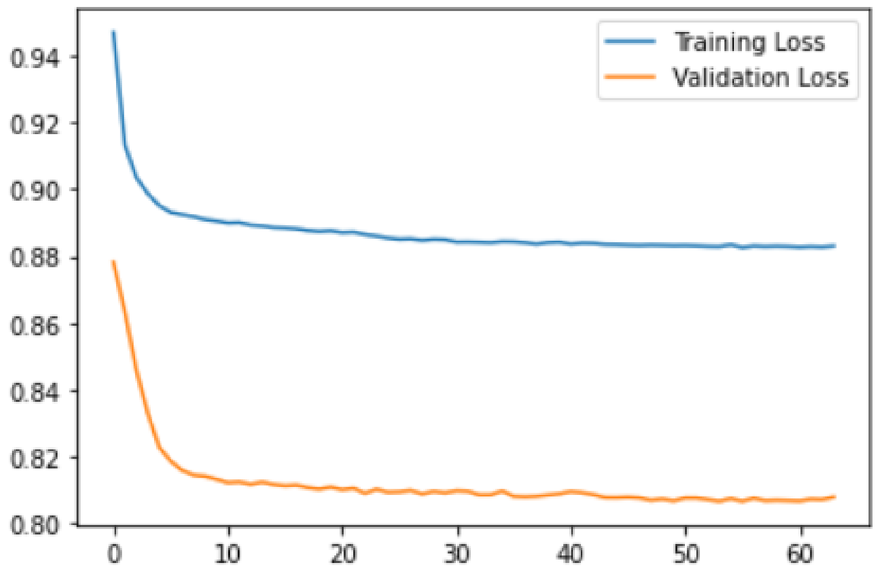


```
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 128)               3200
_________________________________________________________________
batch_normalization_20 (Batc (None, 128)               512
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 128)               0
_________________________________________________________________
dense_36 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_21 (Batc (None, 64)                256
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 64)                0
_________________________________________________________________
```

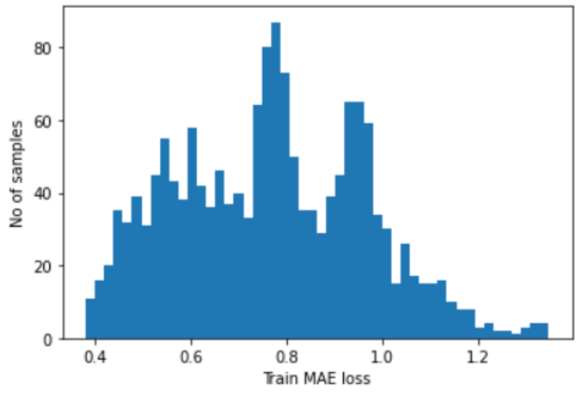
```
_________________________________________________________________
dense_37 (Dense)             (None, 16)                1040      
_________________________________________________________________
batch_normalization_22 (Batc (None, 16)                64        
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 16)                0         
_________________________________________________________________
dense_38 (Dense)             (None, 64)                1088      
_________________________________________________________________
batch_normalization_23 (Batc (None, 64)                256       
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 64)                0         
_________________________________________________________________
dense_39 (Dense)             (None, 128)               8320      
_________________________________________________________________
batch_normalization_24 (Batc (None, 128)               512       
_________________________________________________________________
leaky_re_lu_24 (LeakyReLU)   (None, 128)               0         
_________________________________________________________________
dense_40 (Dense)             (None, 24)                3096      
_________________________________________________________________
activation_10 (Activation)   (None, 24)                0         
=================================================================
Total params: 26,600
Trainable params: 25,800
Non-trainable params: 800
_________________________________________________________________
```

The above model was fit on the created training with batch size of 32 and 64 epochs.

Training Vs Validation Loss:

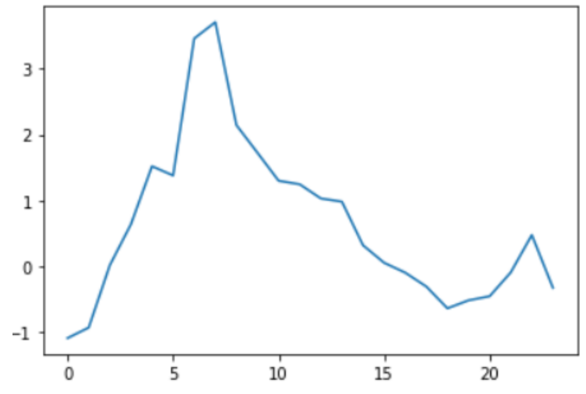
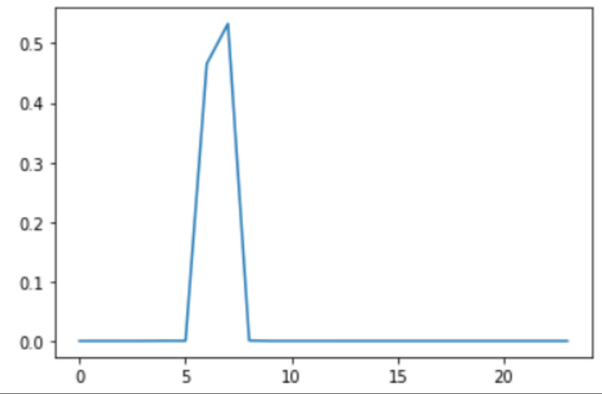


Defining threshold and plotting Absolute loss:

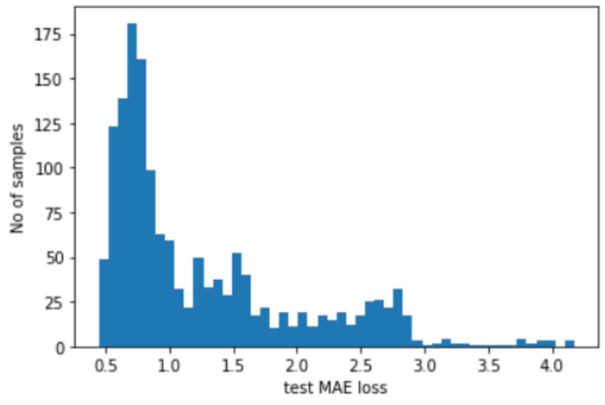


```
Reconstruction error threshold:  1.3493178102721888
```

Comparing the reconstructed data with original data:

Original data (second sequence)	Reconstructed data (second sequence)
	

Fitting the model on anomalous data and classifying the sequences with error beyond above defined threshold:



```
Number of anomaly samples:  494
```

Number of anomalous sequences are 494 but all of the timesteps involved cannot be considered as an anomaly as the errors can easily get skewed with one large anomalous spike.

Below is an example how anomalous time steps must be defined from these anomalous sequences:

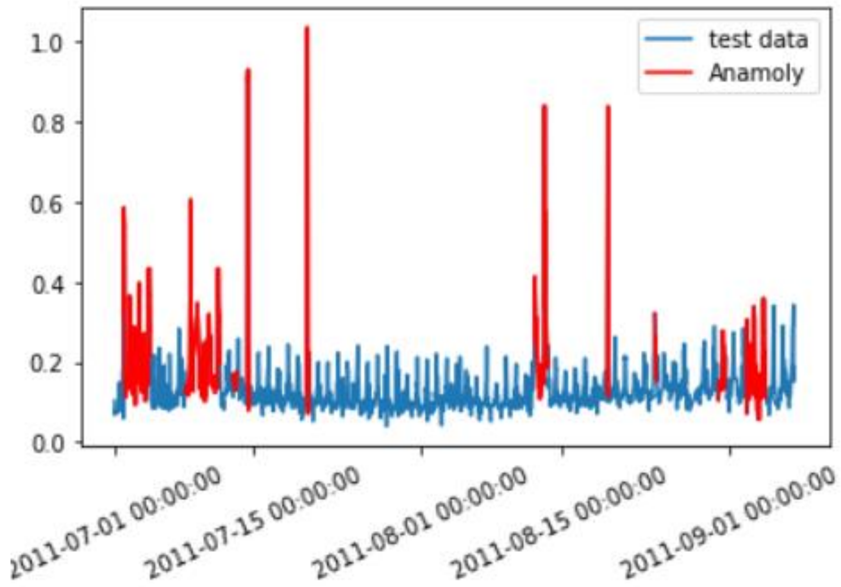
We now know the samples of the data which are anomalies. With this, we will find the corresponding 'timestamps' from the original test data. We will be using the following method to do that:

Let's say time steps = 3 and we have 10 training values. Our `x_train` will look like this:

- [0, 1, 2]	- [1, 2, 3]	- [2, 3, 4]	- [3, 4, 5]
- [4, 5, 6]	- [5, 6, 7]	- [6, 7, 8]	- [7, 8, 9]

All except the initial and the final time_steps-1 data values, will appear in 'time steps' number of samples. So, if we know that the samples [(3, 4, 5), (4, 5, 6), (5, 6, 7)] are anomalies, we can say that the data point 5 is an anomaly.

