Online Anomaly Detection on Streaming Data

Zayed Uddin Chowdhury

Abstract

This paper focuses on anomaly detection problem for dynamically changing data streams. It discusses old algorithms and contrast it with newly developed algorithms. How they can be used or adapted to detect anomalies within very short duration maintaining a fair accuracy.

Index Terms

Anomaly detection algorithm, outlier detection, long short term memory, median absolute deviation

I. INTRODUCTION

Anomaly detection is a very important task with many real life applications in various fields. Business, Health, Production, Communication and many other fields rely on good anomaly detection. Business Analytic, Fraud Detection, Forecasting, Device monitoring etc are some of the practical use cases where this is used.

What is an anomaly? In general, anything that is not normal or out of its regular path is an anomaly. However, detecting something as an anomaly is not simple. It depends on different parameters like the problem, domain, time range etc. Our scope of this paper is detecting anomaly in time series data. Most common approaches for this includes different statistical method or offline machine learning method which trains its model using past data and use this model to detect anomalies in new unseen time series data.

Now a days, streaming data is everywhere and growing at a rapid pace. Focus on industrial Internet of Things (IoT) [3] is one of the main reasons behind it. IoT may use multiple sensors in different locations, each one of them generating thousands of data each second. One of the major anticipation from this type of technology is to automate the monitoring and some cases reaching to conclusion such as giving alert in case of abnormal behaviour. For example, in large data centers time-taken to run a batch-job is recorded as a time series. The time series is observed to find out if it has any non-stationary behavior due to periodic (e.g. weekly, monthly, seasonal) changes in transaction volumes. It is desirable to detect the situations when the execution time of a job is abnormally high or low. Similar patterns are observed in power-plants due to seasonal variations in power demand, different operating modes (full-load vs partial-load), varying quality of inputs (such as coal quality in thermal power plants), etc. Automatically detecting anomalies such as increased temperature or vibration becomes challenging under such dynamic conditions.

Many researches have been done and still running on the field of anomaly detection. Old anomaly detection techniques profoundly relied on statistical algorithms. After the rapid improvement of Machine Learning, the detection techniques shifted towards machine learning algorithms. However, in both cases the detection of anomaly depends heavily on previous batch data. In other words, it analyzes/learns from a huge amount of past data to detect anomaly in the incoming data. But, with the increase of streaming devices like different IoT devices, real time services etc it has become more challenging. As, these kind of systems generate a lot of real time data within a very few time window. As a result, in contrast to batch processing, these type of applications demand fairly accurate anomaly detection within a very short time. Also most of these are done on some cloud server. Through this paper we will focus on some of the algorithms developed for detecting anomaly on streaming/time series data in an online fashion.

II. Literature Review

A. *Long Short Term Memory Networks for Anomaly Detection[15]*

1) Long Short Term Memory: A Long short-term memory (LSTM) is a type of Recurrent Neural Network specially designed to prevent the neural network output for a given input from either decaying or exploding as it cycles through the feedback loops. The feedback loops are what allow recurrent networks to be better at pattern recognition than other neural networks. Memory of past input is critical for solving sequence learning tasks and Long short-term memory networks provide better performance compared to other RNN architectures by alleviating what is called the vanishing gradient problem.[11]

