Data Analytics
Stiftung Universität Hildesheim
Marienburger Platz 22
31141 Hildesheim
Prof. Dr. Dr. Lars Schmidt-Thieme

Thesis
Unsupervised Real-Time Time-Series Anomaly Detection

Abdul Rehman Liaqat

271336, Liaqat@uni-hidesheim.de

Contents

List of Figures

List of Tables

Abstract

Anomaly detection is a crucial task for machine learning due to wide-spread usage and type. In particular, it is worth noting that most data arising in industrial setups are of a streaming nature, thus restricting the range of standard anomaly detection tools. This thesis will identify the potential approaches to learn the identification of abnormal behavior from large-scale streaming data. An empirical comparison of state-of-the-art methods will to be extended by a novel technical contribution. In this thesis, the focus is particularly on streaming time-series Anomaly Detection which changes in nature with time and novel contribution will especially try to target this dynamic nature of time-series.

1 Introduction

1.1 Usage of streaming data

With the increase in networking of objects, amount of data being generated is increasing. A big chunk of this data involves streaming data. Streaming data can be defined as data being generated continuously or with a continuous time interval. Such data is produced incrementally without having access to all of the data. Streaming data term is ususally used in context with big data. These streams of data is dumped into a large amount of unstructured databases called data lakes, which will be later treated in a batchwise fashion for the extraction of useful information. Streaming data is the most fresh form of data and is closest to being real-time, this is why any action or analysis done on top of data stream translates to near real-time action or analysis. Few examples of such real-time actions are e-commerce, network monitoring, fraud detection, pricing and bidding and so on.

Since streaming data is generated continuously it is assumed that it will keep on being generated forever. Best example of such streaming data is internet of things (IoT) sensors which almost all the time generate streaming data. For example a heat sensor of exhaust fan of a data center is producing time series stream of data. This temperature can depicts many different facts about the data center and the condition of the fan itself. Increase in the temperature can signify high load on the data ceter hence higher usage and higher temperature generation. Similarly sudden change in the temperature can signify a failure of fan which should be replaced on urgent basis. Statistically, IoT data has grown from 4.4ZB (Zeta Bytes) to 44.4ZB with 50 billions devices. As the data is being generated continuously and endlessly, it is very likely that the data will change it's shape based upon external variables which are effecting the sensor or the data generating system itself. Taking the same example of exhaust fan temperature sensor. After changing the exhaust fan and installing a new one, it is very likely that the output tempreature time series stream is very different then the one generated by the previous exhaust fan temperature sensor. This signifies a possible permanent change in the streaming data which should be seriously taken if anykind of stream processing is being done.

These IoT devices are covering a lot of different fields including a small temperature sensor installed in a room to ECG data to satellite communication data. Even videos are kind of streaming data since they include data points (frames) separated by a constant interval. Data generated from human mounted sensors such as smart watch is also a kind of streaming data. Since the data being generated is continuous and infinite in nature, it is necessary to have techniques which are using data the same way. In other words, to make use of this data it is necessary to build system which are also running live and changing themselves online. Similarly, it is important to keep the system running as often times the components which are generating streaming data are at the core of the system and in case of failure of these components, there is a danger of whole system being out of service. That is why it is necessary to find out any unusual behavior of the data being generated by the component.

Early detection of such unusual behavior can help avoid any catastrophic impact on the system. It can also alarm the system before the break down of any thing hence acting as predictive prevention. This kind of unusal behavior is called anomaly and such detection of such unusual behavior is called anomaly detection.

1.2 Usage of anomaly detection

An anomaly is defined as significantly different data point or set of data points from the historical values or the expected values. There are various definitions of the "difference" between data points which leads to different types of anomalies. Detecting an anomaly helps in identifying abnormal behavior of a process with potentially useful information. For example, detecting an anomaly in the temperature sensor output data of exhaust sensor of a data center can signify a problem with the fan or the data center itself. Depending upon the type of anomaly it can translate to a failure in the data center or the break down of the fan itself. Now, in the systems such as data centers, detecting such anomaly as early as possible and taking action as swiftly as possible is extremely important. Such detection can save a catastrophic failure of the whole data center.

In normal machine learning and data science projects, anomaly and outlier detection is an important step which will help clean the data and provide useful insights. As described in [1], going forward, software development especially machine learning based software development will be divided into two parts. One part is the preparation of the data and the second part is the design and optimization of algorithms. Both of these steps are used iteratively. Interestingly, as described in the [1], more and more time is spent on accumulating, massaging and cleaning datasets. One major part of this process is outlier detection, anomaly detection and novelty detection. By detection of anomalies, it will become sufficiently easy for human expert to find the one data point in a heap of millions data points and then decide whether to remove that data or get more of the same data points. It also helps provide a useful insight and analysis of edge cases or extreme cases which can be of high importance.

1.3 Usage of anomaly detection in streaming data

As described above, anomaly detection is an important part of normal machine learning and data science projects. It is also very useful as first line of analysis on streaming data. It can act as real-time anomaly detection as the input to the anomaly detection is streaming data. This way one can detect any kind of anomaly occuring on the system real-time.

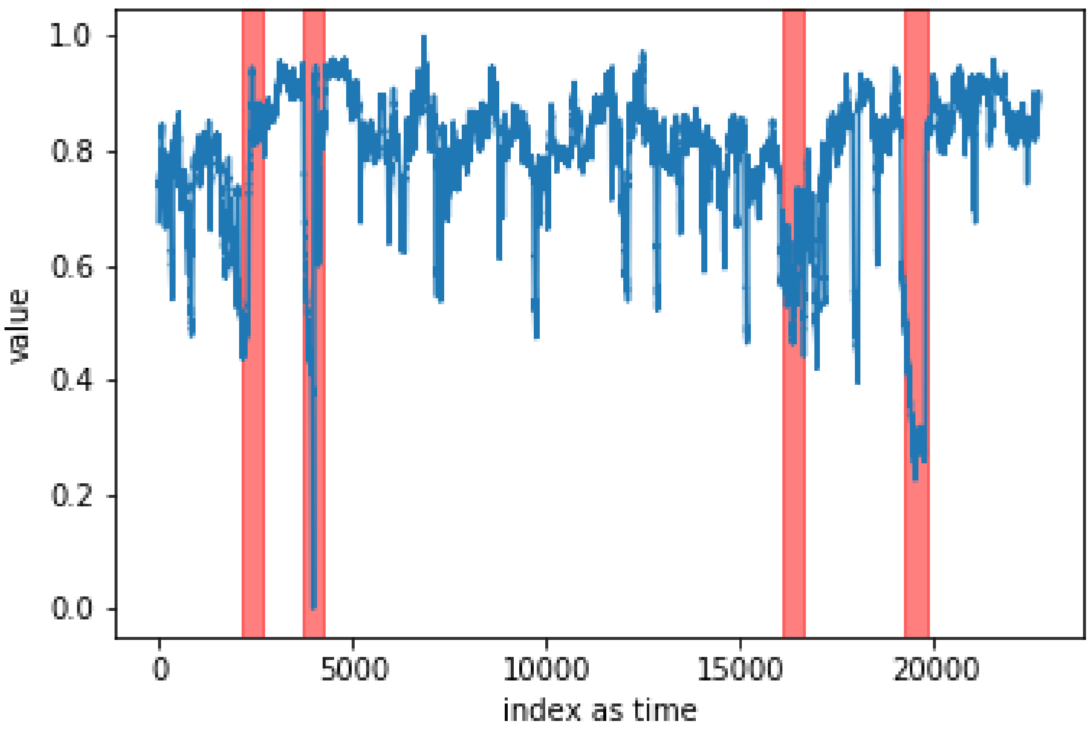


Figure 1: Machine Temperature System Failure

For example figure (1) shows real-world temperature sensor data from an internal component of a large industrial machine. Anomalies are highlighted with red background. The second anomaly was a planned shutdown hence there was an exceptional decrease in temperature. Last anomaly was a catastrophic system failure. Third anomaly just before the failure was a signal predicting the failure. If the anomlay detection system correctly detected the third anomaly, system level failure could have been avoided. Such examples motivates to define an ideal anomaly detection system. As suggested by [2] the characteristics of an ideal anomaly detections system are:

1. Predictions must be made online which means that the algorithm must identify state x_t as normal or anomalous before receiving the next data point x_{t+1}
2. The algorithm must learn continuously without a requirement to store the entire stream.
3. The algorithm must run in an unsupervised, automated fashion. In other words the algorithm should not be fed any labeled data or shouldn't have any manual parameter changing.

4. Algorithms must adapt to dynamic environments and concept drift, as the underlying probability distributions often change.

5. Algorihtms should make anomaly detections as early as possible.

6. Algorithms should minimize false positives and false negatives.

Each of the characteristics is backed by real world requirements and justification. These characteristics also leads to motivation of the problem and proposed methodology.

1.4 motivation

First characteristics is the continuous arrival of the data. We cannot have access to any future data. More specifically we will only have access to current time data as x_t and previous data points as $x_{t-1}, x_{t-2}, x_{t-3}, ...$ and so on. This apply a constraint of finding the importance of data in the past and update the model online. Also as the data stream is continuous it becomes evident to have the system doing online prediction. Thus the first contraint of an anomaly detection algorithm is that it should predict online.

Second characteristics of an ideal anomaly detection algorithm is to train itself in an online fashion. This characteristics is derived from the requirement that data is of streaming type. There is no knowledge of batch end size or ending of stream. Also streaming data has high velocity and volume which is why it is no suitable to store all previous training data and then train on it. Instead it is advisable to ideally train on latest data point or small batch of latest data points. Another possibility is to derive features from previous data and use them to train the algorithm at current time step.

Third characteristics is to have an unsupervised algorithm. This is derived from the fact that streaming data is usually not labeled. Streaming data is meant to be of huge size and high velocity and to detect anomalies in real-time streaming data, it is difficult even for the experts in the domain. Also next characteristics also implies another requirement which is the cause of having anomaly detection algorithm unsupervised.

Next characteristics of an ideal anomlay detection algorithm is that the algorithm must have the ability to adapt to dynamic environments and **concept drift**. Concept drift is a special concept for which a quick review of literature is required hence, starting with the official definition of anomaly then defining type of anomalies and basic properties of time series streaming data, we will define concept drift.

An anomaly is defined as a data point X_t or set of data points $X_{t-l:t}$ for which either of following is true:

1. Probability of point X_t belonging to a probability distribution ϕ is very less then the probability of previous data points X_n where $n \leq t-1$. In other words:

$$P(X_t) \in \phi \ll P(X_n) \in \phi \tag{1}$$

This type of anomaly is called point anomaly. Here ϕ is the probability distribution generating all the data points. An example of such anomaly is given in 2

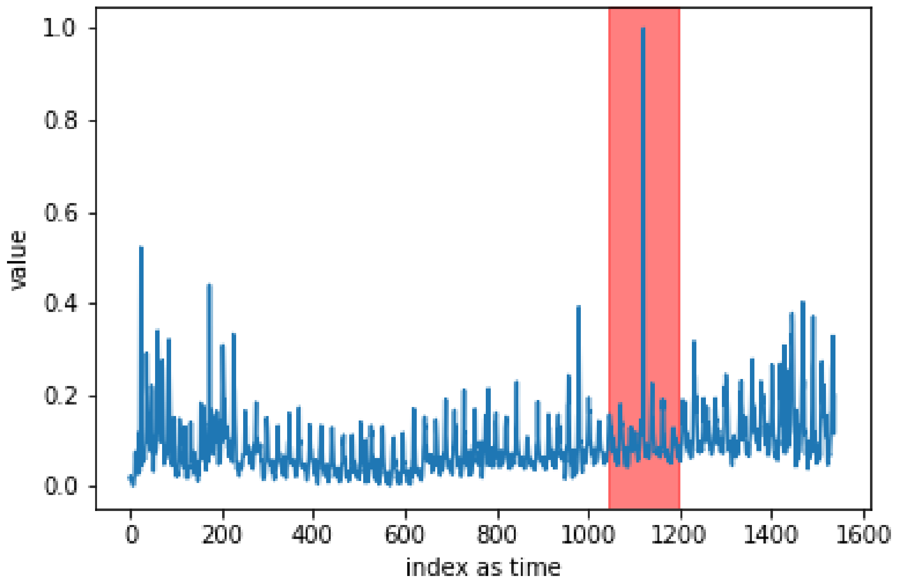


Figure 2: Cost Per 1000 impressions - An example of point anomaly

2. Probability of points $X_{t-l:t}$ belonging to a probability distribution ϕ is very less then the probability of $X_{n-l:n}$ where $n \leq t-1$. In other words:

$$P(X_{t-l:t}) \in \phi \ll P(X_{n-l:n}) \in \phi \tag{2}$$

This type of anomaly is called collective anomaly. An example of collective anomlay is given in 3. This type of anomaly requires short term context and window of past set of values to be detected. Thus the anomaly detection algorithm should have history of past $t-w$ time steps or extracted features from these $t-w$ time steps. Here t is the current time and w defines the window of time.