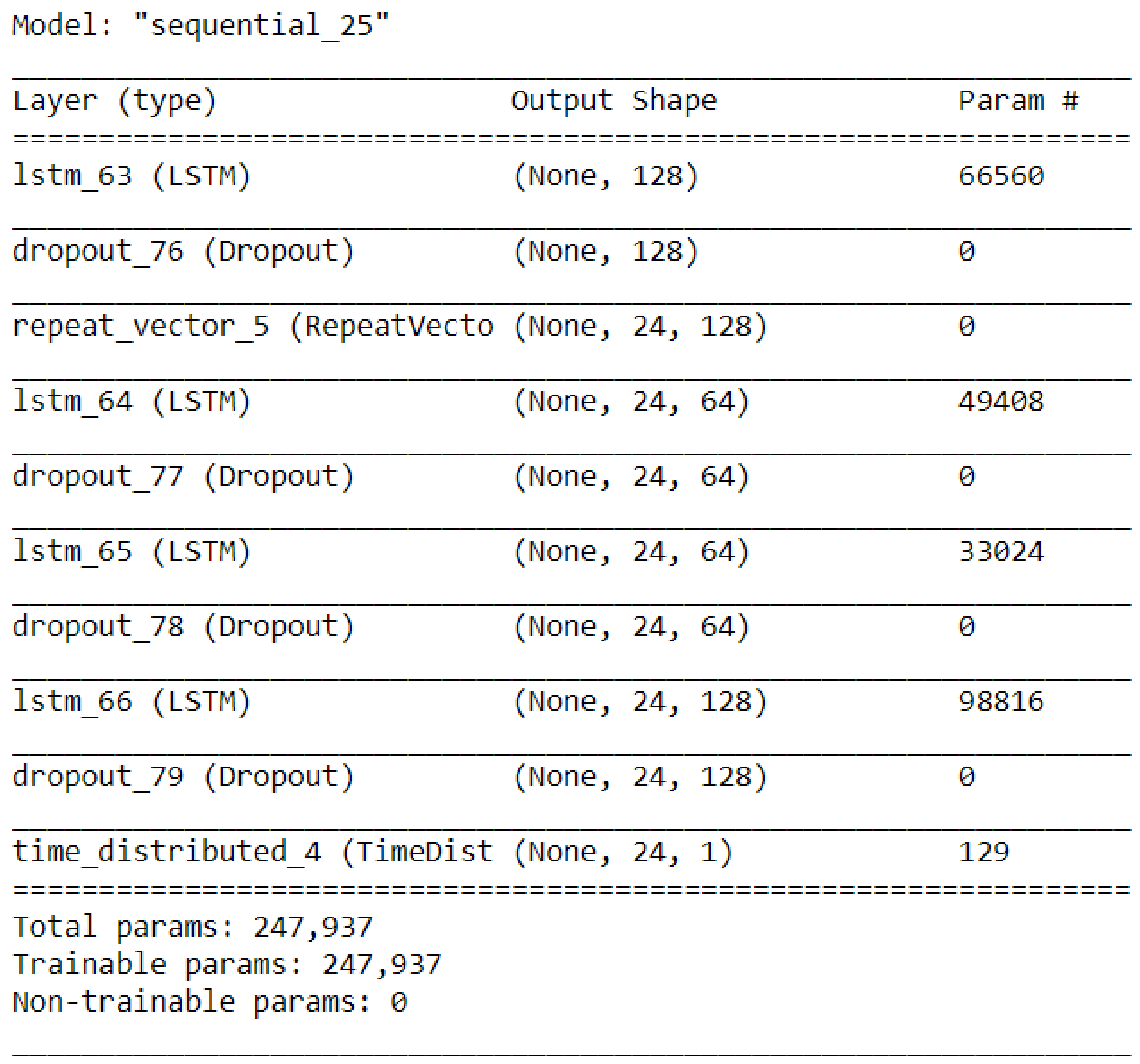
Applying the above logic to the 494 anomalous sequences detected. Below is the plot of detected anomalies in the anomalous data:



Observation: By looking at the reconstructed plot and the anomalies detected, we can clearly see that the autoencoder model created using MLP had not detected as much as anomalies as present and also the reconstructed plot of normal data was not up to the mark. But this model has detected anomalies comparatively better than the MLP model from part1.

Building an autoencoder using LSTM setup:

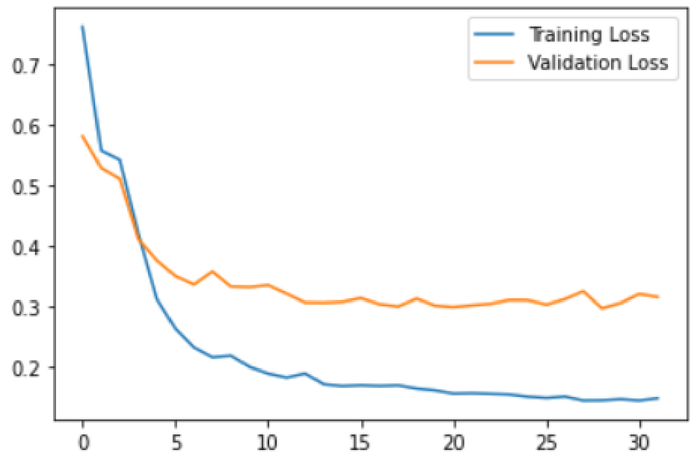
Model summary:



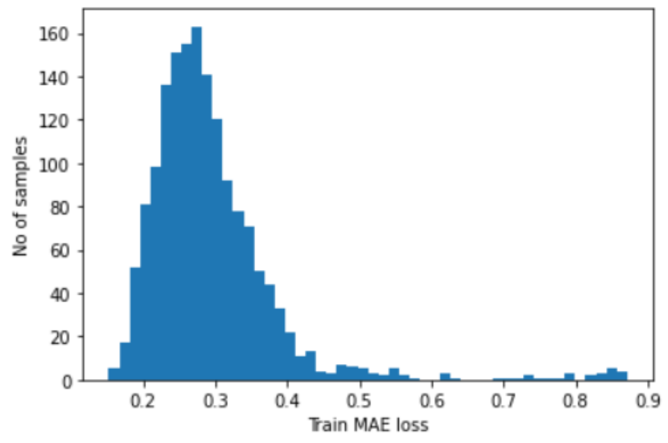
```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_63 (LSTM)               (None, 128)               66560     
_________________________________________________________________
dropout_76 (Dropout)         (None, 128)               0         
_________________________________________________________________
repeat_vector_5 (RepeatVecto (None, 24, 128)           0         
_________________________________________________________________
lstm_64 (LSTM)               (None, 24, 64)            49408     
_________________________________________________________________
dropout_77 (Dropout)         (None, 24, 64)            0         
_________________________________________________________________
lstm_65 (LSTM)               (None, 24, 64)            33024     
_________________________________________________________________
dropout_78 (Dropout)         (None, 24, 64)            0         
_________________________________________________________________
lstm_66 (LSTM)               (None, 24, 128)           98816     
_________________________________________________________________
dropout_79 (Dropout)         (None, 24, 128)           0         
_________________________________________________________________
time_distributed_4 (TimeDist (None, 24, 1)             129       
=================================================================
Total params: 247,937
Trainable params: 247,937
Non-trainable params: 0
_________________________________________________________________
```

The above model was fit on the created training with batch size of 32 and 32 epochs.

Training Vs Validation Loss:

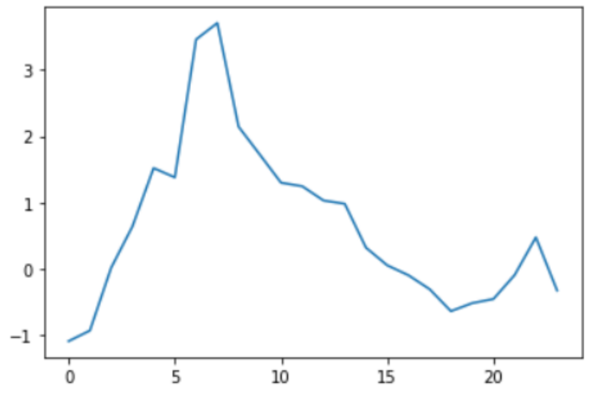
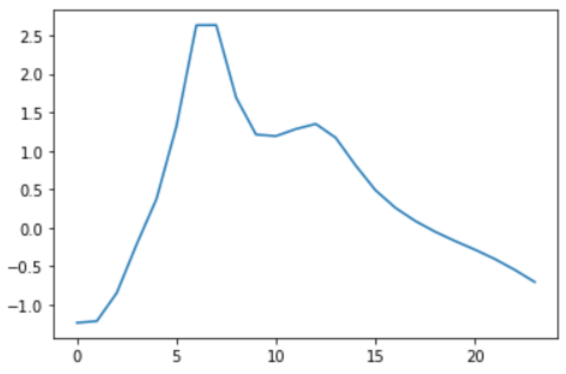


Defining threshold and plotting Absolute loss:

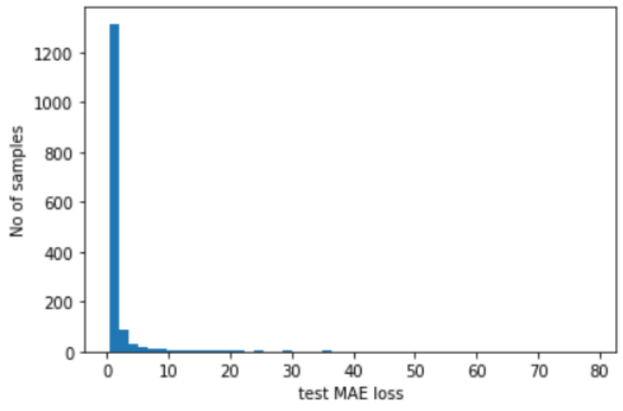


```
Reconstruction error threshold:  0.871270971026287
```