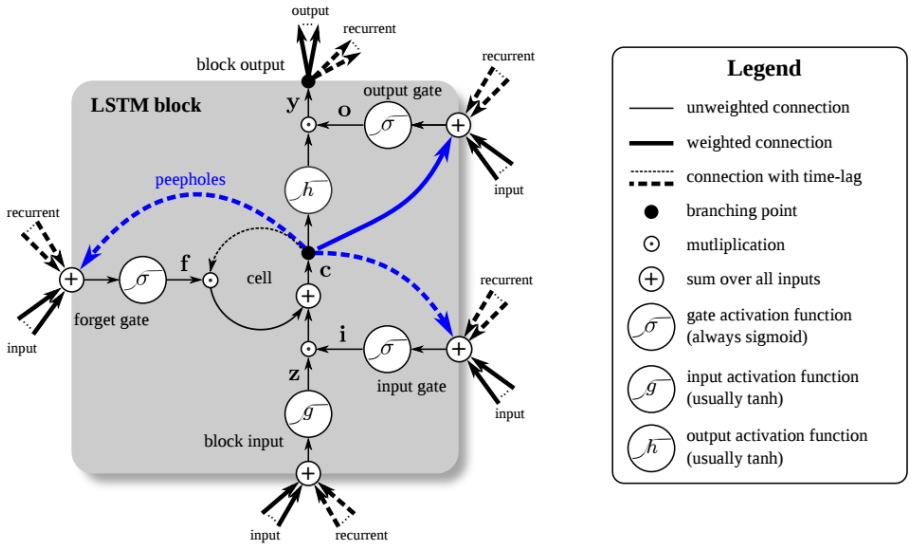


Fig. 1: LSTM Architecture [11]

The Long Short-Term Memory Architecture (Figure: 1) consists of linear units with a self-connection having a constant weight of 1.0. This allows a value (forward pass) or gradient (backward pass) that flows into this self-recurrent unit to be preserved and subsequently retrieved at the required time step. With the unit multiplier, the output or error of the previous time step is the same as the output for the next time step. This self-recurrent unit, the memory cell, is capable of storing information which lies dozen of time-steps in the past. This is very powerful for many tasks. For example for text data, an LSTM unit can store information contained in the previous paragraph and apply this information to a sentence in the current paragraph. Bidirectional LSTMs train the input sequence on two LSTMs - one on the regular input sequence and the other on the reversed input sequence. This can improve LSTM network performance by allowing future data to provide context for past data in a time series. These LSTM networks can better address complex sequence learning/machine learning problems than simple feed-forward networks.

"Long Short Term Memory (LSTM) networks have been demonstrated to be particularly useful for learning sequences containing longer term patterns of unknown length, due to their ability to maintain long term memory" [15]. Stacking recurrent hidden layers in such networks also enables the learning of higher level temporal features, for faster learning with sparser representations. In this paper, we use stacked LSTM networks for anomaly/fault detection in time series. A network is trained on non-anomalous data and used as a predictor over a number of time steps. The resulting prediction errors are modeled as a multivariate Gaussian distribution, which is used to assess the likelihood of anomalous behavior.

Consider a time series $X = \{x^{(1)}, x(2), ..., x(n)\}$, where each point $x^{(t)} \in R^m$ in the time series is an m-dimensional vector $\{x_1^{(t)}, x_2^{(t)}, ..., x_m^{(t)}\}$, whose elements correspond to the input variables. A

prediction model learns to predict the next l values for d of the input variables s.t. $1 \leq d \leq m$. The normal sequence(s) are divided into four sets: normal train (S_N), normal validation-1 (V_{N1}), normal validation-2 (V_{N2}), and normal test (t_N). The anomalous sequence(s) are divided into two sets: anomalous validation (V_A), and anomalous test (T_A). We first learn a prediction model using stacked LSTM networks, and then compute the prediction error distribution using which we detect anomalies.

B. Online Anomaly Detection with Concept Drift Adaptation using RNN[15]

Most of the approaches for time series anomaly detection using RNNs (e.g. [8] , [4]) rely on offline models learned using historical data assuming that the training and testing data come from the same distribution. However, in practice, most time series data is non-stationary in nature. In this method, one of the popular offline approaches for anomaly detection using RNNs in [5] is extended and an Online RNN-AD is proposed, where the RNN is trained incrementally as new data arrives for time series anomaly detection under concept drift. RNN is trained to predict values for multiple time steps using historical readings. The prediction error is used to detect anomalies and change points as well as appropriately adapt to changes.

We assume that a model that is able to predict the next few steps in a time series based on history is a suitable model for the time series. In other words, if a model is able to predict the values at next few steps in a time series well, it has effectively learned to capture the important temporal characteristics in the time series. In case of stationary time series, once trained, such a model is expected to predict the time series depicting normal behavior well but perform badly on the prediction task on time series corresponding to anomalous behavior. This idea has been effectively applied to several domains where a temporal model for normal behavior is learned using multilayered LSTM based RNNs.

