Real-Time Deep Learning-Based Anomaly Detection Approach for Multivariate Data Streams with Apache Flink

Tae Wook Ha, Jung Mo Kang, and Myoung Ho Kim(✉)

Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea
{twha,jmkang,mhkim}@dbserver.kaist.ac.kr

Abstract. For detecting anomalies which are unexpected behaviors in complex systems, deep learning-based anomaly detection algorithms for multivariate time series have gained a lot of attention recently. While many anomaly detection algorithms have been widely proposed, there has been no work on how to perform these detection algorithms for multivariate data streams with a stream processing framework. To address this issue, we present a real-time deep learning-based anomaly detection approach for multivariate data streams with Apache Flink. We train a LSTM encoder-decoder model to reconstruct a multivariate input sequence and develop a detection algorithm that uses reconstruction error between the input sequence and the reconstructed sequence. We show that our anomaly detection algorithm can provide promising performance on a real-world dataset. Then, we develop a Flink program by implementing three operators which process and transform multivariate data streams in a specific order. The Flink program outputs anomaly detection results in real time, making system experts can easily receive notices of critical issues and resolve the issues by appropriate actions to maintain the health of the systems.

Keywords: Multivariate data streams · Anomaly detection · LSTM encoder-decoder · Stream processing · Apache Flink

1 Introduction

In real-world applications, multivariate data are continuously generated by monitoring complex systems such as readings of sensing devices (e.g. pressure and moisture) in power plants or multiple metrics (e.g. CPU load, network usage) in server machines. Detecting anomalies, unexpected behaviors of the systems, in multivariate data streams is a critical task in managing these systems such that system managers can receive notices of critical issues and resolve the issues by providing appropriate actions to maintain the health of the systems. Traditionally, a system expert defines thresholds of normal behaviors for every measurements and then if a measurement exceeds the defined threshold, it is considered as an anomaly that means the system does not behaves normally. Because the size and complexity of the systems increase, however, the number of sensors

M. Bakaev et al. (Eds.): ICWE 2021 Workshops, CCIS 1508, pp. 39–49, 2022.
https://doi.org/10.1007/978-3-030-92231-3_4

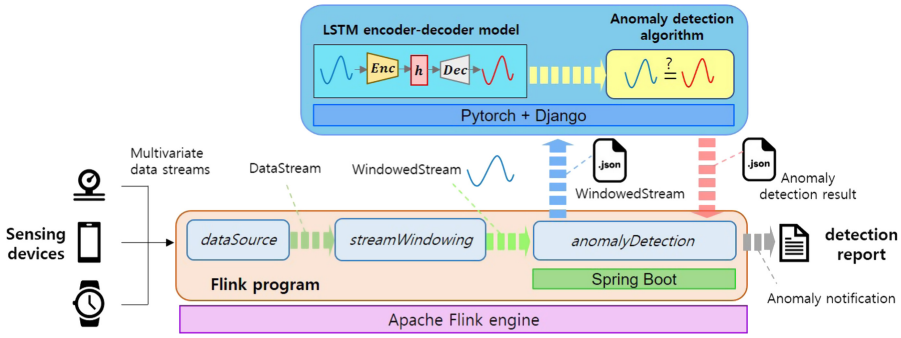

Fig. 1. Overall architecture of an anomaly detection of multivariate data streams with Apache Flink

has dramatically increased making system experts hard to define thresholds of normal behaviors.

To solve this issue, many anomaly detection algorithms have been proposed in multivariate time series data analytic fields for several decades [1]. Several studies [12–14] propose supervised learning methods to detect anomalies, but the lack of labeled anomalous data for model training makes these methods infeasible. As a result, unsupervised anomaly detection methods to learn normal behaviors of the systems have received a lot of attention. Early methods find anomalies by using machine learning models such as k-nearest neighbor [15], clustering [16], one-class svm [17], or a time series prediction model [18]. However, the traditional learning models are not efficient to infer complex dependencies of multivariate time series data. A lot of recent anomaly detection algorithms use deep neural network models such as LSTM encoder-decoder model [2], a deep autoencoder with a Gaussian mixture model [10], LSTM networks with a variational autoencoder [11], LSTM or convolutional LSTM networks for prediction [7,8], bi-directional generative adversarial networks [9] and convolutional recursive encoder-decoder model [3], a stochastic recurrent neural network [4], a temporal hierarchical one-class network [5] and an autoencoder model with adversarial training [6]. While these deep learning-based anomaly detection algorithms have been proposed, as far as we know, there has been no work on how to perform these detection algorithms for multivariate data streams with a stream processing framework.