Comparing the reconstructed data with original data:

Original data (second sequence)	Reconstructed data (second sequence)
	

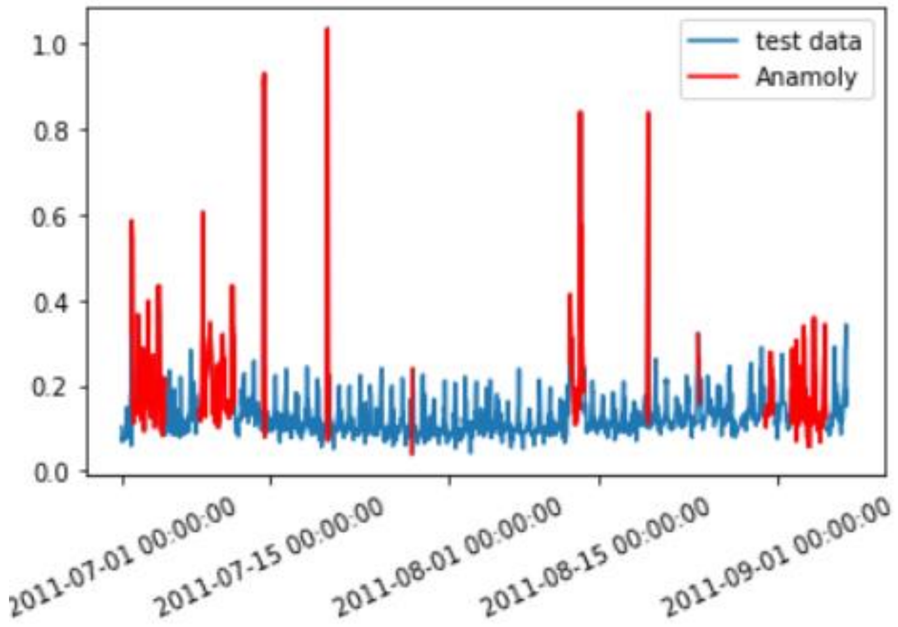
Fitting the model on anomalous data and classifying the sequences with error beyond above defined threshold:



```
Number of anomaly samples:  568
```

Number of anomalous sequences are 568 but all of the timesteps involved cannot be considered as an anomaly as the errors can easily get skewed with one large anomalous spike.

Applying the above discussed logic to the 568 anomalous sequences detected. Below is the plot of detected anomalies in the anomalous data:

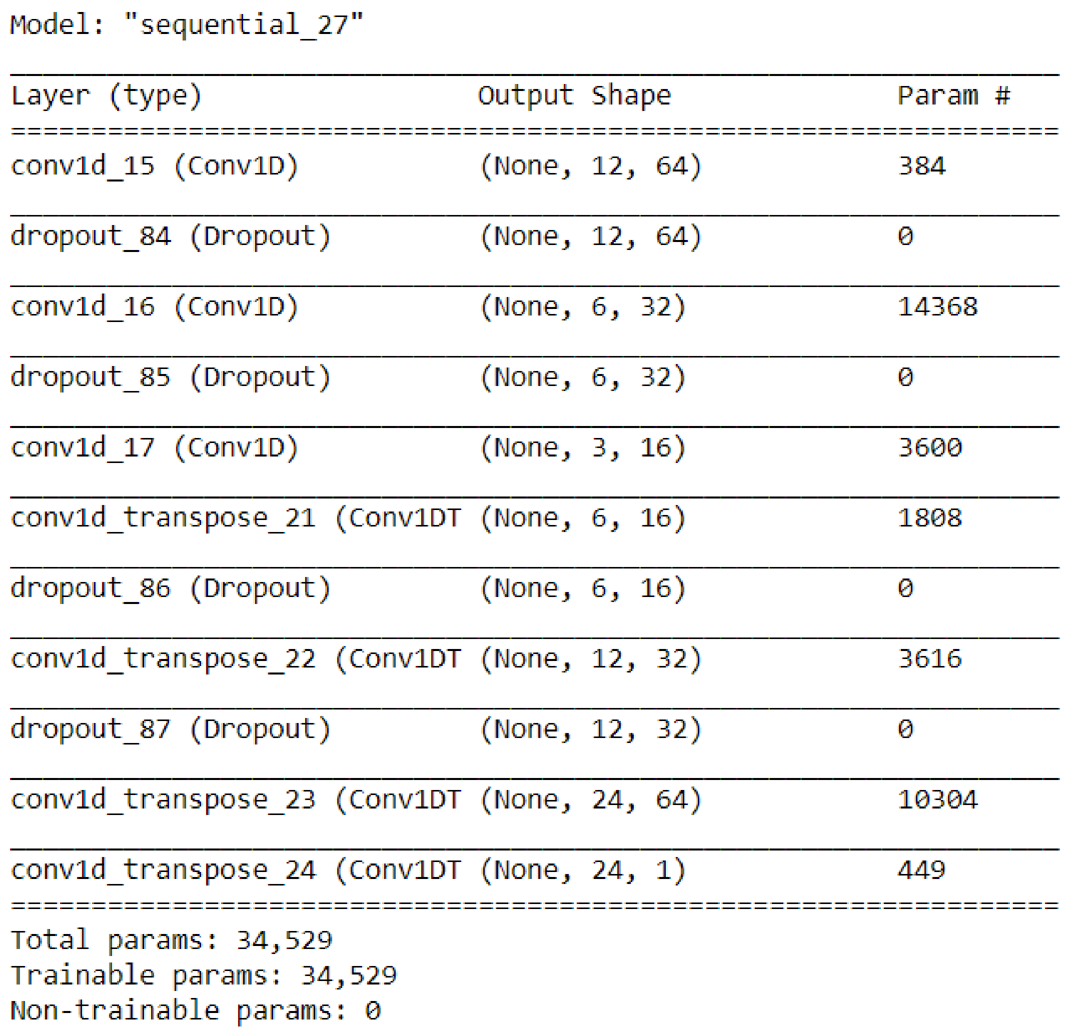


Observation: It is evident that LSTM autoencoder was better than autoencoder built using MLP in detecting anomalies. Validation loss was also less than the previous one.

We can also observe that this autoencoder built by LSTM had almost reconstructed the original plot formed by normal data.

Building an autoencoder using Conv1D layers:

Model summary:

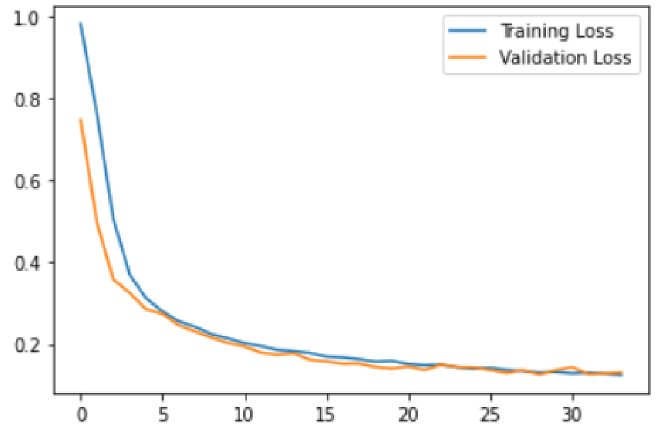


```
Model: "sequential_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_15 (Conv1D)           (None, 12, 64)            384       
_________________________________________________________________
dropout_84 (Dropout)         (None, 12, 64)            0         
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 6, 32)             14368     
_________________________________________________________________
dropout_85 (Dropout)         (None, 6, 32)             0         
_________________________________________________________________
conv1d_17 (Conv1D)           (None, 3, 16)             3600      
_________________________________________________________________
conv1d_transpose_21 (Conv1DT (None, 6, 16)             1808      
_________________________________________________________________
dropout_86 (Dropout)         (None, 6, 16)             0         
_________________________________________________________________
conv1d_transpose_22 (Conv1DT (None, 12, 32)            3616      
_________________________________________________________________
dropout_87 (Dropout)         (None, 12, 32)            0         
_________________________________________________________________
conv1d_transpose_23 (Conv1DT (None, 24, 64)            10304     
_________________________________________________________________
conv1d_transpose_24 (Conv1DT (None, 24, 1)             449       
=================================================================
Total params: 34,529
Trainable params: 34,529
Non-trainable params: 0
```

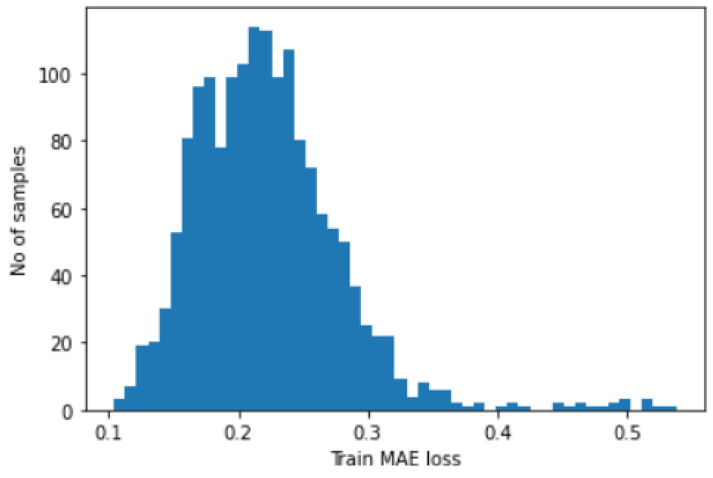
We can the dimensional change in the network all along.