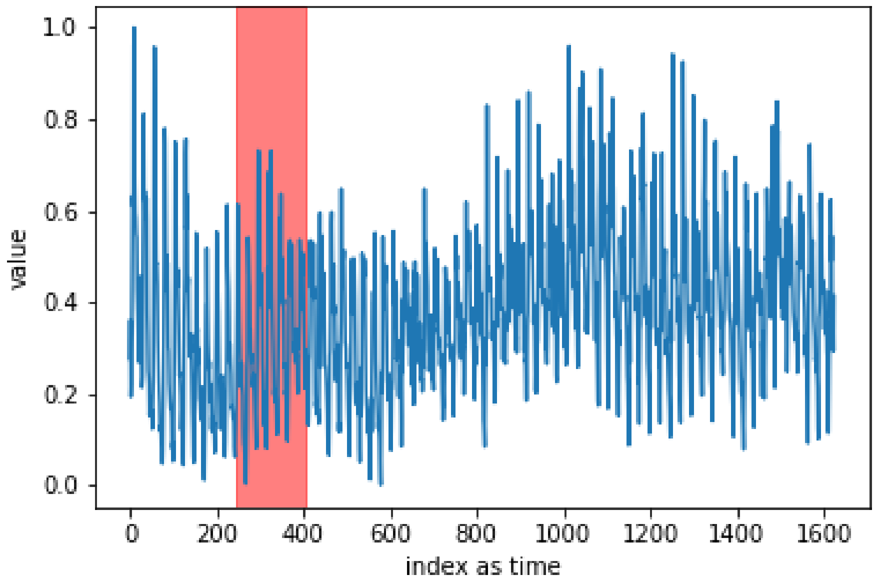


Figure 3: Cost Per Click with time - An example of collective anomaly

3. Probability of point X_t belonging to a probability distribution ϕ is almost same as the probability of previous data points X_n where $n \leq t - 1$ but the probability of point X_t belonging to a probability distribution θ is very less than the probability of point X_p belonging to θ. Here In other words:

$$P(X_t) \in \phi \approx P(X_n) \in \phi \tag{3}$$

$$P(X_t) \in \theta \ll P(X_n) \in \theta \tag{4}$$

   Here θ and X_p are probability distribution and data points. *theta* is a distribution defined to a set of local points X_p where $p \leq t$ and p is a set of indices. This type of anomaly is called Contextual anomaly. Contextual anomaly require both short term and long term contenxtual information which makes it difficult to detect. It also enforce a requirement to the algorithm to have the ability to be able to store both long term and short term important information.

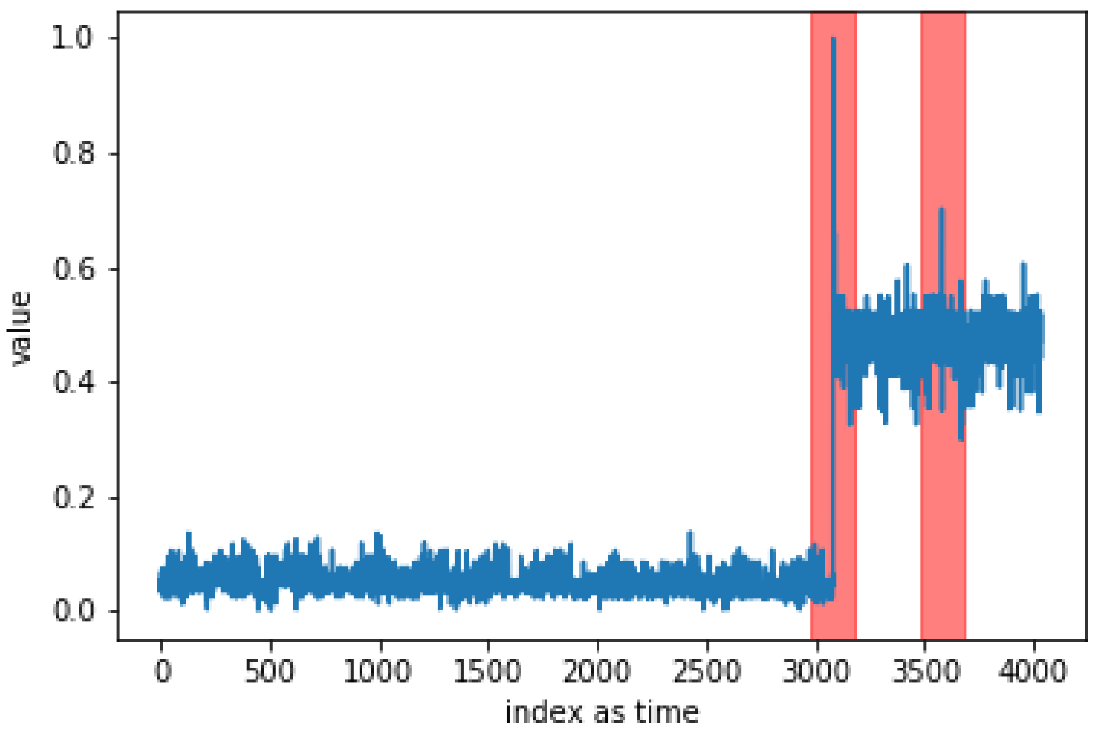


Figure 4: AWS CPU Utilization Concept Drift

1.5 Constraints

In many cases there is a system change which causes a permanent change in the probability distribution being generated by the data and this change is called **concept drift**. An example is in figure 4, where software updates and configuration changed can change the behavior of the system forever. Now the system must adapt to detect anomalies according to the new type of data. To do that the model must first recognize that if it is a concept drift and then change itself accordingly, that too in an unsupervised and automated way. There is also an intuitive meaning behind the name of Concept drift.

Considering that there is an underlying probability distribution in every time series and this underlying distribution is named Concept of that time series such as

$$Concept = P_t(X) \tag{5}$$

The the concept drift occurs between two times t and u when the distribution or concept changes as

$$P_t(X) \neq P_u(X) \tag{6}$$

And the new data points are generated from the second distribution $p_u(X)$ onward rather then the original $P_t(X)$ for a length of time T. This time T is decided by the domain knowledge and frequency of data points. As in real life systems it is highly likely that there will be a change in the concept of the time-series hence happening of concept drift is very likely and possible. For example, there will be a concept drift in the time-series output of a heat sensor measuring the temperature of a motor of exhaust fan and the motor of the fan is changed. Thus it is necessary for our system to first recognize the concept drift. If there exists a concept drift then the model should adapt itself swiftly to avoid false positives. Similarly if there is not a concept drift then the model should not give much learning weight to these readings. Thus the model adaption according to presence of concept drift is necessary. This leads to fourth characteristics of an ideal anomaly detection algorithm that it should adapt itself according to the concept drift.

In streaming applications it is also very useful to detect the anomaly as early as possible. For example in the streaming data of cardiac patient's heart; earlier detection can help in taking emergency measures to stop the onset of heart attack. Obviously detecting them a little earlier is way better than later. This early detection acts as an alarm and will really fulfill the purpose of alarm when enough time is remaining to take action to avoid catastrophic failure. So, the fifth characteristics of an ideal anomaly detector is devised from this requirement. If the predictions are made online or as early as possible such as before the arrival of next data point x_{t+1}, early detection of anomlay can be made possible.

Of course, having this early detection or online prediction property of an algorithm comes with drawbacks. For online prediction, algorihtm has to be fast which in turn constrain the requirement of having the less number of computations hence rather simple architecture. This is why, the algorithm will have an extra senstivity, which can cause many false alarms. Therefore, there is a trade-off between false alarms and true positives. One can setup variable sensitive algorithm which can be trained according to the importance of false positive, true positive and false negatives. This property is also introduced in the proposed method ahead.

Lastly, it is obvious that the anomlay detection algorithm must have ideally zero false positive and false negative rate and hunderd percent true postivie and true negative rate. Of course, it is difficult to fulfill all the requirement. Still the best detection algorithm will try to maximize the completion of all above. Secondly based upon the requirement of the system few requriements can also be relaxed or others few requirements can be given preference. For example, in case of heart's patient having anomaly detected as early as possible with extremely highly true positive will be necessary. In the process, few false postives can be afforded. On the other hand in a supply chain management system, where there are already a lot of alarms about low inventory are ringing, having false negatives can be afforded.

Concluding, with the requriements given above it becomes obvious that many current anomaly detection algorithms do not fulfill them as discussed in the related work section. We tried to create a system which fulfill these requirements to their maximum.

1.6 General framework of anomaly detectors and summary of each step

Generally, most of anomlay detection frameworks follow a pipeline of steps.These steps are described in figure 9. We will be summarizing each step to provide the extent of problems and possible area of innovations available in anomlay detection system and define our contribution along the way.

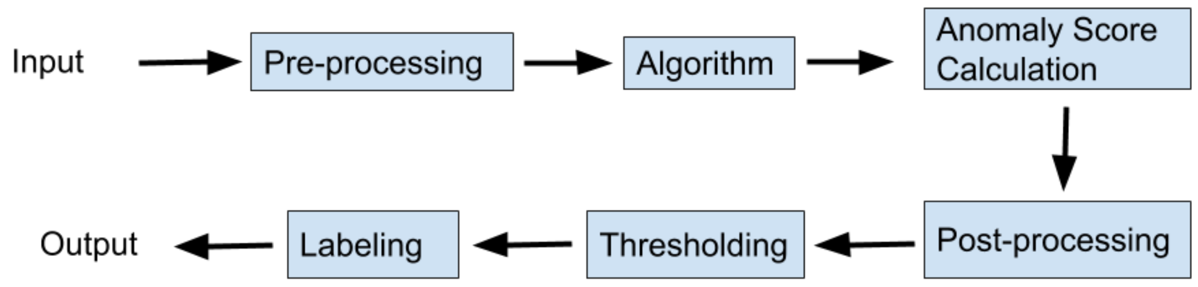


Figure 5: General Anomaly Detection Framework

First step is to pre-process the input data. Usually this step is performed to engineer features or do normalization of the data. That way the algorithm will be able to learn better. Preprocessing step can also be used to generate features from the data to be used as long term contenxtual information. Although it would be better to have domain expertise for the creation of such features. While doing normalization, one fact should be kept in mind that the regular methods of data normalization might not be valid for streaming data. For example, in case of normalization using standard deviation, it is required to have the mean and standard deviation of the data before hand, which is not possible in case of streaming data. Hence either modified version of normalization using standard deviation can be used or some other kind of normalization is prefered.

Second step is to estimate the underlying probability distribution. Instead of estimating the probability distribution directly, most of the time a model is built upon the streaming data such that the prediction of the model denoted by Y_t at time t is closed to the actual value X_t at time t. Thus it is assumed that the model will automatically learn the underlying distribution of the data. Now depending upon the type of input and output required, different kind models can be trained. Assuming that any past $t - w$ data points are stored, one model can be trained to predict the point x_{t+1} using data from $x_{t-w:t}$. Another kind of model will instead try to reconstruct the whole input stream. These models are based upon autoencoders. Usually the difference of the output value Y_t and actual value X_t gives us some significant information about the data at X_t. As the model prediction is actually the prediction of modeled distribution of data hence any significant difference of the predicted or output value and actual or input value respectively, can be hint of an anomaly. Also the model should follow the characteristis of an ideal anomaly detection algorithm defined above. On top of that, there is another open quenstion about the complexity and evaluation criterion of the model. The model can have very less predictive error but still the whole anomlay detection system might not be able to detect the anomaly. Similarly a model can be relatively less accurate but still be able to detect all types successfully. The ability of detecting a concept drift and then adapting itself should also be the part of model. The model should be built carefully so as it can learn online and is not fully converged as it will be trained continuously with incoming of data.

Next step would be to score the output of the model. At this moment we will use the previously estimated probability distribution as model and types of anomalies defined above to score the incoming data point. Scoring each data point is also an important area of innovation. Each point should be scored if it is an anomaly or not. The score should be based upon the history of model predictions and actual point plus current prediction and actual value. The score will try to quantize each data point according to the types of anomalies described above. Depending upon the type of model algorithm used, anomlay score can have different ranges. This is not the final score to be considered as it passes through a post-processing step which is also very important in removing the noise and building far and near contextual connections.