In this work, we present a real-time deep learning-based anomaly detection approach for multivariate data streams with a stream processing framework, Apache Flink. We first train a LSTM encoder-decoder model in [2] to reconstruct a multivariate sequence and develop an anomaly detection algorithm that uses the reconstruction error between the input sequence and the reconstructed sequence. And then, we develop a Flink program to process multivariate data streams in real time. Flink is one of popular even-driven stream processing frameworks for stateful computations over unbounded data streams. For a user to easily develop a program, Flink provides a programming model

that is a directed acyclic graph where each node is an operator that defines a functionality for incoming data streams. With implementing operators, we can specify the order of transformations of the data streams. Figure 1 shows the overall architecture of our Flink program consisting of three operators: *dataSource*, *streamWindowing*, and *anomalyDetection*. *dataSource* operator receives raw data streams and send them to DataStream objects where DataStream is a basic type for handling a data stream in Flink. *streamWindowing* then generates a sequence containing multiple consecutive data from the DataStream object. *anomalyDetection* feeds the sequence into a trained LSTM encoder-decoder model and notifies anomalies by receiving detection results from the detection algorithm.

The rest of this paper is organized as follows. We first discuss an anomaly detection method for a multivariate sequence in Sect. 2. And then we describe implementation details of Flink operators in Sect. 3 and conclude this paper in Sect. 4.

2 Anomaly Detection for a Multivariate Sequence

We focus on a multivariate sequence $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, ..., \mathbf{x}_T\}$ of length T where each data point $\mathbf{x}_t \in R^m$ is a m-dimensional vector of readings for m variables at a time step t. Since most real-world applications have lack of labeled anomalous data, we train the LSTM encoder-decoder model with normal sequences obtained by collecting multiple normal sequences or taking a sliding window of length T over a large time series data. After training the LSTM encoder-decoder model, the reconstruction error of each data point in an input sequence is used to compute an anomaly score for the point. A higher anomaly score indicates the point being anomalous, and if the input sequence contains enough anomalous points, we notice that the input sequence is anomaly.

2.1 LSTM Encoder-Decoder Model for Reconstruction

The LSTM encoder-decoder model consists of a LSTM encoder and a LSTM decoder as shown in Fig. 2. Given X, the LSTM encoder updates the hidden state $\mathbf{h}_t^E$ at a time step t for each $t \in \{1, 2, ..., T\}$ where $\mathbf{h}_t^E \in R^d$, d is the number of units in a LSTM network. Note that the final hidden state of the encoder $\mathbf{h}_t^E$ is learnt as a vector representation of X and the LSTM decoder uses $\mathbf{h}_t^E$ as the initial state $\mathbf{h}_t^D$ to reconstruct X in reverse order [2]. A LSTM network in the decoder outputs a reconstructed data point $\mathbf{x'}_t$ at t by computing $\mathbf{x'}_t = \mathbf{W^T h}_t^D + \mathbf{b}$ where $\mathbf{W}$ is a weight matrix of a linear layer in the LSTM network and $\mathbf{b} \in R^m$ is a bias vector. Then, $\mathbf{x'}_t$ is used to update the next hidden state $\mathbf{h}_{t-1}^D$. The objective function for model training is $L = \sum_{X \in S_X} \sum_{t=1}^{T} \|\mathbf{x}_t - \mathbf{x'}_t\|^2$ where S_X is a set of normal sequences.