The above model was fit on the created training with batch size of 128 and 50 epochs.

Training Vs Validation Loss:

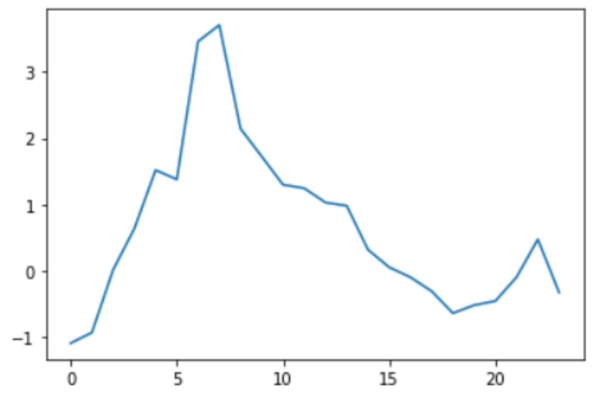
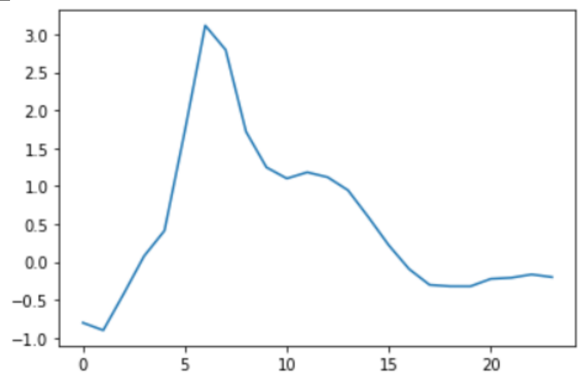


Defining threshold and plotting Absolute loss:

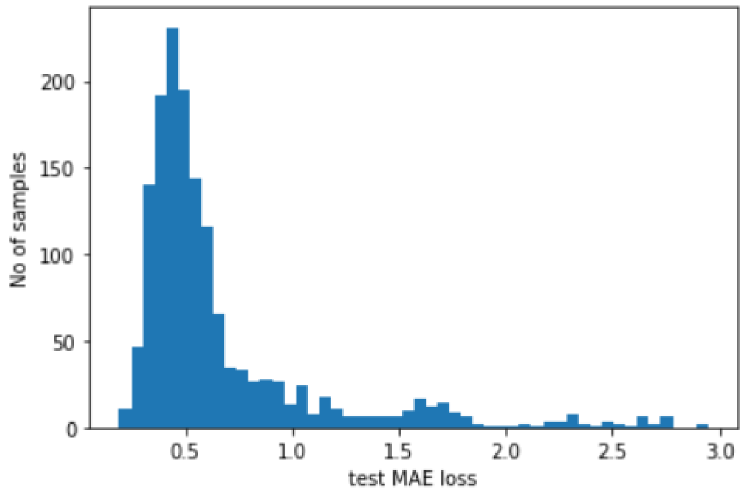


```
Reconstruction error threshold:  0.5377567854493958
```

Comparing the reconstructed data with original data:

Original data (second sequence)	Reconstructed data (second sequence)
	

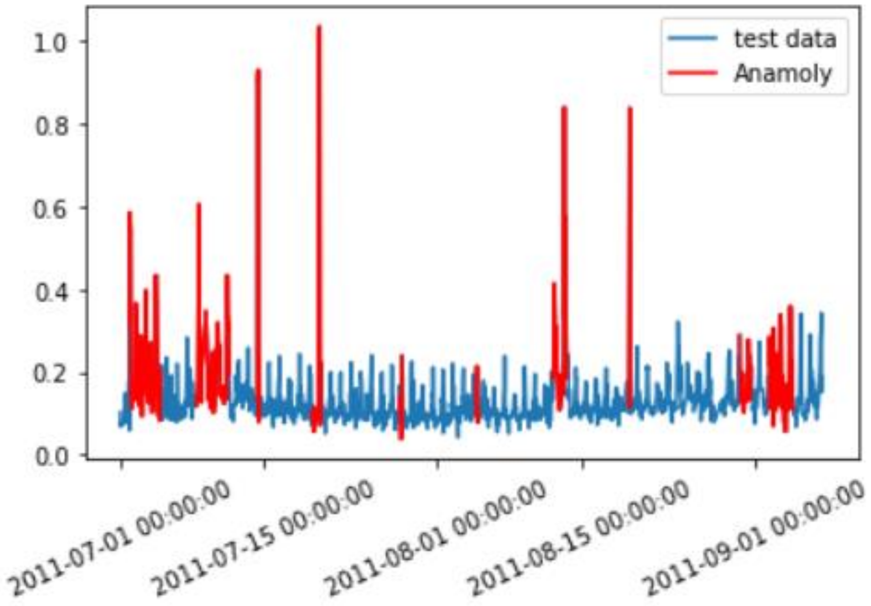
Fitting the model on anomalous data and classifying the sequences with error beyond above defined threshold:



```
Number of anomaly samples:  656
```

Number of anomalous sequences are 656 but all of the timesteps involved cannot be considered as an anomaly as the errors can easily get skewed with one large anomalous spike.

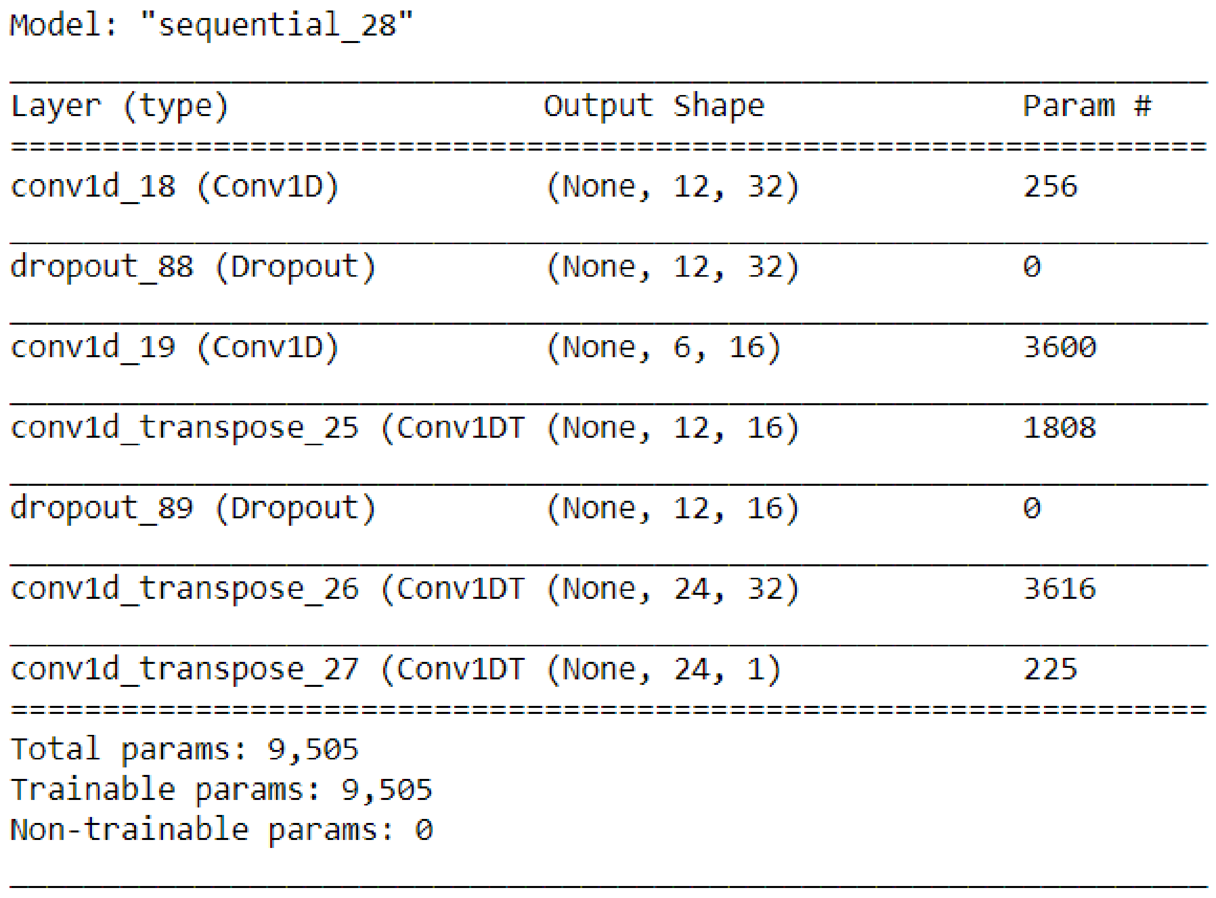
Applying the above discussed logic to the 568 anomalous sequences detected. Below is the plot of detected anomalies in the anomalous data:



Observation: We can observe that this autoencoder model built using convolutional layers had reconstructed the data comparatively better than LSTM and MLP autoencoders. We can also observe the decrease in validation loss comparatively.