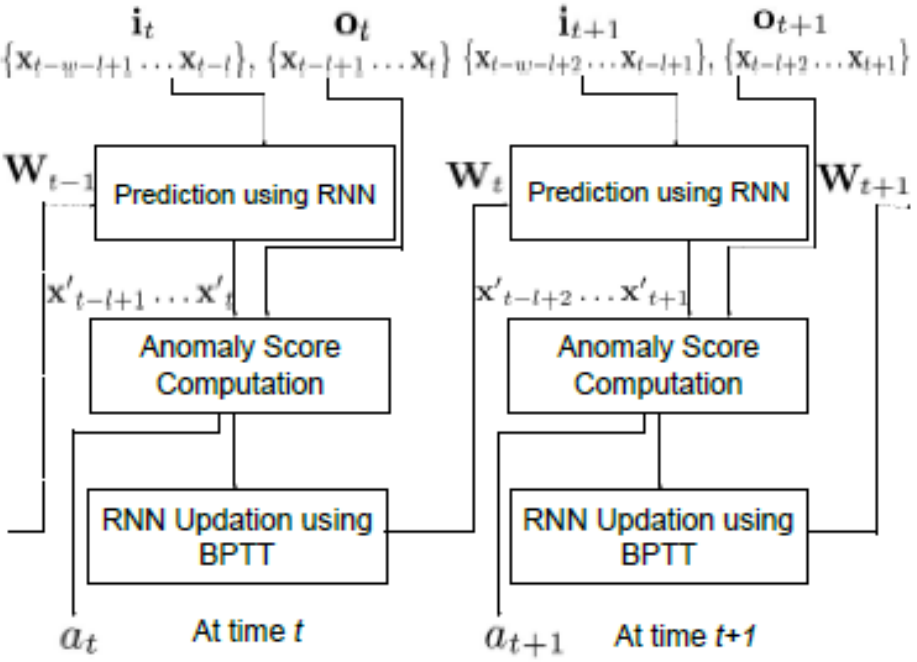


Fig. 2: Steps in Online RNN-AD approach [15]

In the online setting, such a model of normal behavior needs to be updated in an incremental manner as new data arrives. We use the most recent prediction error(s) to serve two primary purposes: i) update the model parameters, ii) compute the anomaly score. We identify following key properties of a good online anomaly detection model based on prediction errors:

- If there is a drift in values being observed over a significant period of time, the model should quickly adapt to the new norm by updating the parameters.
- If there is a short sub-sequence of anomalous values, the model need not be updated. Rather the model updating step should be restrained.
- If there is a continuous drift in the normal behavior, the model should get updated continuously or model the continuous drift as part of the normal behavior.

Overall steps of the algorithm are depicted in Figure 2.

C. Anomaly Detection Using Median Absolute Deviation [7]

The mean absolute deviation of a dataset is the average distance between each data point and the mean. It gives us an idea about the variability in a dataset [1]. The formula [12] of median absolute deviation is:

$$MAD = median|x - \tilde{x}|$$

where $\tilde{x}$ is the median of the variable. This statistic is sometimes used as a robust alternative to the standard deviation as a measure of scale. The scaled MAD is defined as

$$MADN = MAD/0.6745$$

MAD is immune to sample size. It is said to be the single most useful ancillary estimate of scale [6]. It is for example more robust than the classical interquartile range, which has a breakdown point of 25% only.

input : Streaming data, Threshold
output: Anomaly list of streaming data

$med \leftarrow median(data)$
$differences \leftarrow empty$
$anomaly\_list \leftarrow empty$

for $j \leftarrow 1$ **to** $len(data)$ **do**
  $diff \leftarrow |data[j] - med|$
  $differences.append(diff)$
end
for $k \leftarrow 1$ **to** $len(differences)$ **do**
  $modified\_z\_score \leftarrow 0.6745 * z$
  if $modified\_z\_score > Threshold$ **then**
    $anomaly\_list[k] \leftarrow true$
  else
    $anomaly\_list[k] \leftarrow false$
  end
end
return $anomaly\_list$

Algorithm 1: MAD anomaly detection [13]

MAD can be used to detect anomaly/outlier in different sample sizes. First of all, a range of data needs to be selected on which the algorithm will work. We will call this range "Window". After that, a feasible threshold needs to be selected. If the scoring of data using MAD reaches above the threshold, then it is an anomaly. Depending on the stringency of the researcher's criteria, which should be defined and justified by the researcher, the values of 3 (very conservative), 2.5 (moderately conservative) or even 2 (poorly conservative) can be selected [9].