Afterwards the score is passed through a post-processing step. In this step possibly normalization and probabilistic likelihood estimation is applied on top of the previously calculated anomaly score. In other words previously calculated anomaly score is transformed between the range of 0 and 1 thus scoring each data point between defined range. This constraint helps in defining a common threshold for all types of data sets in the next step. This step is also important and is an important area of innovation. It can be used to compliment the previous step of score calculation. One can also apply another model at this point which can be used to capture the long term and short term patterns in the anomaly scores itself.

After post-processing, we can use the score to actually rank each incoming data point among previously existing data points. To label each data point as anomaly or not anomaly a thresholding procedure is necessary. Depending upon the type of post-processing extreme cases are considered as anomalies. Since all types of data is subjected to a defined range, it is easier to define a common threshold for all.

1.6.1 Our Contribution

We contributed in different areas of general anomaly detection framework.9. First contribution was the implementation of Numenta Anomaly Detection Benchmark (NAB) [3]. NAB is an anomlay detection benchmark which comes with labelled dataset, customized scoring mechanism and benchmark algorithms to test against. This benchmark is an open source project implemented in Python2.7. So, in the first step NAB was implemented and results of benchmark algorithms were reproduced. Additionally, system was setup to test proposed methodologies using same dataset and scoring function.

Second contribution is the implementation and testing of multi-step prediction algorithms. Three different types (fully connected, convolutional and LSTM) based multi-step prediction architecture were designed. Multi-step prediction algorithm, in short, refers to predicting multiple time step values at once. Changing the length of these steps from 1 to 5 provided interesting insights and helped break few benchmarks.

Third contribution is the implementation and testing of autoencoder based algorithms. These are also based upon three different types, just like previous one. Next contribution is in the form of autoregressive input of the loss from previous algorithms. This contribution serves two purposes. One is to improve the prediction accuracy of the algorithm especially because we are not using all previous data instead a small moving window as x_{t-w} where w is window size and it is tested for different values. Second is to have the ability to control the algorithm's response to concept drift. This method also proved to be useful as we passed few benchmarks along with few interesting insights.

Next contribution is the implementation of anomaly liklihood calculation of anomlay score as a post-processing step. This is the same implementation as described in [2]. Similarly, a second type of post-processing step is inspired by simple min-max normalization. We tried rolling min-max normalization which proves to provide better results than anomaly liklihood method on the same dataset.

1.6.2 Details of sections coming up-next

In the next sections we will first define the state of the art methods and related work needed as foundation for the understanding of proposed methodology. Next in the proposed methodology, we will describe each contribution and architecture in details. Then experimental setup and data set used will be explained in the next section. Afterwards results of these experiments and their comparison to other benchmarks will be done. Lastly, we will conclude with discussion on the experiments done providing useful insights, failed tries and future research direction.

2 Related Work and State-of-the-art

For the understanding of unsupervised anomaly detection in streaming data, it is important to have a summarized review of time series and it's features, general anomaly detection methods and general anomaly detection methods applied on streaming data. We will start with a quick review of time series.

Time series is usually defined as a series of data points indexed in time. A single variable where the value of variable is time series and the only other information available for each data point is the time of occurence, is called uni-variate time series while if there are multiple variables for each time index such as: $X = \{x^1, x^2, x^3, x^4, .., x^m\}$ where each point $x^t \in R^m$ in the time series is an m-dimensional vector $\{x_1^t, x_2^t, x_3^t, ..., x_m^t\}$. This is called multi-variate time series as defined in [4].

3 Proposed method

Here we will again use figure 6, as generic framework for anomaly detection system and start with describing each individual component design for the proposed methodology.

3.1 General anomaly score based architecture and detailed explanation

3.1.1 preprocessing

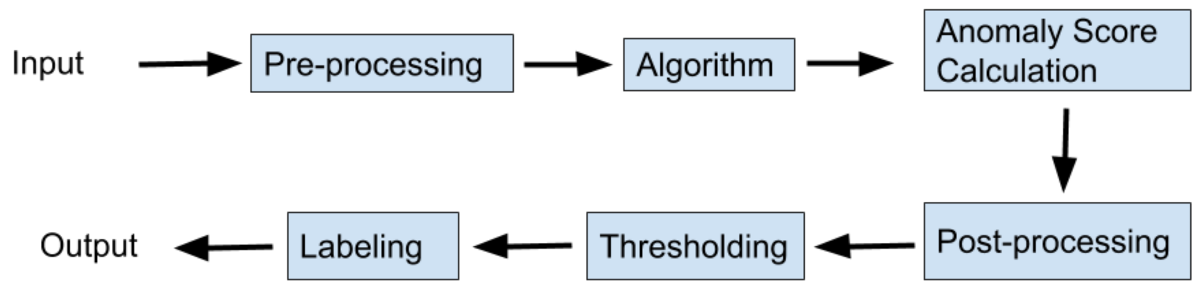


Figure 6: General Anomaly Detection Framework

As a first step, raw input of uni-variate time series is fed. Since the data is in the form of streaming format, hence we don't have any information about data beyond t. Instead of just using the value at time x_t, we actually process a window of size x_{t-w}. The value of w is used as a hyperparameter. Although it is dependent upon different data sets and types of algorithms but after testing a few different values, w is set to 50. Hence as a first pre-processing step, x_t is fed to buffer which will pop the last value in the queue and push x_t in. Using such window based input comes with few benefits and drawbacks. Since window size is small it is easier and quicker to process. Although in return, it turns simple time series prediction into a sequence to sequence prediction problem which might not able to take a time series features and properties into account. This was one of the problem faced with our proposed methodologies which we tried to cover using auto-regressive component. Using window made us ignroe the value of trend, seasonality, autocorrelation and similar imporant concept related to time series.

Another important pre-processing step is normalized using equation 7. Normalization helps the algorithm learn faster and since all type of datasets have same input range, it helps in better generalization. One assumption made for the normalization is that we already know the possible minimum and maximum values of the data which is generally true in most real life cases. Thus all of the data sets are normalized before feeding into the algorithm. We use min-max normalization to make sure that all input values are limited between 0 and 1. By including normalization of input as first step makes it possible to be able to use the same anomaly detection model framework for all kind of input data range. Min-max normalization can be defined as

$$z = \frac{x - min(x)}{max(x) - min(x)} \tag{7}$$

3.1.2 algorithm

Now that we have availibility to the preprocessed input, it is time to describe the design of proposed algorithms. All of the proposed algorithms are designed to consider the characteristics of an ideal anomaly detection algorithm. Since we cannot store the whole data and have access to future data, as it is one of the characteristics, that is why we are keeping a small batch of input data which can be easily stored in memory of the system. Other two characteristics, which are related to online training and prediction, enforce the algorithm to have less number of computations.

3.2 Multistep prediction architecture

First type of architecture used is multistep prediction architecture. In this type of architecutre sequence to sequence prediction task is trained. Input sequence for datapoint X_t at time t will be $X_{t-w-l:t-l}$. Here w is the window size of input and l is the length of output sequence. Thus the output will be $Y_{t-l:t}$. Intuitively, it means to take a sequence in the past and predict in the past starting from the end of the input sequence. More specifically the output prediction, when the input sequence $X_{t-w-l:t-l}$ is fed, the trained function h will be:

$$Y_{t-l:t} = h(X_{t-w-l:t-l}) \tag{8}$$

The input will be a uni-variate vector of size w and the output will be another uni-variate vector of size l. The final loss for which these multistep prediction architecture algorithms are trained against is mean squared error or MSE defined as:

$$L(x, y) = \frac{1}{l} \sum (Y_{t-l:t} - h(X_{t-w-l:t-l}))^2 \tag{9}$$

Multistep prediction architecture is motivated from [5]. The idea is to have feedback from more than one data points. This helps in reducing the senstivity of the model on only one data point. This also helps in reducing the noise in anomaly score. At the end, the parameters of the function h will be trained to minimze the loss function 9. Using the input, output and loss function defined above, we designed three different types of algorithms train the function h. The algorithms are based upon famous deep learning components. We designed three types of architecture based upon multistep predictions and tested them through experiments. These three types are:

1. Fully connected architecture based multistep prediction or NnMultiStep
2. CNN based architecture based multistep prediction or CnnMultiStep
3. LSTM based architecture or LstmMultiStep

Here size of architecture is usually kept smaller and compact. The reason is the same to have the ability to train and predict online and having smaller faster architecture is important. These algorithms are also modified later on to incorporate auto-regressive component. Moving on we will describe each of these architectures in detail.

3.3 Fully connected multistep ahead prediction architecture

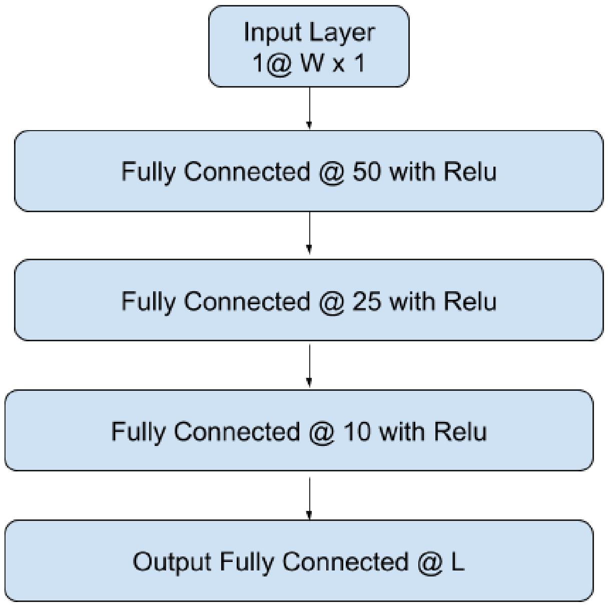

Input Layer
1@ W x 1
Fully Connected @ 50 with Relu
Fully Connected @ 25 with Relu
Fully Connected @ 10 with Relu
Output Fully Connected @ L

Figure 7: Fully connected Multistep Ahead Prediction with input length W and output length L

In fully connected multistep ahead prediction architecture as shown in figure 7, we use three fully-connected layers. Input is a one dimensional vector of window size w and can be represented as $X_{t-w-l:t-l}$. Input is passed to a fully connected layer of 50 cells. Next two fully connected layers with 25 and 10 number of neurons are set up. Lastly there is another fully connected layer with the number of neurons equal to l. Here l is multistep ahead prediction length or output sequence size. This architecture is rather rigid against different input size or different window size w. We test the architecture for different lenghts of w and l.

3.4 Convolutional multistep ahead prediction architecture

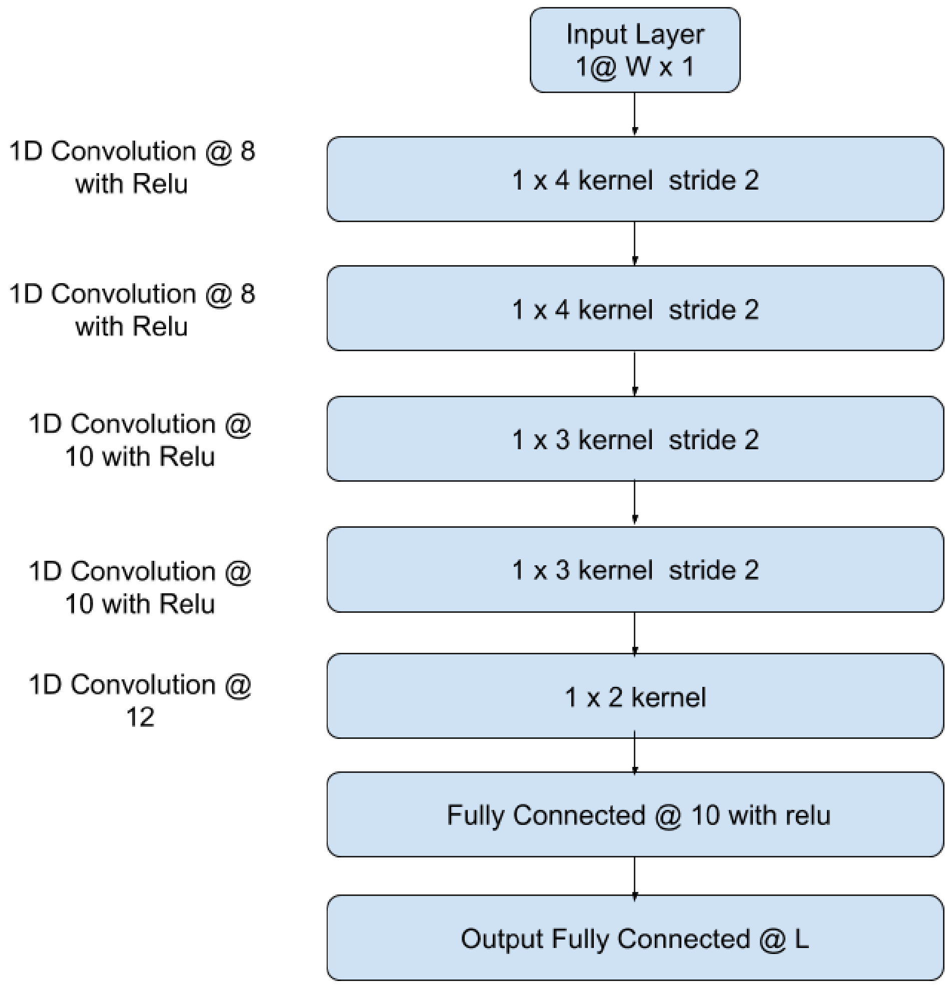

Input Layer
1@ W x 1
1D Convolution @ 8 with Relu
1 x 4 kernel stride 2
1D Convolution @ 8 with Relu
1 x 4 kernel stride 2
1D Convolution @ 10 with Relu
1 x 3 kernel stride 2
1D Convolution @ 10 with Relu
1 x 3 kernel stride 2
1D Convolution @ 12
1 x 2 kernel
Fully Connected @ 10 with relu
Output Fully Connected @ L