Building an autoencoder using Conv1D layers (Number of layers have been decreased to limit the case of overfitting and kernel size has been increased from 5 to 7):

Model summary:

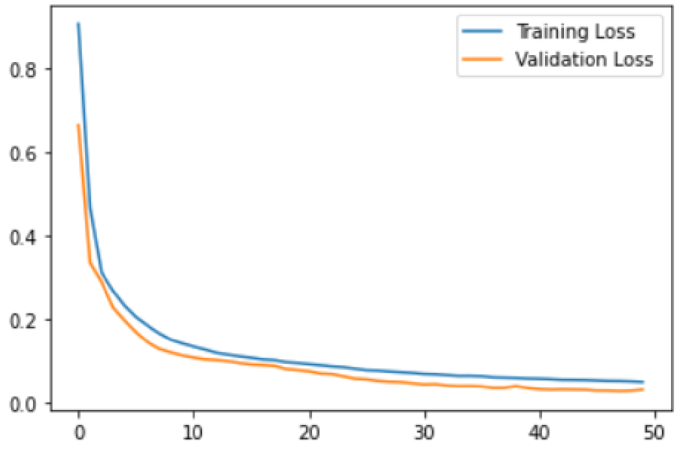


```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_18 (Conv1D)           (None, 12, 32)            256
_________________________________________________________________
dropout_88 (Dropout)         (None, 12, 32)            0
_________________________________________________________________
conv1d_19 (Conv1D)           (None, 6, 16)             3600
_________________________________________________________________
conv1d_transpose_25 (Conv1DT (None, 12, 16)            1808
_________________________________________________________________
dropout_89 (Dropout)         (None, 12, 16)            0
_________________________________________________________________
conv1d_transpose_26 (Conv1DT (None, 24, 32)            3616
_________________________________________________________________
conv1d_transpose_27 (Conv1DT (None, 24, 1)             225
=================================================================
Total params: 9,505
Trainable params: 9,505
Non-trainable params: 0
_________________________________________________________________
```

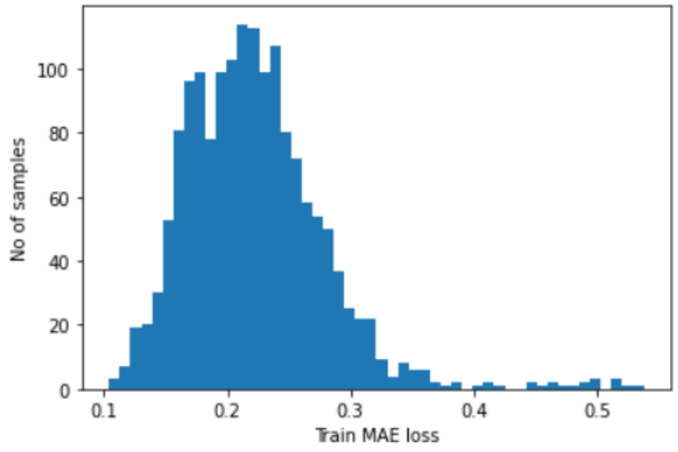
We can the dimensional change in the network all along. Structure of the autoencoder is clearly visible above.

The above model was fit on the created training with batch size of 128 and 50 epochs.

Training Vs Validation Loss:

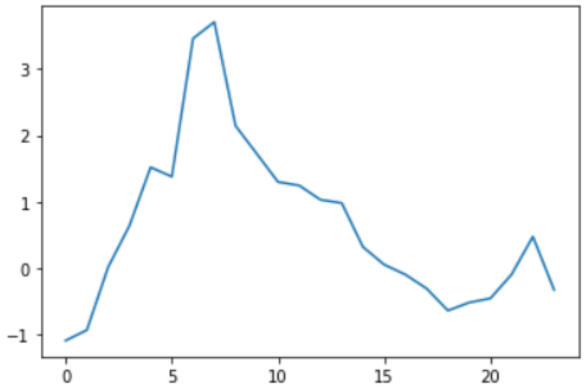
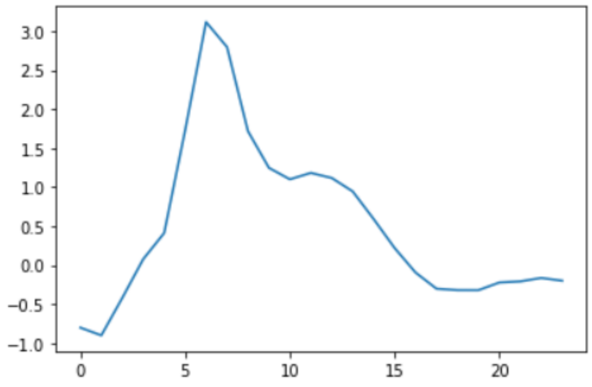


Defining threshold and plotting Absolute loss:

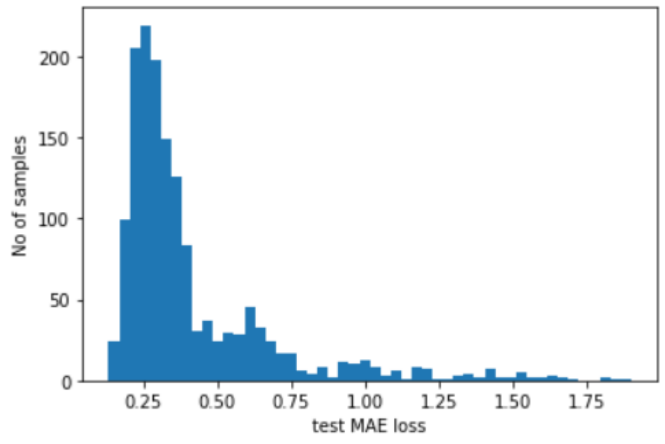


```
Reconstruction error threshold:  0.5377567854493958
```

Comparing the reconstructed data with original data:

Original data (second sequence)	Reconstructed data (second sequence)
	

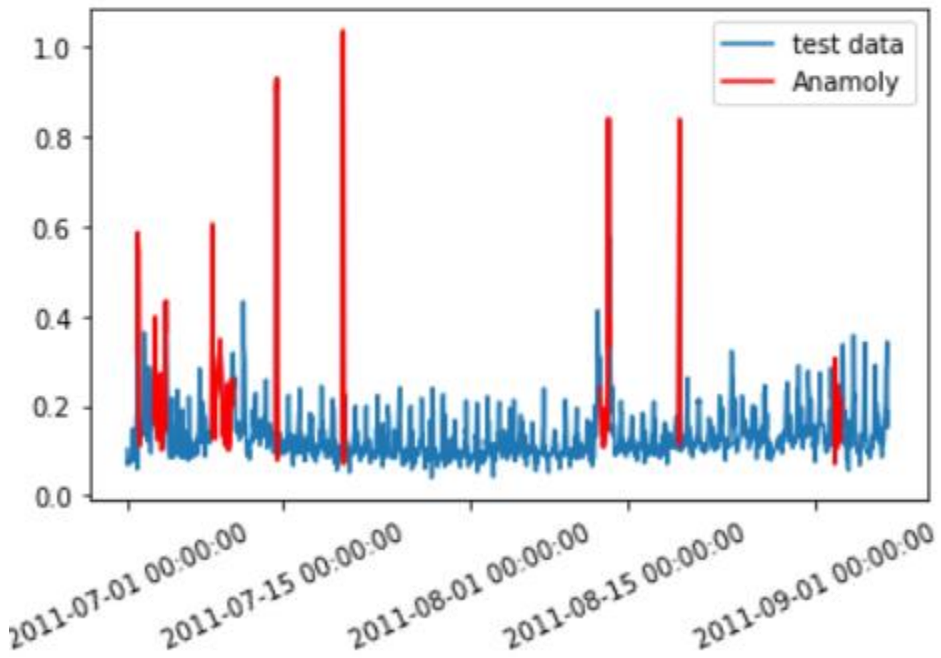
Fitting the model on anomalous data and classifying the sequences with error beyond above defined threshold:



```
Number of anomaly samples:  307
```

Number of anomalous sequences are 568 but all of the timesteps involved cannot be considered as an anomaly as the errors can easily get skewed with one large anomalous spike.

Applying the above discussed logic to the 568 anomalous sequences detected. Below is the plot of detected anomalies in the anomalous data:

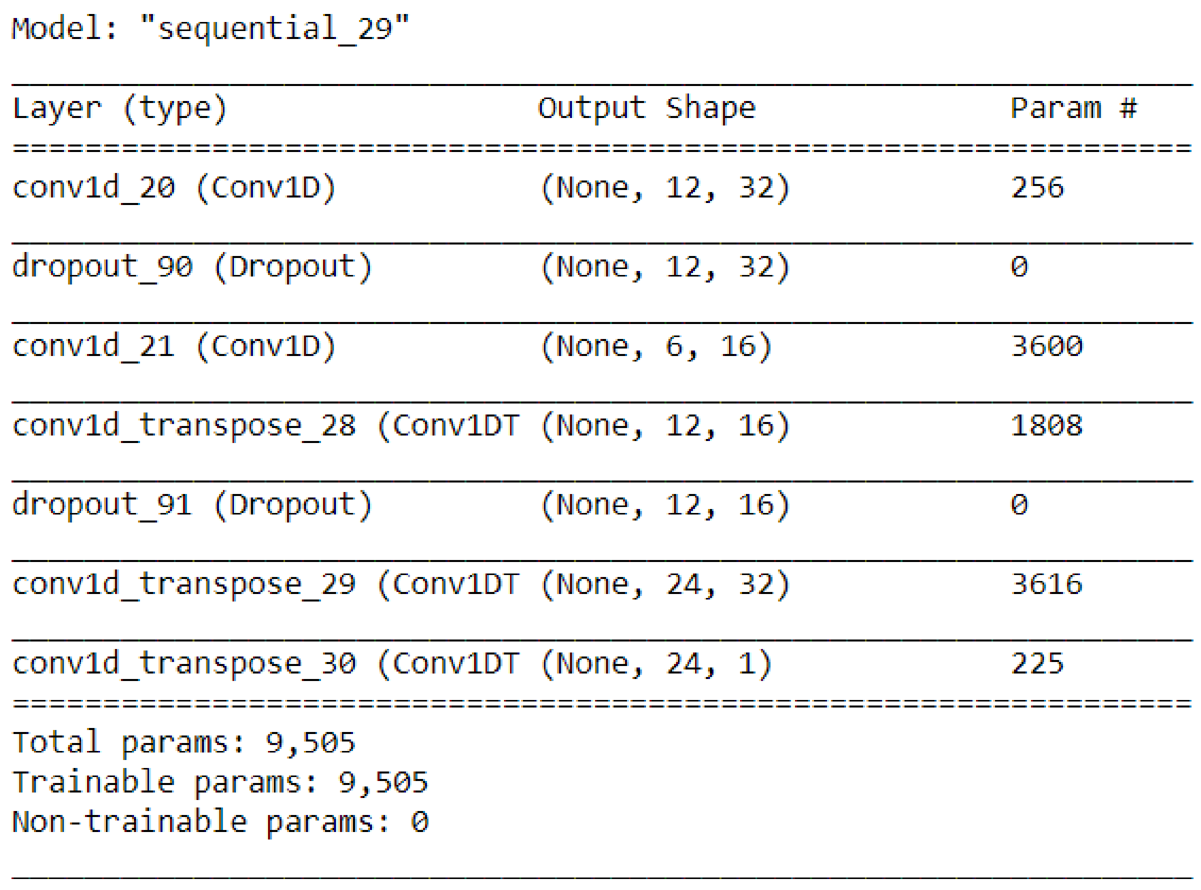


Observation: We can observe that the model had reconstructed normal data better than the previous convolutional autoencoder. And even validation loss was comparatively lesser than previous one.

Building another Autoencoder using Conv1D layers by defining the threshold using Mean Squared Error and changing the optimizer to RMSprop:

As this time series data has a little dependency on the previous time stamp values, RMSprop may be a good choice to better the performance.

Model summary:

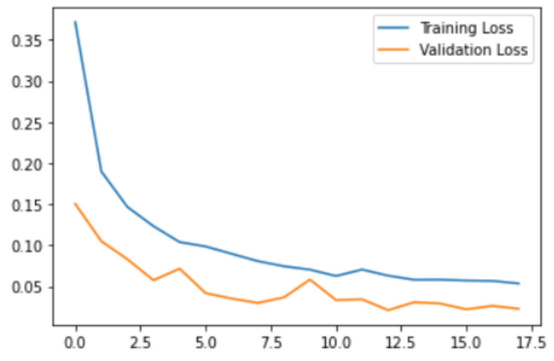


```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_20 (Conv1D)           (None, 12, 32)            256       
_________________________________________________________________
dropout_90 (Dropout)         (None, 12, 32)            0         
_________________________________________________________________
conv1d_21 (Conv1D)           (None, 6, 16)             3600      
_________________________________________________________________
conv1d_transpose_28 (Conv1DT (None, 12, 16)            1808      
_________________________________________________________________
dropout_91 (Dropout)         (None, 12, 16)            0         
_________________________________________________________________
conv1d_transpose_29 (Conv1DT (None, 24, 32)            3616      
_________________________________________________________________
conv1d_transpose_30 (Conv1DT (None, 24, 1)             225       
=================================================================
Total params: 9,505
Trainable params: 9,505
Non-trainable params: 0
_________________________________________________________________
```

We can the dimensional change in the network all along. This model structure is very much similar to the previous one.

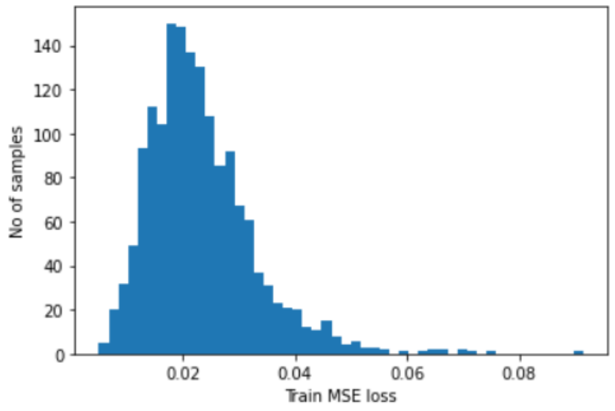
The above model was fit on the created training with batch size of 128 and 50 epochs.

Training Vs Validation Loss:



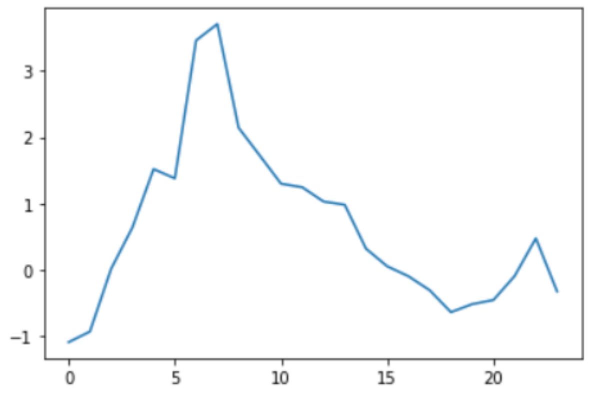
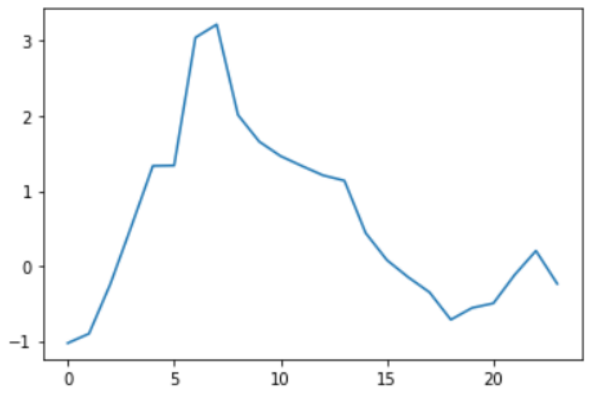
There may be few patterns in the training data which allowed the model to study the pattern in validation data leading to lesser validation loss than training loss.

Defining threshold and plotting Absolute loss:



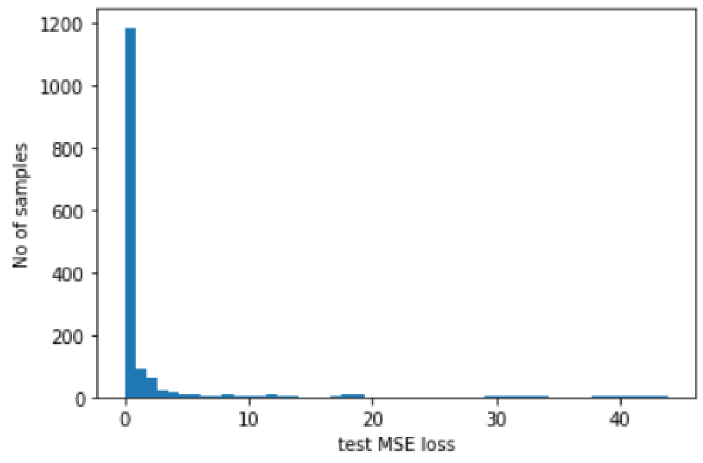
```
Reconstruction error threshold:  0.09136741665224539
```

Comparing the reconstructed data with original data:

Original data (second sequence)	Reconstructed data (second sequence)
	

We can see that model has almost perfectly reconstructed the training data

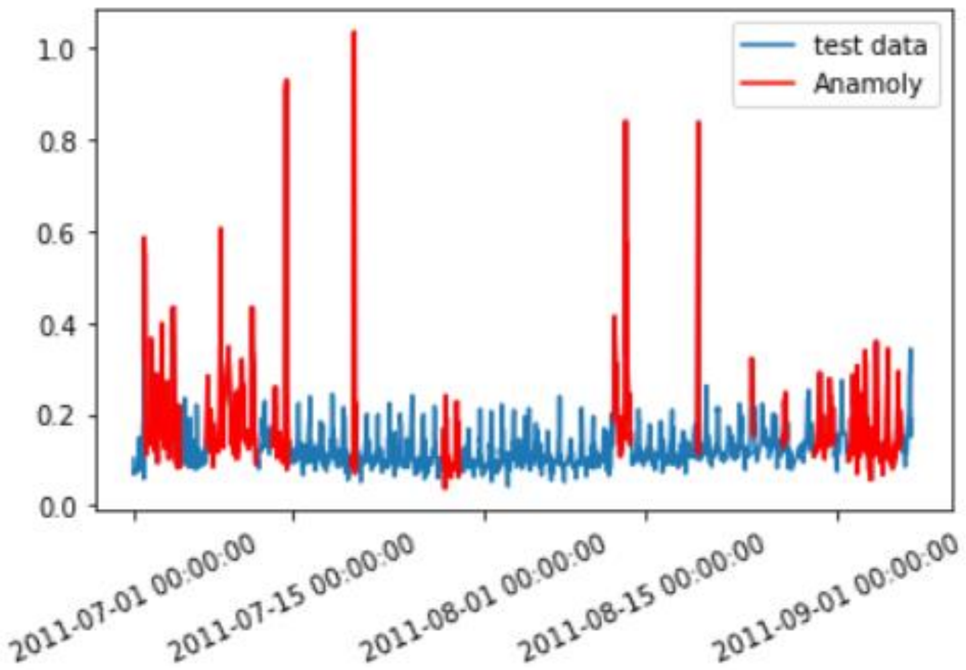
Fitting the model on anomalous data and classifying the sequences with error beyond above defined threshold:



```
Number of anomaly samples:  833
```