D. Long-Term Anomaly Detection in the Cloud [17]

This process was developed in Twitter. A novel statistical technique to automatically detect long-term anomalies in cloud data. Specifically, the technique employs statistical learning to detect anomalies in both application as well as system metrics. Further, the technique uses robust statistical metrics, median, median absolute deviation (MAD), and piecewise approximation of the underlying long-term trend to accurately detect anomalies even in the presence of intra-day and/or weekly seasonality.Multiple teams at Twitter are currently using the proposed technique on a daily basis.

input : Timeseries(X), maximum number of anomalies(k)
output: Anomaly list of streaming data

Determine periodicity/seasonality
Split X into non-overlapping Windows $W_x(t)$ containing at least 2 weeks

foreach $W_x(t)$ **do**
  n_w = number of observations in $W_x(t)$
  $k \leq (n_w \times .49)$
  Extract seasonal S_x component using STL[2]
  Compute median, $\tilde{X}$
  Compute residual, $R_x = X - S_x - \tilde{X}$
  /*Run ESD[14]/detect anomalies vector X_A with $\tilde{X}$ and MAD in the calculation of the test statistics*/
  $X_A = ESD(R_x, k)$
  $v = v + X_A$
end
return v

Algorithm 2: Piecewise median anomaly detection [17]

In this method, the underlying trend in a long-term time series is approximated using a piece-wise method. Specifically, the trend computed as a piece-wise combination of short-term medians. Based on experimentation using production data, twitter observed that stable metrics typically exhibit little change in the median over 2 week windows. These two week medians provide enough data points to decompose the seasonality, while also acting as a usable baseline from which to test for anomalies. [17]

III. Methodology

In this section, the LSTM technique discussed in section II-A and MAD technique discussed in section II-C is implemented and tested. The tools and environments are described in the next section. Dataset used for these test are from The Numenta Anomaly Benchmark(NAB Benchmark) [10]

A. LSTM anomaly detection implementation

1) Loading data (Figure 3): For this test, first of all we created a virtual environment containing tensorflow and keras in PyCharm. Next we need to read the data from the downloaded dataset. The dataset is a one valued time series, that means each row of the dataset contains a timestamp and a value against it.

2) Processing data (Figure 4): Once all the datapoints are loaded as one large timeseries, we have split it into examples. Again, one example is made of a sequence of 100 values. Using the first 99, we are going to try and predict the 100th.

After that we have split the data into training set and test set as normally is done in machine learning algorithms. Though normally around 70% data is taken for training and 30% for test, we have made a 50-50 split for this experiment.

```
def get_file_data():
    fileData = []
    with open('nyc_taxi.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        for row in csv_reader:
            if line_count > 0:
                fileData.append(float(row[1]))
            line_count += 1
    return fileData;

run_network()
```

Fig. 3: Load CSV data [15]

```
def prepare_data():
    data = get_file_data()
    train_start = 0
    train_end = int(len(data) * 0.50)
    test_start = train_end + 1
    test_end = len(data) -1

    # train data
    print ("Creating train data...")
    result = []
    for index in range(train_start, train_end - sequence_length):
        result.append(data[index: index + sequence_length])
    result = np.array(result)  # shape (samples, sequence_length)
    result, result_mean = z_norm(result)

    print ("Mean of train data : ", result_mean)
    print ("Train data shape  : ", result.shape)
    train = result[train_start:train_end, :]
    np.random.shuffle(train)  # shuffles in-place
    X_train = train[:, :-1]
    y_train = train[:, -1]

    # test data
    print ("Creating test data...")
    result = []
    for index in range(test_start, test_end - sequence_length):
        result.append(data[index: index + sequence_length])
    result = np.array(result)  # shape (samples, sequence_length)
    result, result_mean = z_norm(result)

    print ("Mean of test data : ", result_mean)
    print ("Test data shape  : ", result.shape)
    X_test = result[:, :-1]
    y_test = result[:, -1]
    print("Shape X_train", np.shape(X_train))
    print("Shape X_test", np.shape(X_test))

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

    return X_train, y_train, X_test, y_test
```