Figure 8: Convolutional Network based Prediction Network

In convolutional mutlistep ahead prediction architecutre, input is fed to a convolution layer with 8 different kernels of 1×4 with stride 2. Following is another layer of the same dimensions. Then we have two layers of 10 different kernels of 1×3 with stride 2. The last convolution layer consists of 12 kernels of dimension 1×2 with no stride. Activation function for all the neurons of convolution layers is relu. Next a fully connected layer with 10 neurons is set up whose activation function is relu. Lastly the output layer is a fully connected layers consisting of L neurons where L describes the size of output sequence.

3.5 LSTM multistep ahead prediction architecture

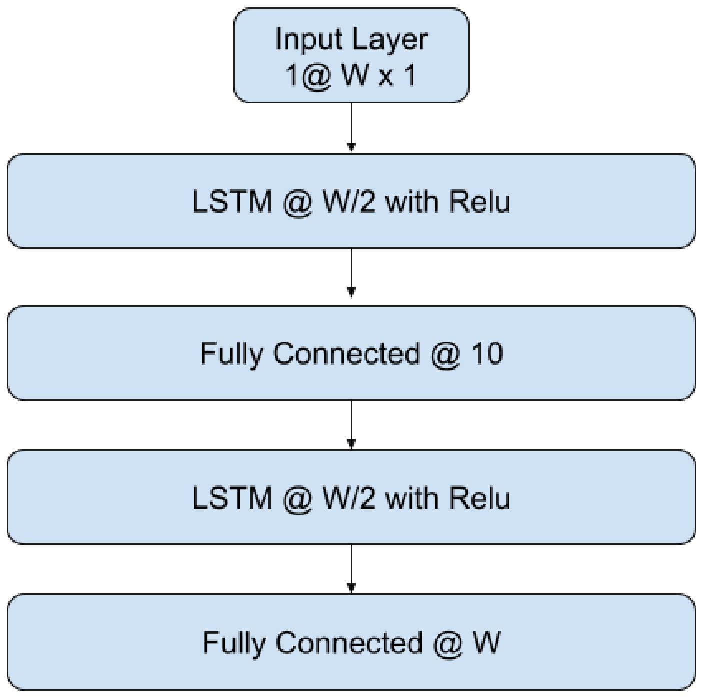


Figure 9: Convolutional Network based Prediction Network

LSTM based multistep ahead prediction architecture is rather simple. As usual input is a uni-variate vector of size W which is fed to 50 concatenated LSTM cells with Relu activation. Lastly a fully connected layers with L neurons and no activation is kept as an output layer.

3.6 Autoencoder based architecture

A second group of architecture is based upon autoencoder layout. As described in state-of-the-art, an autoencoder compresses the input into a compressed vector format which is then decoded to the original input shape.An autoencoder is a network which is trained to copy its input to its output, with the typical purpose of dimension reduction - the process of reducing the number of random variables under consideration. It features an encoder function to create a hidden layer (or multiple layers) which contains a code to describe the input. There is then a decoder which creates a reconstruction of the input from the hidden layer. An autoencoder can then become useful by having a hidden layer smaller than the input layer, forcing it to create a compressed representation of the data in the hidden layer by learning correlations in the data. This compressed representation is actually summarized version of input. This way for the anomaly input values, summarized version will not output very similar output and hence ending up creating big reconstruction loss. The backpropagation will try to $h_{W,b}(x) \approx x$, so as to minimise the mean square difference:

$$L(x,y) = \sum (x - h_{W,b}(x))^2 \tag{10}$$

Similarly for time series let X_t be the input value of the streaming data at time t then the input of autoencoder based architecture will be $X_{t-w:t}$ and output will be $Y_{t-w:t}$ as Y is the predicted reconstruction of the input.

$$Y_{t-w:t} = h(X_{t-w:t}) \tag{11}$$

Thus the input and the output will have an identical size. The loss function of an autoencoder is also called reconstruction loss as it is the difference between the actual input and the reconstruction of the input. Reconstruction loss is then defined as:

$$L(x,y) = \frac{1}{w} \sum (Y_{t-w:t} - X_{t-w:t})^2 \tag{12}$$

3.7 Fully Connected Autoencoder

In fully connected autoencoder architecture as shown in figure 10, we use a symmetric fully connected autoencoder. After input, first layer of encoder consist of 30 neurons with Relu activation. Second layer contains 15 and third layers is the last layer consisting of 7 neurons. This layers also represents endoded representation of input. Next layers are decoder and to follow the symmetry. Fourth layer has 15 nerons, fifth has 30. All previous layers has Relu as an activation function. Last layer contains the same number of neurons as the length of input vector which is w. Last layer has no activation function and the output should be identical to the input.

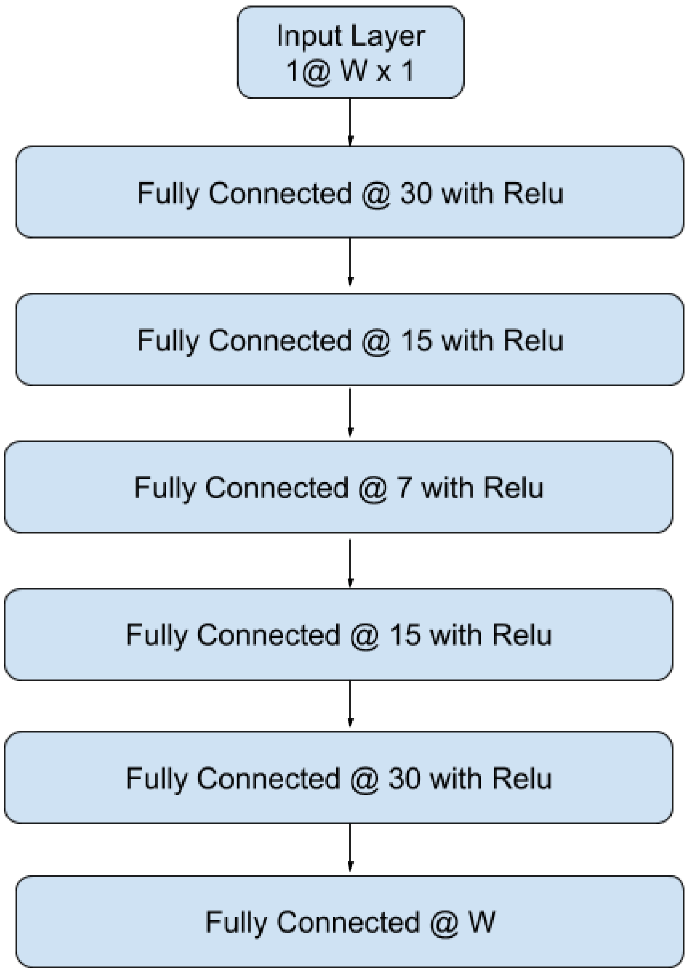

Input Layer
1@ W x 1
Fully Connected @ 30 with Relu
Fully Connected @ 15 with Relu
Fully Connected @ 7 with Relu
Fully Connected @ 15 with Relu
Fully Connected @ 30 with Relu
Fully Connected @ W

Figure 10: Fully Connected Autoencoder

3.8 Convolutional Autoencoder

In convolutional autoencoder architecutre, input is fed to a convolution layer with 5 different kernels of 1×4 with stride 2. Following is another layer of the same dimensions. Next layer is the encoder layer wihch represents the input in compressed format. Last layer of encoder is a fully connected layer with *neurons*. Unlike fully connected autoencoder, convolutional autoencoder is not symmetric although we tried to make it as symmetrics as possible. First layer of decoder, which is the fourth layer of the architecture, consist of convolutional transpose layer having 5 filters of size 2×1. Next layer has 2 filters of the same dimension. Lastly the output of fifth layer is flattened and treated as output of the decoder. It has the same size and dimension as input layer. Convolution transpose layer tries to reverse the effect of convolution.

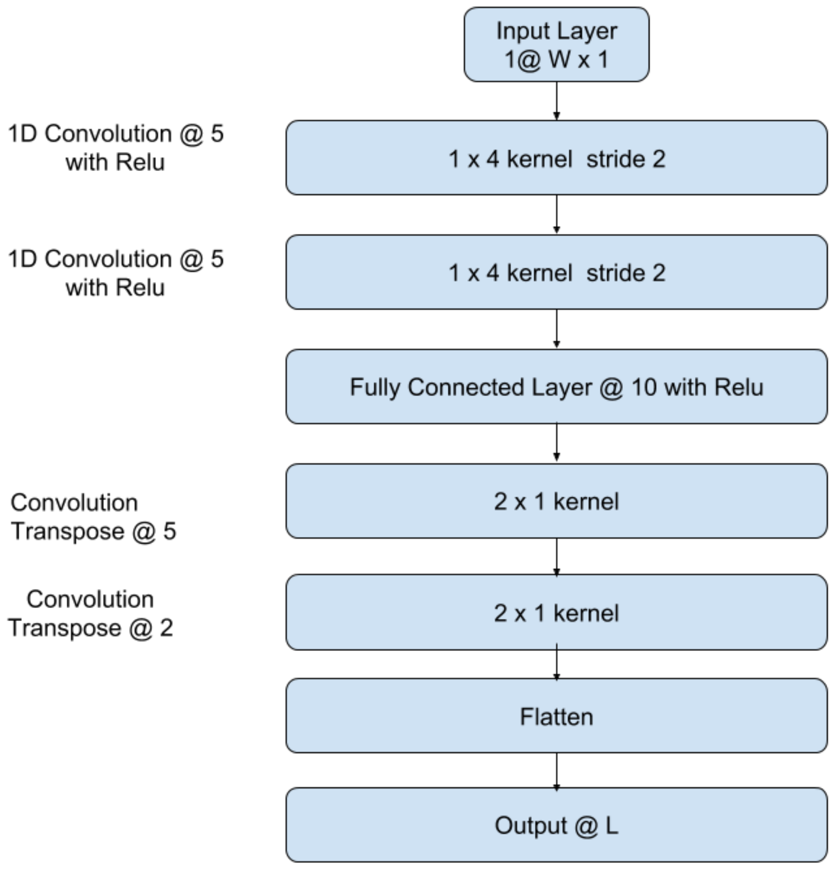

Input Layer
1@ W x 1
1D Convolution @ 5 with Relu
1 x 4 kernel stride 2
1D Convolution @ 5 with Relu
1 x 4 kernel stride 2
Fully Connected Layer @ 10 with Relu
Convolution Transpose @ 5
2 x 1 kernel
Convolution Transpose @ 2
2 x 1 kernel
Flatten
Output @ L

Figure 11: Convolutional Autoencoder

3.9 LSTM Autoencoder

LSTM based autoencoder has the intrinsic characteristics of understanding temporal relationships. The architecture of LSTM autoencoder is also simple. Uni-variate input vector of size w is fed to stacked LSTM layers whose number is equal to $w/2$. Since we have set the value of w to 50 hence the amount of LSTM cells used in encoder is 25. Final layer of encoder consist of a fully connected layer with 10 neurons. Decoder layer takes input from the fully connected layer and consist of $w/2$ number of LSTM cells. Last layer of the architecture again consists of a fully connected layer with W number of neurons as W is the length of the input data.