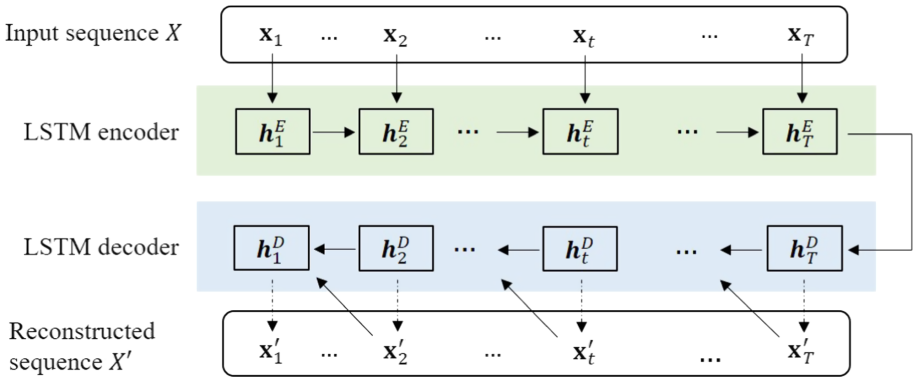

Fig. 2. Inference step of a LSTM encoder-decoder model for an input sequence X to output reconstructed sequence X'

2.2 Computing Anomaly Scores for Detection

Given the input sequence X and the reconstructed sequence X', the output of the LSTM encoder-decoder model, the error vector is obtained by $\mathbf{e}_t = |\mathbf{x}_t - \mathbf{x'}_t|$ for all time steps in X. Then, we compute the anomaly score of each data point $a_t = (\mathbf{e}_t - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{e}_t - \boldsymbol{\mu})$ where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are a mean vector and a covariance matrix of a normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ respectively. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are estimated by error vectors $\mathbf{e}_t$ for all data points $\mathbf{x}_t$ in a validation set.

We determine whether the input sequence X is anomaly based on the anomaly scores a_t for all time steps. After we compute an anomaly score a_t at time step t, if $a_t > \tau$, the data point $\mathbf{x}_t$ can be detected as "an anomalous data point", otherwise "a normal data point". If the number of anomalous data points in X is larger than δ, we notice that X is an anomalous sequence. (otherwise, X is noticed by a normal sequence). Note that if there is a very few incorrectly reconstructed data points in X, we consider this is because there are some noises in X making the LSTM encoder-decoder model cannot reconstruct the data points, or a very few anomalies occur but they are not severe to the system [3]. We set the values of τ and δ that maximize $F1 = 2 \times (P \times R)/(P + R)$ score on the validation sequences where P is a precision score and R is a recall score.

2.3 Performance Evaluation of Anomaly Detection

Dataset. To evaluate the performance the anomaly detection described in Sect. 2, we conduct an experiment on a real-world dataset: SMD (Server Machine Dataset). SMD is a new 5-week-long dataset collected by [4] from 28 server machines in a large Internet company. It contains observations measured for every 1 min and each observation has 38 metrics such as TCP active opens, memory usage, CPU load, Disk write, etc. For each machine, data are divided into two subsets of equal size: the first half is a training set and the second half is a testing set. In our experiment, we use 80% of training set for model training and use 20% of training set for model validation. Anomalies in the testing set

are labeled by domain experts based on incident reports. Detailed information of SMD dataset can be seen in Table 1. The meanings of columns in Table 1 are as follows. "machine id" is an id of each machine, and "train/validation/test time steps" is the number of observations in the training/validation/testing set. "anomaly time steps" is the number of anomalies in the testing set and "anomaly ratio" is a ratio of anomalies over the testing set. "anomalous periods" is the number of periods of consecutive anomalies.