Fig. 4: Prepare data

```
def build_model():
    model = Sequential()
    layers = {'input': 1, 'hidden1': 64, 'hidden2': 256, 'hidden3': 100, 'output': 1}

    model.add(LSTM(
            input_length=sequence_length - 1,
            input_dim=layers['input'],
            output_dim=layers['hidden1'],
            return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
            layers['hidden2'],
            return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
            layers['hidden3'],
            return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
            output_dim=layers['output']))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop")
    print ("Compilation Time : ", time.time() - start)
    return model
```

Fig. 5: Build model

We have shuffled the training examples so that we train in no particular order and the distribution is uniform. Then we reshape the inputs to have dimensions. Here each value is 1-dimensional, they are only one measure.

3) Building model (Figure 5): After processing the data, it is time to build a model which is the most crucial part of machine learning algorithms. So here we are going to build our Sequential model. This means we're going to stack layers in this object. Here we have built a network with 1-dimensional input, three hidden layers of sizes 64, 256 and 100 and a 1-dimensional output layer. We have added a 20% Dropout in each layer. Then, we have added a Dense layer. Lastly, we compile the model using a Mean Square Error, which is a standard for regression and the RMSprop optimizer.

4) Fitting data, predicting and detecting anomaly (Figure6): The next step of building a model is to fit the data. Using Keras's built-in function, we fit the shaped data according to our model. After that, we predict the values of the test data. So, now as we have the predicted data and also the actual data for those predictions, we can calculate the error rate. We have used "Squared Error" for our test. Now, we put a threshold(i.e. 5) on that value of error. We can then say, any value above the threshold is an anomaly.

B. MAD anomaly detection implementation (Figure 7)

For this technique, we do not need any training data. So, to contrast this technique with the previous one, we have taken the same data from the same dataset.

- The code for this is straight forward. We take the median of the current dataset.
- Find out absolute difference of each data point from the median.

```
try:
    print("Training...")
    model.fit(
            X_train, y_train,
            batch_size=batch_size, nb_epoch=epochs, validation_split=0.05)
    print("Predicting...")
    predicted = model.predict(X_test)
    print("Reshaping predicted")
    predicted = np.reshape(predicted, (predicted.size,))
except KeyboardInterrupt:
    print("prediction exception")
    print ('Training duration (s) : ', time.time() - global_start_time)
    return model, y_test, 0

# plotting figure
try:
    plt.figure(1)
    plt.subplot(311)
    plt.title("Actual Test Signal")
    plt.plot(y_test[:len(y_test)], 'b')
    plt.subplot(312)
    plt.title("Predicted Signal")
    plt.plot(predicted[:len(y_test)], 'g')
    plt.subplot(313)
    plt.title("Squared Error")
    mse = ((y_test - predicted) ** 2)
    plt.plot(mse, 'r')
    plt.show()
except Exception as e:
    print("plotting exception")
    print (str(e))
print ('Training duration (s) : ', time.time() - global_start_time)
```

Fig. 6: Fit data, predict

```
def isMADAnomaly(data,thresh):
    med = np.median(data)
    isAnomalyList = []
    diffs = []
    for d in data:
        diff = abs(d-med)
        diffs.append(diff)

    med_abs_deviation = np.median(diffs)
    for z in diffs:
        modified_z_score = 0.6745 * z / med_abs_deviation
        # print(modified_z_score)
        if modified_z_score > thresh:
            isAnomalyList.append(1)
        else:
            isAnomalyList.append(0)

    return isAnomalyList
```

Fig. 7: MAD implementation

- Take the median of the absolute differences.
- Calculate the modified z score according to formula.
- If the score is higher than the threshold, then list it as anomaly.