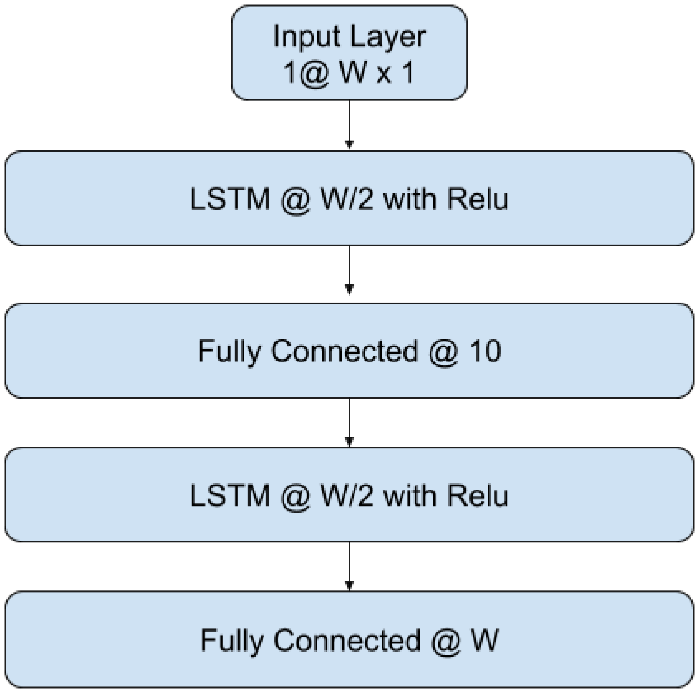


Figure 12: LSTM Autoencoder

3.10 Anomaly Score

Next step in the 6, is the calculation of anomaly score. We take the simple mean squared distance as anomaly score for all architectures. This is also the same as loss of all proposed algorithms. Anomaly score at time t can be formally defined as:

$$e_t = ||x_a - y_a||_2, where\ a = \begin{cases} t-l+1, ...t, & \text{if multistep prediction algorithm} \\ t-w, ...t, & \text{if autoencoder algorithm} \end{cases} \tag{13}$$

here $||.||_2$ means L_2 norm

$$a_t = \frac{1}{l}\sum_{i=a}^{t} e_i^2 \tag{14}$$

The definition of a in 14 is the same as in equation 13 . Such anomaly score has both merits and demerits. Since one algorithm's loss has the same definition as 14, anomaly score ends up performing one additional role. That role is the rate of netwrok upgradation. We are using a form of stochastic gradient descent for which the network weights are updated as following:

$$W_t = W_{t-1} - \eta \nabla a_t(W_{t-1}) \tag{15}$$

Since the anomlay score is computed as a mean, which is why a point anomaly will not have a huge impact on the network learning behavior. On the other side, a higher anomaly score consistently is a way to recognize concept drift thus the network will update itself swiftly as the step size is huge consistently. This was a nice logical intuition of using convergence loss as anomaly score and was originally reported by [5].

3.11 Post-processing

Post-processing is a very important part of anomlay detection. Previously calculated anomlay score is passed through another layer of processing which performs few important tasks. First and the most important task is to restrict the anomaly score value of all data to a specific range. For us this range is 0 to 1. This helps in generalization of the algorithm as it will only output values from 0 to 1, no matter what kind of input is fed. Also a common universal thresholding mechanism can be setup for all kind of data as we know that after post processing, anomaly score will always be between 0 to 1. Second task is the removal of noise and initial useless information. In the beginning, an algorithm specially when training online, takes some data to attain reasonable convergence. Obviously, in the beginning the both anomaly score and convergence loss will be high and volatile. Thus post-processing helps in filtering out these initial anomay score values. Thirdly, post-processing an help reduce the noise in anomaly score which will ultimately, help in reducing the false positive rate. Additionally, it is a stage where another model can be trained to rank anomalies.

Mainly motivated by [6] and [2], we tried two different kind of post-processing techniques. First technique is suggested by [2] and is called anomaly liklihood estimation. Second technique is a modification of simple min-max noarmalization. We perform a rolling min-max normalization. In anomaly likelihood estimation, instead of considering every single anomaly score individually, mean of a small set of anomaly scores is considered. This helps in reducing the noise as one single anomaly score can vary alot but mean of a small set has less variance. More specifically this mean and standard deviation is defined by:

$$\tilde{\mu_t} = \frac{\sum_{i=0}^{i=w-1} S_{t-i}}{w} \tag{16}$$

here w has the value of 10. Equation 16 is the mean of ten latest anomaly score values. Next step in anomaly likelihood estimation is to normalize the local mean with global mean and standard deviation. Gloabl mean and standard deviation are calculated on rolling basis and updated with the arrival of each new anomaly score. Global mean and variance are calculated using equation 17 and equation 18.

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} S_{t-i}}{W} \tag{17}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (S_{t-i} - \mu_t)^2}{W-1} \tag{18}$$

In both equations above, W is defined by a large window size and has the value of 8000. As the anomaly likelihood is a probabilistic metric defining how anomalous the current state is based on the prediction history. Next step is the normalization of local mean with global mean and standard deviation. This normalization is performed as following:

$$\mu_{normal} = \frac{\tilde{\mu_t} - \mu_t}{\sigma_t} \tag{19}$$

now only thing left is to convert this mean normalized anomaly score to anomaly liklihood. We compute the tail probability of occurance of mean normalized anomaly score μ_{normal} which is then substracted from by 1 giving us probability of a mean normalized anomaly score not occuring.

$$L_t = 1 - Q(\mu_{normal}) \tag{20}$$

Here Q is the gaussian Q-function as defined in [7]

Second method is moving mix-max normalization, which is rather simple but quite effective. We take the maximum and minimum value of the past and current data at a given point in time T and use it to normalize the incoming anomaly score value. If the new anomaly score value is greater than the past maximum value then the maximum value is updated. Similarly if the new anomaly score value is less than the past minimum value then the minimum value is updated. In the beginning of training, ususally convergence loss is higher and it does not make sense to compare it with the values calculated later on that is why first N value of the anomaly score are skipped before we start considering the maximum and minimum value. The value N is usually set around 100.

3.12 Thresholding

Now that a post-processed anomaly score is achieved, it is time to apply some ranking or thresholding function on top of anomlay score. For both all datasets a common threshold is used which is selected by NAB includes a simple hill-climbing routine for finding the optimal threshold value given the scoring rules. This algorithm is provided alongside [8].

3.13 Scoring

Based upon the characteristics of an ideal anomaly detection algorithm, a special scoring function is designed by [8] which evaluates each algorithm according to their closeness to the characteristics. We are using the same scoring function. Here we will quickly summarize the said scoring mechanism which can also be called as NAB scoring function.

Each algorithm is tested against three different profiles. Each profile provides a specific weight to true positive (TP), false positive (FP) and false negative (FN). This way an algorithm can be tested according to the requirement of the system. Similarly, according to the system's requriement an algorithm can be optimized. Better yet one can create custom profiles and train the algorithm for it. These three profiles are named as **standard**, **low FN rate** and **low FP rate**.

The value of the ture positive (TP) weight in the default standard application profile is 1.0. If a user wants to emphasize the importance of correct anomaly detection during the evaluation against benchmark, then the TP weight metric can be increased. This will increase the amount that the score is incremented every time an anomaly is detected correctly. Accordingly, if the user wants to reduce the importance of correct anomaly detection during the evaluation, then lowering the value of the TP weight will decrease the amount added to the score every time an anomaly is detected correctly.

The value of the FP weight in the standard application profile is 1.0. If a user doesn't care as much about false positives being detected, then decreasing this value will decrement the score by a lesser amount every time an anomaly is incorrectly identified. Accordingly, if the user wants to emphasize a low FP rate, then increasing the value of FP will decrement the score by a greater amount every time an anomaly is incorrectly identified.

In case of FN weight, the score is reduced by this amount once for each anomaly which is not detected at all. The value of the FN weight in the default application profile is 1.0. If a user wants to make it more important that the detector never miss a real anomaly, then increasing the FN weight will decrement the score by a greater amount when an anomaly is missed. If the user cares less about true anomalies being missed, then decreasing the value of the FN weight will decrement the score by a lesser amount when an anomaly is missed.

With the anomaly windows defined to be 10% of a given data file, true positives are limited to 10% of the data. Yet false positives can occur in 90% of the data. That is, of all the timesteps in each data file, only 10% are eligible to be true positives (within a window), while 90% can be false positives (outside a window) . Thus we scale the score contribution of false positives such that they have the same cumulative weight as true positives – i.e. the score contributions from false positives are divided by 9. This is reflected in the application profiles below.

As described above, with the combination of true positive (TP) weight, false positive (FP) weight and false negative (FN) weight, NAB creates three application profiles. The main one is **standard profile** which attributes equal weight across the scoring metrics. To be specific TP weight is set to 1, FP to 1 and FN to 0.11.

Second profile rewards a detector that has a very low FP rate; they would rather trade off missing a few true anomalies rather than getting multiple false positives.The scoring weights of are set as following:

- true positive weight is set to 1. This will give full credit to properly detected anomalies.
- false positive weight is set to 0.22. This will decrease the score for any false positive.
- false negative weight is set to 0.5. This is bigger than FP weight. Thus the algorithm will be rewarded for low false positives.

Last application profile will reward a detector that doesnot miss any true anomalies; they would rather trade off a few false positives than miss any true anomalies. The scoring weights are set as true positive weight is set to 1. false positive weight is sset to 0.055. False negative weight is set to 2.

With application profiles defined above, each algorithm is tested and benchmarked against each profile. As described earlier, the given scoring function is built around the ideal real-world anomaly detector. Summarizing the characteristics of an ideal real-world anomaly detector:

1. detects all anomalies present in the streaming data
2. detects anomalies as soon as possible
3. triggers no false alarms
4. works with real time data
5. is fully automated across all datasets To satisify the characteristics of early detection an anomaly windows. Each window represents a range of data points that is centered around a ground truth anomaly label.

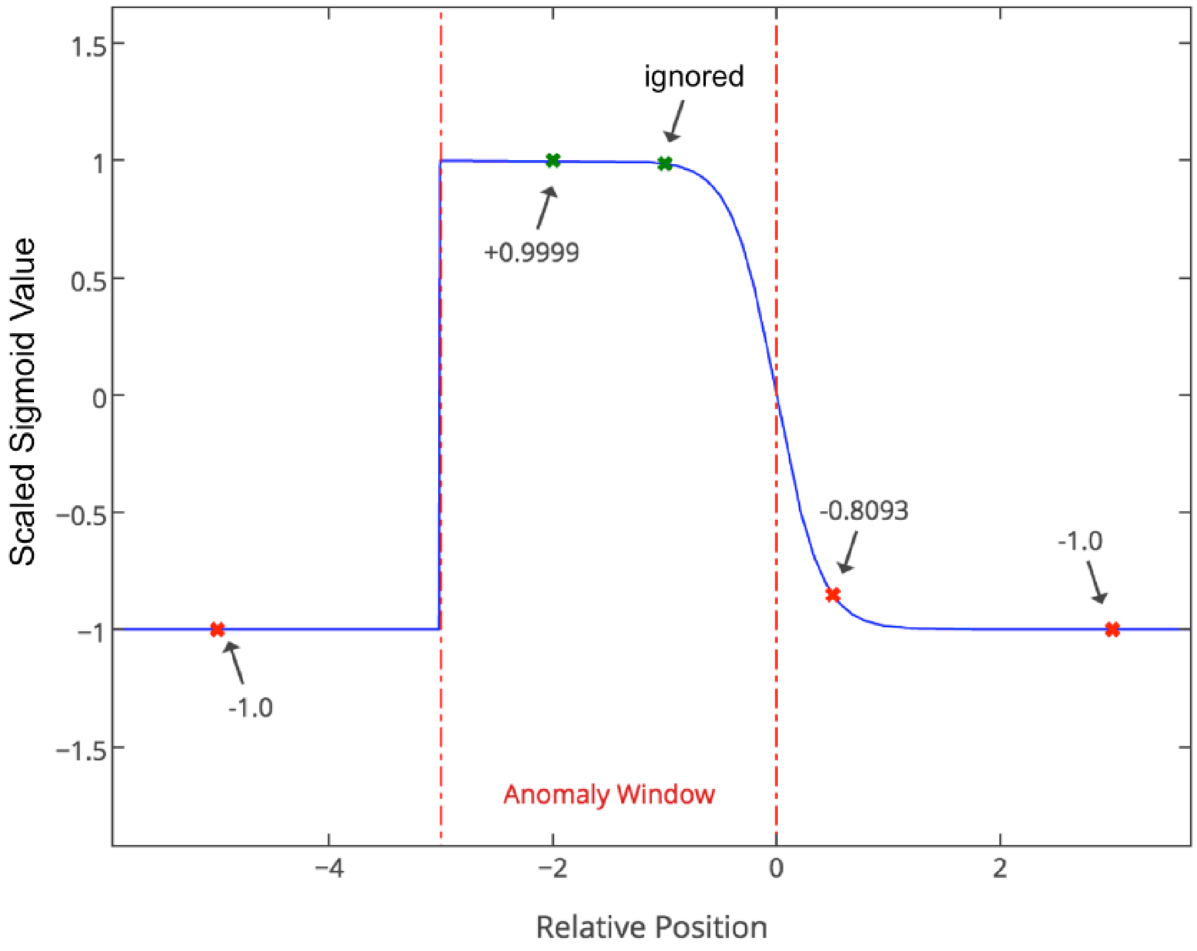


Figure 13: Scoring example for a sample anomaly window, where the values represent the scaled sigmoid function.