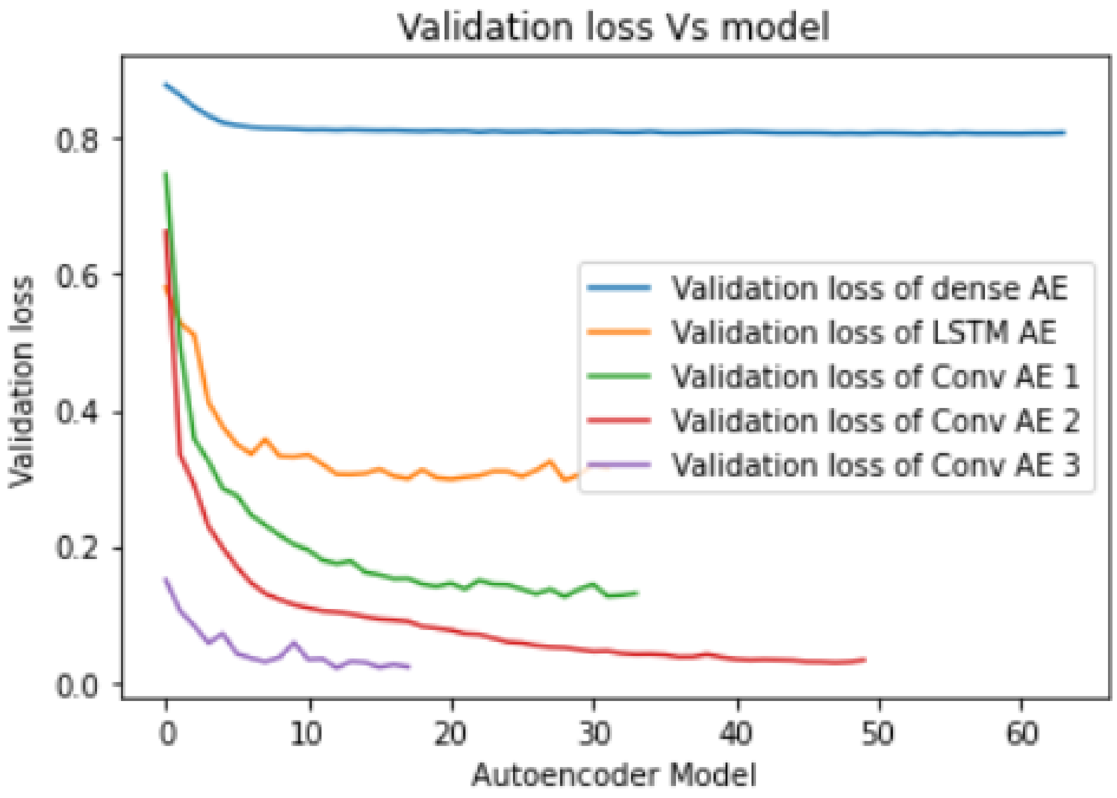
Number of anomalous sequences are 568 but all of the timesteps involved cannot be considered as an anomaly as the errors can easily get skewed with one large anomalous spike.

Applying the above discussed logic to the 568 anomalous sequences detected. Below is the plot of detected anomalies in the anomalous data:



Observation: We can see that change in threshold and optimizer had improved the performance of the autoencoder. This can be observed by looking at the reduction of validation loss and improvement of reconstructed data.

Comparing the validation loss for each model:

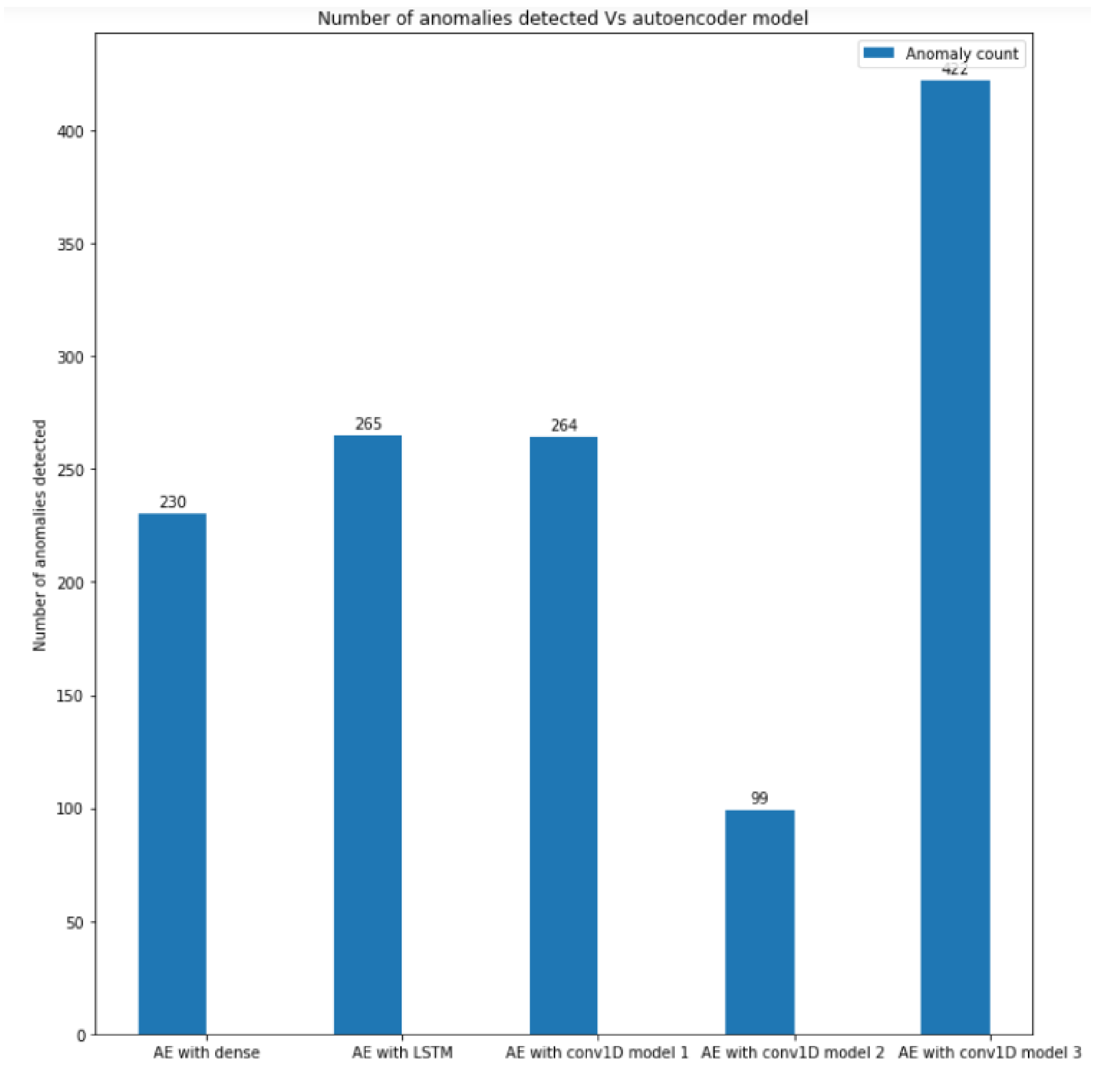


Observation:

We can observe that Autoencoder built by convolutional layers performed better than Autoencoders built by LSTM and MLP.

Comparing the autoencoders built by convolutional layers, we can see that autoencoders which used mean squared loss for defining threshold and RMSprop for optimizer had performed comparatively better.

Plotting the number of anomalies detected by each autoencoder model:



AE with dense: Autoencoder built using Dense layers

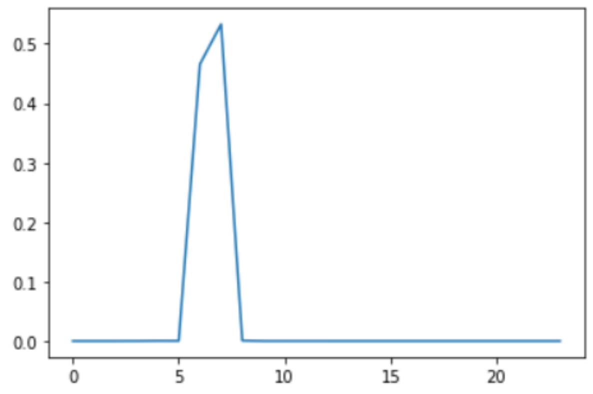
AE with LSTM: Autoencoder built using LSTM layers

AE with Conv1D model 1: Autoencoder built using Convolutional layers

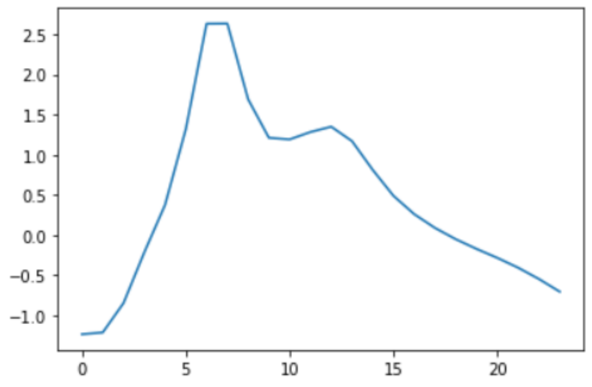
AE with Conv1D model 2: Autoencoder built using Convolutional layers (Number of layers were reduced from previous model)