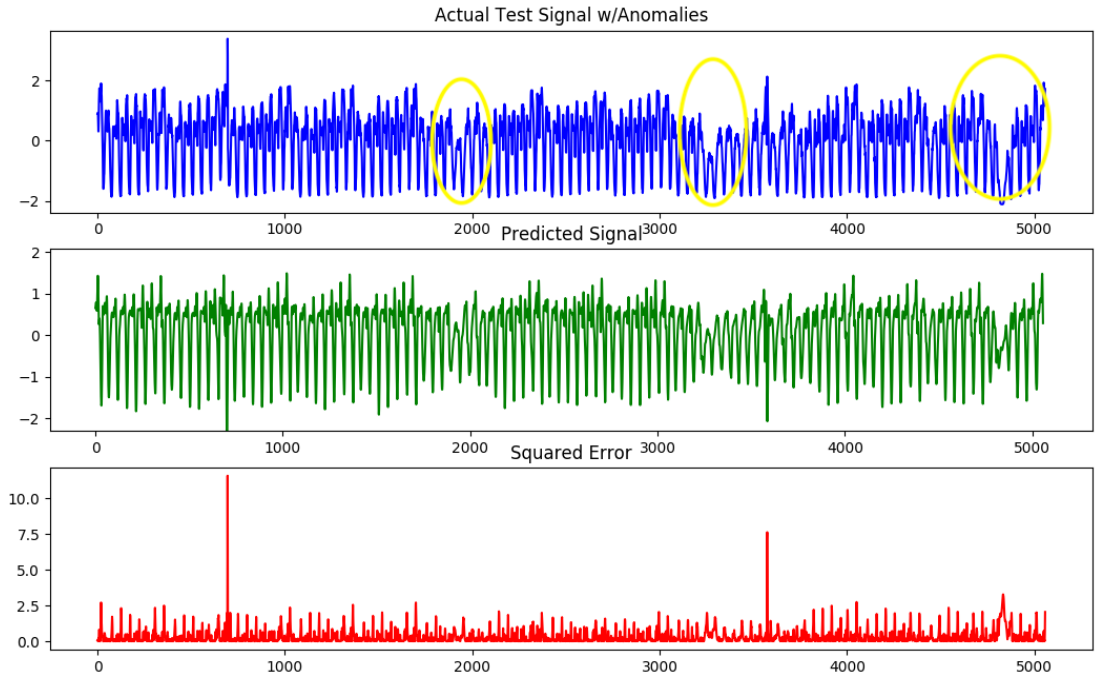
We put the anomalies in the $isAnomalyList$ array. Then we have plotted it in a graph.

IV. Result of Experiment (Figure 8)

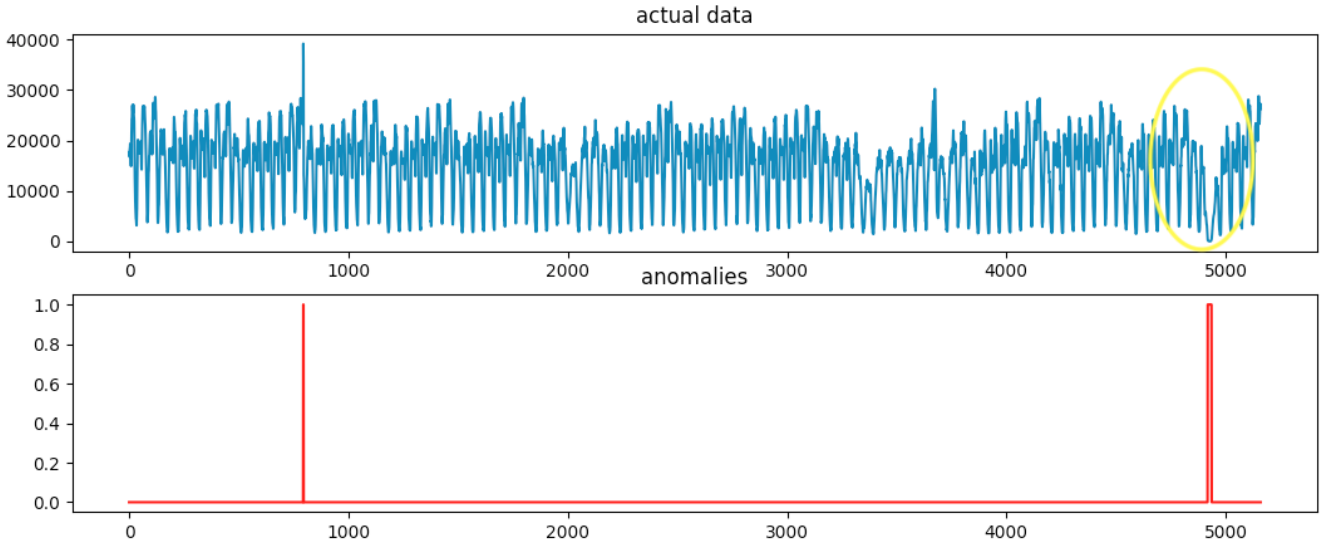
The LSTM technique took around **106.7 seconds**. The MAD technique took around **0.1 second**.