Figure 13 depicts a nice example of such scoring around an anomlay window. The first point in the figure is an FP preceding the anomaly window (red dashed lines) and contributes -1.0 to the score. Within the window we see two detections, and only count the earliest TP for the score. There are two FPs after the window. The first is less detrimental because it is close to the window, and the second yields -1.0 because it's too far after the window to be associated with the true anomaly. TNs make no score contributions. The scaled sigmoid values are multiplied by the relevant application profile weight. The NAB score for this example would calculate as: $-1.0A_{FP} + 0.9999A_{TP} - 0.8093A_{FP} - 1.0A_{FP}$

To promote early detection NAB defines anomaly windows. Each window represents a range of data points that is centered around a ground truth anomaly label. A scoring function uses these windows to identify and weight true positives, false positives, and false negatives. If there are multiple detections within a window, the earliest detection is given credit and counted as a true positive. Additional positive detections within the window are ignored.

The sigmoidal scoring function gives higher positive scores to true positive detections earlier in a window and negative scores to detections outside the window (i.e. the false positives). How large should the windows be? The earlier a detector can reliably identify anomalies the better, implying these windows should be as large as possible. The tradeoff with extremely large windows is that random or unreliable detections would be regularly reinforced. Using the underlying assumption that true anomalies are rare, we define anomaly window length to be 10% the length of a data file, divided by the number of anomalies in the given file. This technique allows us to provide a generous window for early detections and also allow the detector to get partial credit if detections are soon after the ground truth anomaly. 10% is a convenient number but note the exact number is not critical. NAB [8] tested a range of window sizes (between 5% and 20%) and found that, partly due to the scaled scoring function, the end score was not sensitive to this percentage. Different applications may place different emphases as to the relative importance of true positives vs. false negatives and false positives.

Now to compute NAB score for an algorithm and a given application profile we use following procedure. Let A be the application profile under consideration, with A_{TP}, A_{FP} , A_{FN} and A_{TN} the corresponding weights for true positives, false positives, false negatives and true negatives.These weights are bounded $A_{TP} \geq 0$, $A_{TN} \leq 1$ and $A_{FP} \leq -1$, $A_{FN} \leq 0$. Let D be the set of data files and let Y_d be the set of data instances detected as anomalies for datafile d. (As discussed earlier, we remove redundant detections: if an algorithm produces multiple detections within the anomaly window, we retain only the earliest one.) The number of windows with zero detections in this data file is the number of false negatives, represented by f_d. The following scaled sigmoidal scoring function defines the weight of individual detections given an anomaly window and the relative position of each detection:

$$\sigma^A(y) = (A_{TP} - A_{FP})(\frac{1}{1+e^{5y}}) - 1 \tag{21}$$

In equation (21), y is the relative position of the detection within the anomaly window. The parameters of the equation are set such that the right end of the window evaluates to $\sigma(y = 0.0) = 0$ and it yields a max and min of A_{TP} and A_{FP}, respectively. Every detection outside the window is counted as a false positive and given a scaled negative score relative to the preceding window. The function is designed such that detections slightly after the window contribute less negative scores than detections well after the window. Missing a window completely is counted as a false negative and assigned a score of A_{FN}. The raw score for a data file is the sum of the scores from individual detections plus the impact of missing any windows:

$$S_d^A = (\sum_{y \in Y_d} \sigma^A(y)) + A_{FN} f_d \tag{22}$$

Equation (22), accumulates the weighted score for each true positive and false positive, and detriments the total score with a weighted count of all the false negatives. The benchmark raw score for a given algorithm is simply the sum of the raw scores over all the data files in the corpus:

$$S^A = \sum_{d \in D} S_d^A \tag{23}$$

The final reported score is a normalized NAB score computed as follows:

$$S_{NAB}^A = 100 \times \frac{S^A - S_{null}^A}{S_{perfect}^A - S_{null}^A} \tag{24}$$

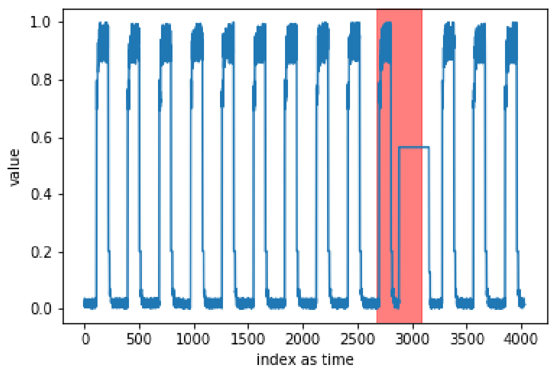
Here we scale the score based on the raw scores of a "perfect" detector (one that outputs all true positives and no false positives) and a "null" detector (one that outputs no anomaly detections). It follows from equation (24) that the maximum (normalized) score a detector can achieve on NAB is 100, and an algorithm making no detections will score 0.

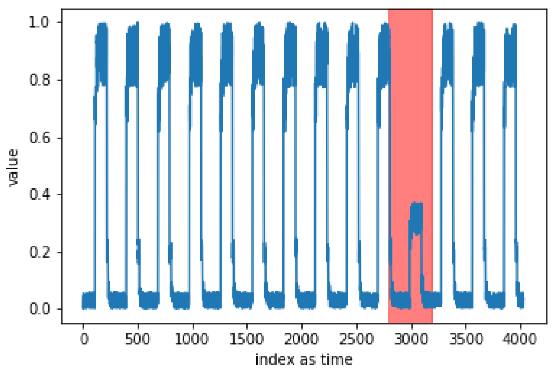
4 Empirical Formulation and Experiments

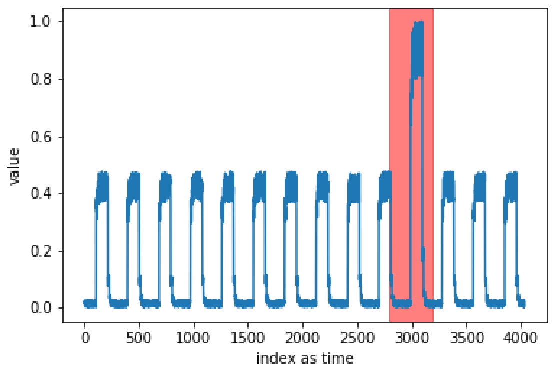
4.1 Dataset

To experiment with proposed method and campare them with state-of-the-art results Numenta Anomaly Benchmark (NAB) **??** was used. Numenta is a benchmark specifically designed for online time series anomaly detection. It is first of its kind as NAB tries to cover all type of anomalies. Also they have come up with a custom scoring function to map the characteristics of an ideal online anomaly detector as defined in the previous section.

Using NAB provides us with many benefits. First benefit is that it is a benchmark dataset which 58 various univariate time series, each with timestamp as index and value. Length of each time series ranges from 1000 to 22000 data points. This provides us with enough variability in data. These 58 time series are from six different categories. These categories include artificially generated time series, times series gathered from advertisement industry related data such as click through rate (CTR), AWS usage metrics, server traffic data, tweets volume time series and known cause anomaly which has the a solid reason for the labeled anomaly.


(a) Artificial with sudden flat-middle


(b) Artificial with sudden jumps-down


(c) Artificial with sudden jumps-up

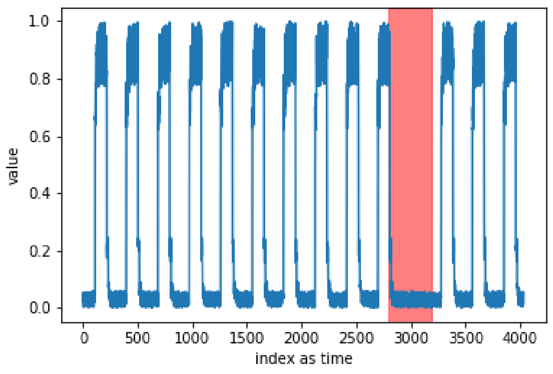
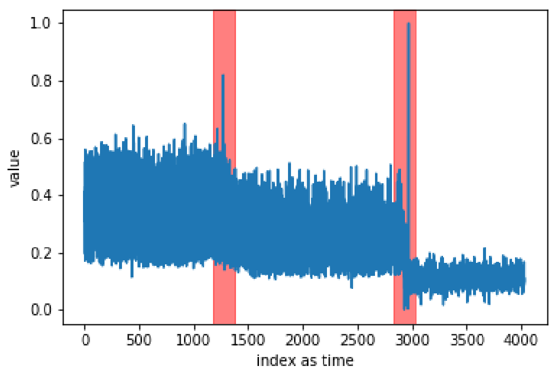

(d) Artificial with sudden flat

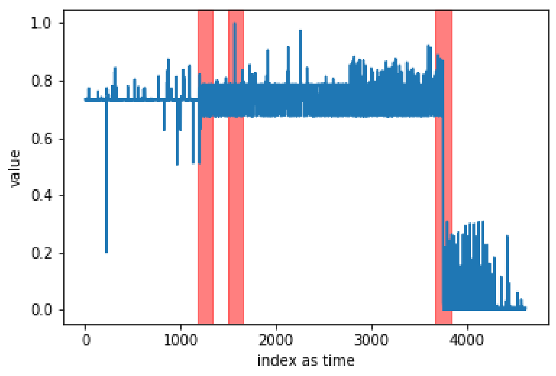
Figure 14: Examples of artifical anomaly data set

The diversity of these categories provide the second benefit. With the diversity almost all type of anomalies are covered hence each algorithm can be tested against each type. Artificial datasets further contains time series with anomalies and without anomalies. Datasets without anomalies contain only normal patterns and no anomaly thus any predicted anomaly will be a false positive here. This way one can test false positive prediction tendency of an algorithm.

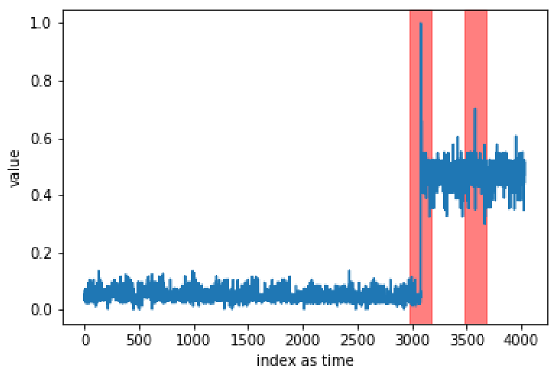
In NAB, concept drift is given a special importance that is why many datasets contains concept drift. These datasets are mostly real life ones. Example of these are following:



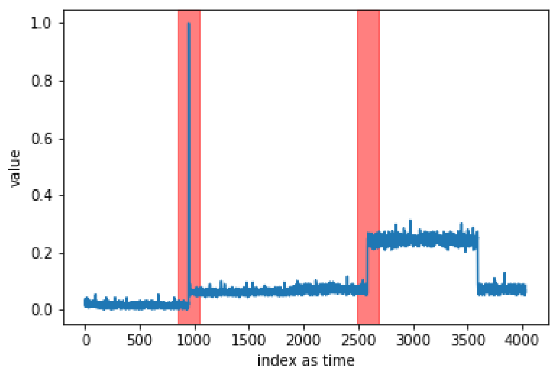
(a) Artificial with sudden flat-middle



(b) Artificial with sudden jumps-down



(c) Artificial with sudden jumps-up



(d) Artificial with sudden flat

Figure 15: Examples of Concept drift in AWS data sets

Similarly almost all kind of anomalies exist in the dataset. For example for point anomaly we can see that in following.