AE with Conv1D model 3: Autoencoder built using Convolutional layers (By changing the error metric (to MSE) used to determine threshold and the optimizer (to RMSprop)

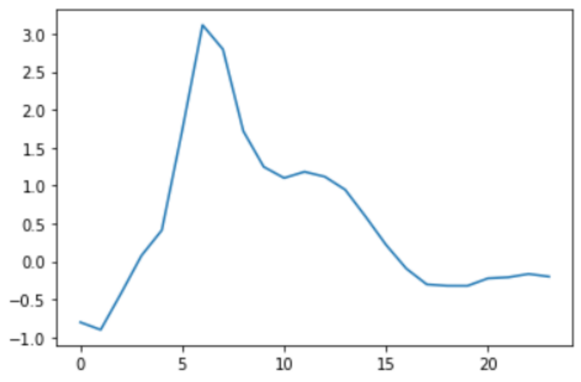
Comparing the reconstructed data of all models:



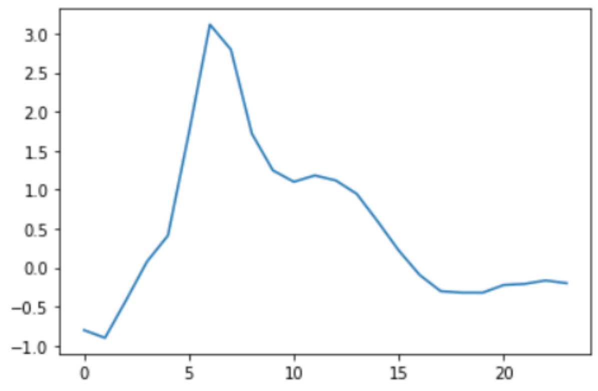
Autoencoder built using Dense layers



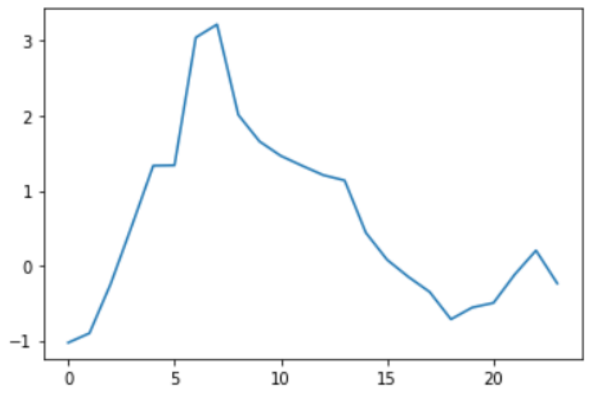
Autoencoder built using LSTM layers



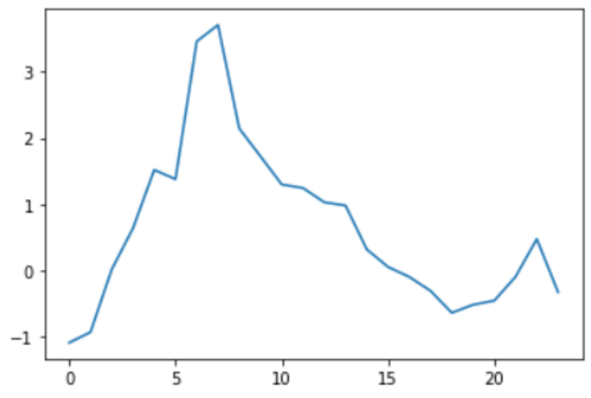
AE with Conv1D model 1



AE with Conv1D model 2



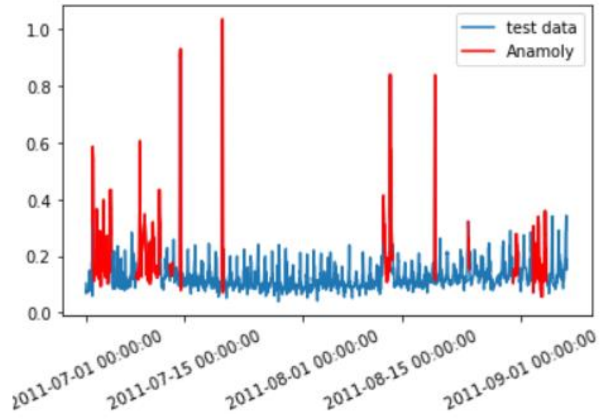
AE with Conv1D model 3



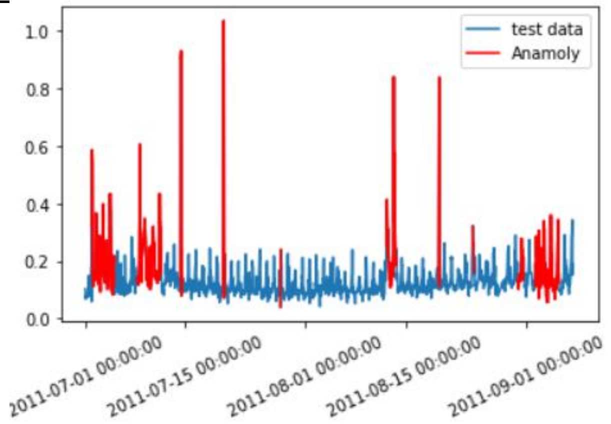
Original data

It is apparent that the Autoencoder built using Convolutional layers (By changing the error metric (to MSE) used to determine threshold and the optimizer (to RMSprop) reconstructed the data almost as perfectly as the original data.

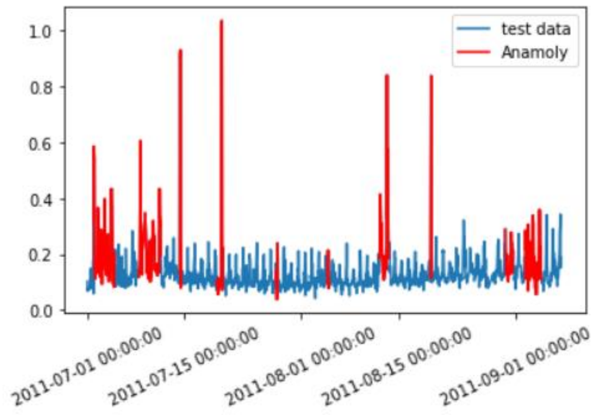
Comparing Anomaly Detection on Anomalous data for all models:



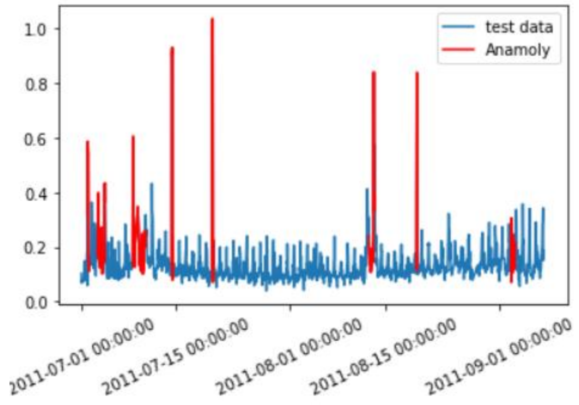
Autoencoder built using Dense layers



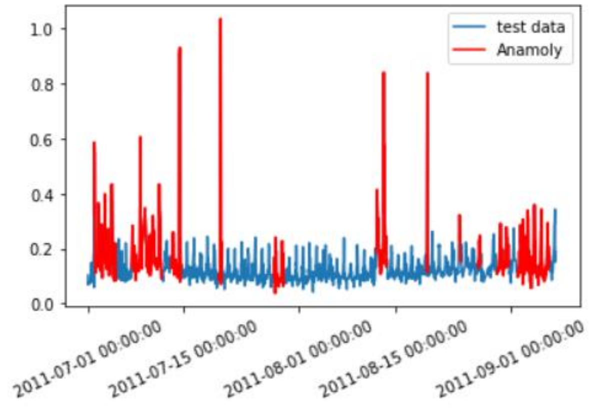
Autoencoder built using LSTM layers



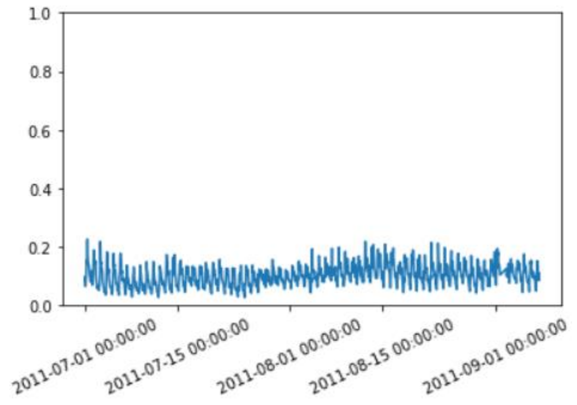
AE with Conv1D model 1



AE with Conv1D model 2



AE with Conv1D model 3



Original data without anomalies

We can see that fine-tuned Autoencoder built with Conv1D layers performed really well and had detected almost all anomalies. This can be clearly seen above by comparing it with non-anomalous data.