However, from figure 8a, we can see that LSTM does learn some trends(yellow marked) over time. But from figure 8b, we can see that MAD catches the periodic shape as anomaly. Also, it misses one of the anomalies that LSTM catches. However, the missed one is actually not so far away from the actual contour of the dataset.

Fig. 8: Comparison of LSTM and MAD technique



(a) LSTM output



(b) MAD output

V. Environment

OS: macOS High Sierra.
Language: Python 3.6
IDE: PyCharm
External Libraries: Tensorflow with Keras [16]

VI. HARDWARE

Processor: 2.5 GHz Intel Core i5
Primary Memory: 8 GB 1600 MHz DDR3
Hard disk: 250 GB Solid State SATA Drive

VII. CONCLUSIONS

The techniques described in this paper are all used in practical scenarios depending on the field of interest. However, from the above experimentation, we can say that if we are not looking for trends over long time period and need a real time anomaly detection MAD technique is a good choice. But, if we are more inclined towards accuracy than time consumption, LSTM technique is the better choice. In future, the concept drift LSTM technique and other methods is planned to be implemented and compared with each other.

ACKNOWLEDGMENT

- Dr. Muralidhar Medidi, Department head of Computer Science Department, Georgia Southern University
- Muhammad Mahmodul Hoque Rana, Georgia Southern University

References

[1] Khan Academy. Mean absolute deviation (mad) review. [Online; Available:https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/other-measures-of-spread/a/mean-absolute-deviation-mad-review].

[2] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition. *Journal of Official Statistics*, 6(1):3–73, 1990.

[3] B. Dorsemaine, J. Gaulier, J. Wary, N. Kheir, and P. Urien. Internet of things: A definition amp;amp; taxonomy. In *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 72–77, Sept 2015.

[4] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model. *arXiv preprint arXiv:1612.06676*, 2016.

[5] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, Sept 2014.

[6] Peter J. Huber. *Robust Statistics*, pages 1248–1251. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[7] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764 – 766, 2013.

[8] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[9] Jeff Miller. Reaction time analysis with outlier exclusion: Bias varies with sample size. *The quarterly journal of experimental psychology*, 43(4):907–912, 1991.

[10] Numenta. Nab benchmark dataset(nyc taxi. [Online; Available: https://github.com/numenta/NAB/blob/master/data/realKnownCause/nyc_taxi.csv].

[11] NVIDIA. Long short-term memory (lstm). [Online; Available: https://developer.nvidia.com/discover/lstm].

[12] National Institution of Standards and Technology(NIST). Median absolute deviation. [Online; Available: https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/mad.htm].

[13] Joo Rodrigues. Outliers make us go mad: Univariate outlier detection. [Online; Available: https://medium.com/james-blogs/outliers-make-us-go-mad-univariate-outlier-detection-b3a72f1ea8c7].

[14] Bernard Rosner. Percentage points for a generalized esd many-outlier procedure. *Technometrics*, 25(2):165–172, 1983.

[15] Sakti Saurav, Pankaj Malhotra, Vishnu TV, Narendhar Gugulothu, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Online anomaly detection with concept drift adaptation using recurrent neural networks. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 78–87. ACM, 2018.

[16] Tensorflow. Tensorflow. [Online; Available: https://www.tensorflow.org/].

[17] Owen Vallis, Jordan Hochenbaum, and Arun Kejariwal. A novel technique for long-term anomaly detection in the cloud. In *HotCloud*, 2014.