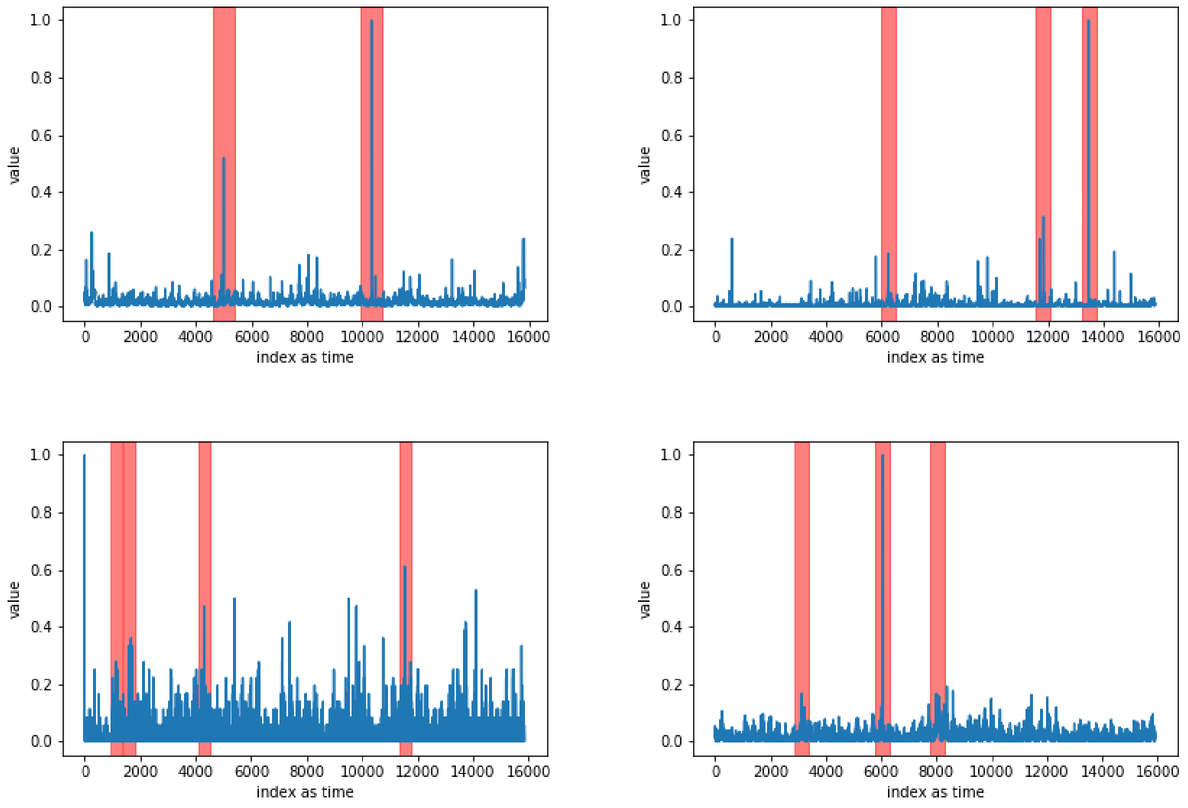


Figure 16: Examples of point anomalies from twitter volume dataset

Another important benefit is the availability of labeled data. Other then the seven real life time series, for which the cause of anomaly is known, all others were labeled diligently using a labelling process.

Three different human labelers labeled the dataset. Each labeler would read each file using all available tools specially visualization and with some external human expertise in the field, label the dataset. The timestamp at the start of each anomaly is noted. This way there is a data file describing raw labels by each labeler. Since there were three human labelers hence three different raw label files were created. Each raw label file contains file name as key and the values contain the start timestamps of anomaly. Next up all these files are merged into one using following merging strategy:

1. For each given data file (i.e. the key), the start timestamps from all the raw labels are collected into buckets with other start timestamps within a certain range. That is, timestamps within a narrow range of one another are bucketed together because they (very likely) identify the same anomaly. Differences in anomaly start times amongst labelers can be attributed to human error. The size of the time range is set as 1/10 the size of an anomaly window (defined in step 3 below), both preceding and following the

given timestamp. The rationale is straightforward: following the underlying assumption anomalies are rare, if multiple anomaly labels fall within the same window, they identify the same sequence of anomalous data.

2. If a given bucket contains more timestamps (raw labels) than the agreement threshold, it is classified as a true anomaly. The standard threshold is 0.50, so a bucket must contain timestamps from 50% or more of the labelers in order to become a true anomaly label .

3. Each true anomaly bucket is then merged into a single timestamp based on frequency. That is, the timestamp which appears most often within a bucket is chosen to represent the entire bucket. Ties are determined by which timestamp occurs first. The resulting timestamps are the ground truth labels marking the start of the true anomalies.

The combined labels are then converted into anomaly windows, indicated by a beginning and ending timestamp. The motivation behind anomaly windows is twofold. First, anomalous data often occurs over time, rather than at a single point, and thus defining anomaly windows improves the NAB scoring efficacy. Second, anomaly windows allow the DUT to not be penalized during scoring if the DUT anomaly detections are slightly before or after the ground truth. We want these windows to be large enough to allow early detection of anomalies, but not so large as to artificially boost the score of inaccurate detectors. That is, the windows give the benefit of the doubt to the detector, but not so much that false positives would count as true positives. The anomaly windows are calculated using the following algorithm:

1. The total amount of window length in a single data file was chosen to be 10% of the data file length. For instance, if a data file contains 4000 data points, then the total amount of relaxation shared by all anomaly windows in the data file is 0.1*4000, or 400 data points. The 10% parameter was validated by qualitatively inspecting the dataset, and experimenting with a range of values showed no significant affect on the resulting scores, likely due to the scaled sigmoid scoring function (described later).

2. The total window length for a data file is then divided by the total number of ground truth anomalies in the data file, and the resulting number of data points makeup each anomaly window, 1/2 before the start of the window, and 1/2 at the end of the window. For instance, if the data file above has four ground truth anomalies, then each anomaly window is 400/4, or 100 data points – i.e. 50 data points both before and after the truth anomaly data point.

3. If two ground truth anomalies are in close proximity such that they overlap, the anomalies are combined into one large window. The rationale being if multiple anomalies fall within the same window length, we assume they identify the same anomalous data.

Thus ground labels of anomalies are obtained. A detailed meta information about these datasets is given in appendix 7.

4.2 Experimental setup

With the scoring and evaluation mechanism setup, we will now share the experimental setup and how the online training and prediction was performed on the datasets. We used following steps to simulate a near real-time streaming data we developed a streaming function which will load each data file individually and then feed only one data point at a time. Thus for each algorithm, only single data point will be fed at a time. Next autoencoder based algorithms training and prediction is done in following steps:

1. Define the window size w of the input data which will be the actual input size to the algorithm.

2. Define algorithm for the specific input window size.

3. Load $x_{0:w-1}$ data points from the dataset, meaning load first w data points of the data and feed it to algorithm.

4. Algorithm parameters will be randomly initialized in the first step. It will then predict for the given input.

5. Algorithm will go through **one** back-propagation step. The reconstruction loss will considered as anomaly score since reconstruction loss is also mean squared error.

6. The reconstruction loss will be fed to post-processing function. Depending upon the type of post-processing one of the following step will be taken:

    - In case of anomaly likelihood estimation, local mean, gloabl mean and global variance will be calculatd as stated in equation 16,17 and 18 respectively. After performing normalization, a gaussian tail probability is calculated and anomaly likelihood is estimated.
    - In case of min-max normalization, global minimum and maximum values are initiated by zero. Then current anomaly score is compared with global minimum and maximum values and global values are updated accordingly such as:

    $$global_minimum = min(global_minimum, a_t)global_maximum = max(global_maximum, a_t) \tag{25}$$

    where a_t is the anomaly score at time t. Next step is to perform simple min-max normalization although with a small practical change. Update equation of min-max normalization is:

    $$z = \frac{x - min(x) + 1}{max(x) - min(x) + 1} \tag{26}$$

7. output of sixth step, will providea post-processed anomaly score which will be stored in a vector of anomaly scores.

8. Now x_t is already processed and we are ready to take on x_{t+1}. In the next step data of $x_{1:w}$ will be loaded and steps from $4th$ till $8th$ will be repeated till all of the dataset is processed.

For an autoencoder algorithm, all datasets are processed through the steps described above. The output of post-processed anomaly score is stored for thresholding and scoring later on. On the other hand for multiple step prediction algorithms, following steps are taken:

1. Define the window size w of the input data which will be the actual input size to the algorithm. Also define the size of multiplestep ahead prediction L.

2. Define algorithm for the specific input window size.

3. Load $x_{0:w-L-1}$ data points from the dataset, meaning load first $w - L$ data points of the data and feed it to algorithm.

4. Similarly load $x_{L:t}$ datapoints.

5. Algorithm parameters will be randomly initialized in the first step. It will then predict for the given input which can be compared againt the input $x_{L:t}$.

6. Algorithm will go through **one** back-propagation step. The prediction loss will be considered as anomaly score since prediction loss is also mean squared error.

7. The anomaly score a_t will be fed to post-processing function. Depending upon the type of post-processing one of the steps will be taken as defined in the $6th$ step of above routine.

8. Output of sixth step, will providea post-processed anomaly score which will be stored in a vector of anomaly scores.

9. Now x_t is already processed and we are ready to take on x_{t+1}. In the next step data of $x_{1:w-L}$ will be loaded and steps from $4th$ till $8th$ will be repeated till all of the dataset is processed.

As an output we will have post-processed anomlay score for each dataset against each algorithm. These anomaly scores are passed through [8] thresholding function which will output a threshold for all data files. Then again [8] scoring function will use the threshold and anomaly scores provided to calculate the final evaluation score for the algorithm as described in the scoring section above. This evaluation can be used as benchmark against other algorithms.

5 Results

Before sharing and discussing results, we can prove that the algorithms converges when trained online using just one iteration. An example of such convergence is given in the figure 17

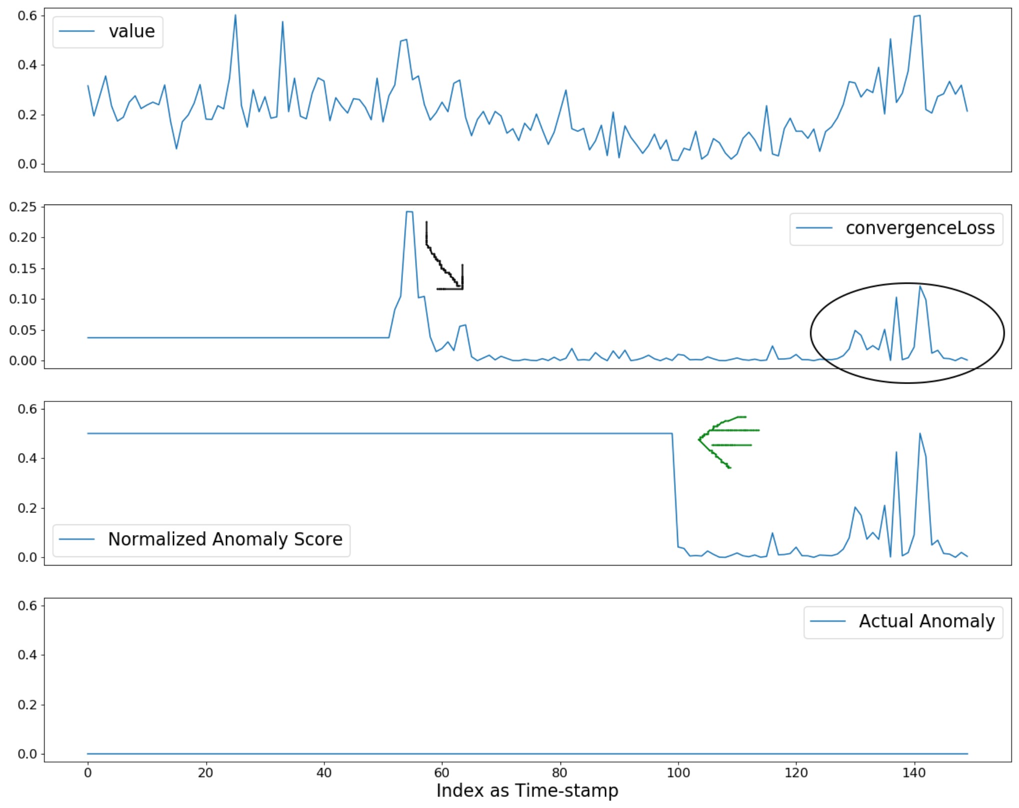


Figure 17: An example of convergence of LSTM multistep prediction algorithm with multiplestep size 1 and window size 50, being trained online.

Figure 17, displays few important variables of our anomlay detection algorithms. All graphs are being shown for first 150 time stamps. For simplicity timestamps are converted into indices. In other words, index 0 and index 1 are the alternative of $2018-09-01$ and $2018-09-02$ respectively, for example. Top graph displays a normalized actual data stream from which we need to detect if every incoming point is anomaly or not. In the second graph, convergence loss is displayed. As the input windows size is 50, hence there are first fifty points of convergence los are simply set to 0.05. As first iteration of network takes first fifty points thus true training starts from $50th$ data point. Soon after start of training from $50th$ index onward, algorithm starts converging as shown by the arrow sign. This trend can be seen from $60th$ index till $130th$. Then there is a sudden increase in loss as expected as shown by an ellipse. This is because there was a sudden changeing the data stream trend plus we are using only one iteration per time stamp. This makes our algorithm relatively sensitive to the input data, especially in the beginning when the algorithm has not learned enough. With time such sudden noisy losses are seen less.

Moving on to the third grpah from top in the figure 17, one can see post-processed anomaly score. Here min-max normalization was used. Another imporant aspect to be noted is that the normalization value remains constant for first 100 time indices as shown pointed out by green arrow. The reason is, as described already, out of bound convergence loss in the beginning which can cause the whole incoming anomlay scores to be of less value. For example in a set of $5, 3, 2, 4$ after min-max normalization vector $1, 0.49, 0.33, 0.67$ will be generated. If this was an anomaly score post-processing, we will never be able to detect any anomaly in the upcoming data as the upcoming anomaly score will seldom be greater than or closer to 5. Since we know that in the beginning the anomaly score generated by the model will always be huge as it just started learning, it is necessary to ignore first few values to not skew our post-processing. We set this threshold to 100.

Last graph at the botton in figure 17 shows that, there was no labelled anomaly in the first 150 index timestamps.

5.1 post-processing type

5.1.1 anomaly likelihood

5.1.2 min-max normalization

5.2 results tables

5.2.1 Difference of results in different algorithms

5.2.2 Change in window size in the same algorithm

5.2.3 Change in multiple-step in the same algorithm

First step was to

Algorithm	low FN rate	low FP rate	Standard
numenta	74.3202	63.1168	70.1011
contextOSE	73.1817	66.9771	69.8996
htmjava	70.4241	53.2569	65.5499
numentaTM	69.18506	56.6653	64.5534
knncad	64.8123	43.4085	57.9943
relativeEntropy	58.8423	47.5984	54.6427
randomCutForest	59.7488	38.3608	51.7171
twitterADVec	53.5011	33.6101	47.062
predictionMultiStepLSTMMultiStep1Window50	59.3125	43.5927	43.2791
windowedGaussian	47.41	20.868	39.6495
predictionMultiStepLSTMMultiStep5Window20	41.6451	35.4072	39.1918
predictionMultiStepLSTMMultiStep5Window20	40.8274	34.6116	38.3963
predictionMultiStepLSTMMultiStep1Window200	40.2705	31.1383	36.0246
skyline	44.481	27.0812	35.687
predictionMultiStepNNMultiStep4Window50	29.1037	25.6366	24.4352
predictionMultiStepCNNMultiStep5Window30	21.228	19.6188	20.9474
predictionMultiStepCNNMultiStep5Window20	21.5211	18.142	19.6384
predictionMultiStepNNMultiStep5Window30	19.4988	18.0512	19.5921
random	27.201	5.76368	17.6554
expose	26.9209	3.19093	16.4367

6 Conclusion and Discussion

6.1 Conclusion

6.2 Problems faced during experiments

6.3 Insights

6.4 Future work

7 Appendix A: Meta of NAB data sets

length	max_val	min_val	data_type	file_name
15833	1258	0	realTweets	Twitter_volume_FB.csv
15858	36	0	realTweets	Twitter_volume_PFE.csv
15902	13479	0	realTweets	Twitter_volume_AAPL.csv
15851	2241	0	realTweets	Twitter_volume_KO.csv
15831	1673	0	realTweets	Twitter_volume_AMZN.csv
15893	139	0	realTweets	Twitter_volume_IBM.csv
15866	231	0	realTweets	Twitter_volume_UPS.csv
15853	50	0	realTweets	Twitter_volume_CVS.csv
15902	209	0	realTweets	Twitter_volume_CRM.csv
15842	465	0	realTweets	Twitter_volume_GOOG.csv
10320	39197	8	realKnownCause	nyc_taxi.csv
7267	86.22321261	57.45840559	realKnownCause	ambient_temperature_system_failure.
18050	100	11.529	realKnownCause	cpu_utilization_asg_misconfiguration.
22695	108.5105428	2.084721206	realKnownCause	machine_temperature_system_failure.
1882	0.895012153	0	realKnownCause	rogue_agent_key_hold.csv
5315	288.207531	0	realKnownCause	rogue_agent_key_updown.csv
4032	99.248	22.864	realKnownCause	ec2_request_latency_system_failure.cs
4032	20	0	artificialWithAnomaly	art_increase_spike_density.csv
4032	87.99838184	18.00203996	artificialWithAnomaly	art_daily_jumpsdown.csv
4032	87.95834813	-21.99878884	artificialWithAnomaly	art_daily_flatmiddle.csv
4032	164.9474805	18.00100982	artificialWithAnomaly	art_daily_jumpsup.csv
4032	3.224814931	0	artificialWithAnomaly	art_load_balancer_spikes.csv
4032	87.97331353	18.00050056	artificialWithAnomaly	art_daily_nojump.csv
2500	5059	9	realTraffic	TravelTime_387.csv
1127	90	1	realTraffic	speed_7578.csv
2380	22.28	0	realTraffic	occupancy_6005.csv
2500	109	20	realTraffic	speed_6005.csv
2495	77	11	realTraffic	speed_t4013.csv
2500	43.06	0	realTraffic	occupancy_t4013.csv
2162	5578	22	realTraffic	TravelTime_451.csv
4032	79.99996928	20	artificialNoAnomaly	art_daily_no_noise.csv
4032	45	45	artificialNoAnomaly	art_flatline.csv
4032	87.97612833	18.00096402	artificialNoAnomaly	art_daily_small_noise.csv
4032	80	20	artificialNoAnomaly	art_daily_perfect_square_wave.csv
4032	18.99640526	8.000581096	artificialNoAnomaly	art_noisy.csv
1624	0.2265979381	0.02684303351	realAdExchange	exchange-2_cpc_results.csv
1643	16.43819968	0.1215201222	realAdExchange	exchange-4_cpm_results.csv
1643	3.126851852	0.02388370424	realAdExchange	exchange-4_cpc_results.csv
1538	5.497540003	0.320650021	realAdExchange	exchange-3_cpm_results.csv

length	max_val	min_val	data_type	file_name
1624	1.051442475	0.0003850049458	realAdExchange	exchange-2_cpm_results.csv
1538	1.034	0.03889884763	realAdExchange	exchange-3_cpc_results.csv
1243	61519397	789781	realAWSCloudwatch	iio_us-east-1_i-a2eb1cd9_NetworkIn.c
4032	99.898	0.064	realAWSCloudwatch	ec2_cpu_utilization_77c1ca.csv
4032	863964000	0	realAWSCloudwatch	ec2_disk_write_bytes_c0d644.csv
4032	2.344	0.066	realAWSCloudwatch	ec2_cpu_utilization_24ae8d.csv
4032	1.602	0.062	realAWSCloudwatch	ec2_cpu_utilization_c6585a.csv
4032	25.1033	5.19	realAWSCloudwatch	rds_cpu_utilization_cc0c53.csv
4032	656	1	realAWSCloudwatch	elb_request_count_8c0756.csv
4032	68.092	34.766	realAWSCloudwatch	ec2_cpu_utilization_5f5533.csv
4032	99.742	2.464	realAWSCloudwatch	ec2_cpu_utilization_ac20cd.csv
4032	76.23	12.628	realAWSCloudwatch	rds_cpu_utilization_e47b3b.csv
4032	2.656	1.604	realAWSCloudwatch	ec2_cpu_utilization_53ea38.csv
4730	547457000	0	realAWSCloudwatch	ec2_disk_write_bytes_1ef3de.csv
4621	45.6229	0	realAWSCloudwatch	grok_asg_anomaly.csv
4032	99.118	18.7225	realAWSCloudwatch	ec2_cpu_utilization_825cc2.csv
4032	245126000	38516.6	realAWSCloudwatch	ec2_network_in_257a54.csv
4730	8285420	42	realAWSCloudwatch	ec2_network_in_5abac7.csv
4032	99.668	1.8	realAWSCloudwatch	ec2_cpu_utilization_fe7f93.csv

8 Experiment Infrastructure

8.1 Experiment Management using MLflow

8.2 Parallel execution using Docker

9 Best practices

Following steps were taken to maximize the efficiency and speed of research:

1. Use version control to track the code and share between different devices.
2. Separate code from data. This will keep the code base small and easy to debug.
3. Separate input data,working data and output data.
   - **Input Data:** Input data-set that never change. For my case it is NAB and other external datasets.
   - **Working Data:** nothing for now.
   - **Output Data:** Results and threshold profiles in my case.
4. Separate options from parameter. This is important:

- Options specify how your algorithm should run. For example data path, working directory and result directory path, epochs, learning rate and so on.
- parameters are the result of training data. it includes the score and hyper-parameters.

9.1 Moving from jupyterlab to pycharm

While working with jupyterlab notebook following routine was followed: 1- Load data with sample function 2- Write an algorithm 3- Test the results 4- Write general executeable .py file. 5- Get results on server

Since we needed to track change on two different places, it was becoming harder to track the bugs and improve on efficiency. That's why pycharm was selected to create executeable files and test algorithms at the same time.

10 Reference Usage

11 References

[1] A. karpathy, available at https://medium.com/@karpathy/software-2-0-a64152b37c35. This article was main motivation structure the infrastructure of research. Further readings in this article was used to finalize the research infrastructure.

[2] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[3] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Dec 2015, pp. 38–44.

[4] *23rd European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015*, 2015. [Online]. Available: https://www.elen.ucl.ac.be/esann/proceedings/papers.php?ann=2015

[5] S. Ranu, N. Ganguly, R. Ramakrishnan, S. Sarawagi, and S. Roy, Eds., *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, COMAD/CODS 2018, Goa, India, January 11-13, 2018.* ACM, 2018. [Online]. Available: https://doi.org/10.1145/3152494

[6] D. T. Shipmon, J. M. Gurevitch, P. M. Piselli, and S. T. Edwards, "Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data," *CoRR*, vol. abs/1708.03665, 2017. [Online]. Available: http://arxiv.org/abs/1708.03665

[7] G. K. Karagiannidis and A. S. Lioumpas, "An improved approximation for the gaussian q-function," *IEEE Communications Letters*, vol. 11, no. 8, pp. 644–646, 2007. [Online]. Available: https://doi.org/10.1109/LCOMM.2007.070470

[8] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms - the numenta anomaly benchmark," *CoRR*, vol. abs/1510.03336, 2015. [Online]. Available: http://arxiv.org/abs/1510.03336

[9] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One model to learn them all," *CoRR*, vol. abs/1706.05137, 2017. [Online]. Available: http://arxiv.org/abs/1706.05137

[10] A. Eslami, available at http://arkitus.com/patterns-for-research-in-machine-learning. This article was main motivation structure the infrastructure of research. Further readings in this article was used to finalize the research infrastructure.

[11] P. M. Ferreira, Ed., *3rd IFAC International Conference on Intelligent Control and Automation Science, ICONS 2013, Sichuan, Chengdu, China, September 2-4, 2013.* International Federation of Automatic Control, 2013. [Online]. Available: http://www.ifac-papersonline.net/Intelligent_Control_and_Automation_Science/3rd_IFAC_International_Conference_on_Intelligent_Control_and_Automation_Science__2013_/index.html

[12] N. Chawla and W. Wang, Eds., *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017.* SIAM, 2017. [Online]. Available: https://doi.org/10.1137/1.9781611974973

[13] M. Balcan and K. Q. Weinberger, Eds., *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016. [Online]. Available: http://jmlr.org/proceedings/papers/v48/