Table 1. SMD dataset information

Machine id	Train time steps	Validation time steps	Test time steps	Anomaly time steps	Anomaly ratio (%)	Anomalous periods
1-1	22,784	5,695	28,479	2,694	9.46	8
1-2	18,956	4,738	23,694	542	2.29	10
1-3	18,962	4,740	23,703	817	3.45	12
1-4	18,965	4,741	23,707	720	3.04	12
1-5	18,964	4,741	23,706	100	0.42	7
1-6	18,951	4,737	23,689	3,708	15.65	30
1-7	18,958	4,739	23,697	2,398	10.12	13
1-8	18,959	4,739	23,699	763	3.22	20
2-1	18,955	4,738	23,694	1,170	4.94	13
2-2	18,960	4,739	23,700	2,833	11.95	11
2-3	18,951	4,737	23,689	269	1.14	10
2-4	18,952	4,737	23,689	1,694	7.15	20
2-5	18,951	4,737	23,689	980	4.14	21
2-6	22,995	5,748	28,743	424	1.48	8
2-7	18,957	4,739	23,696	417	1.76	20
2-8	18,962	4,740	23,703	161	0.68	1
2-9	22,978	5,744	28,722	1,755	6.11	10
3-1	22,960	5,740	28,700	308	1.07	4
3-2	18,962	4,740	23,693	1,109	4.68	10
3-3	18,963	4,740	28,696	632	2.20	26
3-4	18,950	4,737	23,703	977	4.12	8
3-5	18,952	4,738	23,703	426	1.80	11
3-6	22,981	5,745	23,687	1,194	5.04	11
3-7	22,964	5,741	23,691	434	1.83	5
3-8	22,963	5,740	28,726	1,371	4.77	6
3-9	22,971	5,742	28,705	303	1.06	4
3-10	18,954	4,738	28,704	1,047	3.65	13
3-11	22,956	5,739	28,713	198	0.69	3

Experimental Settings and Evaluation Metrics. We employ Pytorch to implement a LSTM encoder-decoder model for each machine, and train each model on a server with Intel(R) Core(TM) i7-10700k CPU 3.80 GHz, 128 GB memory, and a NVIDIA RTX 3090 graphic card. We empirically set hyper-parameters of the LSTM encoder-decoder model in our experiments as follows. The length of a sliding window, for taking an input sequence, is set to 100 and the overlapping length is set to 10. The number of hidden units in LSTM is fixed to 200. We set the batch size to 5 for model training, and run for 200 epochs with early stopping. We utilize Adam optimizer with a learning rate of 10^{-3} for stochastic gradient descent.

To evaluate the performance of the anomaly detection using LSTM encoder-decoder model, we use three metrics, *i.e.* Precision, Recall, and F1 scores. For an input sequence, it is considered as an anomaly if it contains anomalous data points more than δ. We set the threshold values of τ and δ which maximize the F1 scores over the input sequence taken from the validation set of each machine. Precision and recall scores are then calculated based on these threshold values.

Evaluation Results. Table 2 shows the evaluation results of the anomaly detection algorithm for each machine. With our discussion with experts in some real-world applications, we focus on the high recall scores among the detection results. We can see that the detection algorithm in our work has recall scores higher than 0.90 showing it can be effectively used to detect anomalies in some cases (e.g. 1-1, 1-6, 3-1, 3-4, 3-5, and 3-11). However, we can also see that precision scores lower than 0.1 causing low F1 scores in some machine (e.g. 2-8, 3-5, 3-9, 3-11) because of lack capabilities of the LSTM encoder-decoder model for reconstruction. Thus, this motivated us to improve the performance of the detection algorithm in future work, by searching the optimized hyper-parameters with

Table 2. Performance results on SMD dataset

Machine id	Precision	Recall	F1	Machine id	Precision	Recall	F1
1-1	0.2985	**0.9742**	0.4570	2-7	0.6889	0.2616	0.3792
1-2	0.8269	0.5548	0.6641	2-8	0.0107	0.5769	0.2021
1-3	0.1389	0.6414	0.2284	2-9	0.2790	0.8986	0.4258
1-4	0.1551	0.4974	0.2365	3-1	0.0283	**0.9437**	0.0549
1-5	0.4545	0.6329	0.5291	3-2	0.1291	0.7598	0.2206
1-6	0.3209	**0.9684**	0.4820	3-3	0.4504	0.3814	0.4130
1-7	0.7799	0.7588	0.7692	3-4	0.0833	**0.9943**	0.1536
1-8	0.2864	0.4758	0.3575	3-5	0.0936	**0.9603**	0.1706
2-1	0.6018	0.5407	0.5696	3-6	0.7815	0.8087	0.7949
2-2	0.4511	0.5550	0.4977	3-7	0.0826	0.7128	0.1481
2-3	0.7789	0.5873	0.6697	3-8	0.2756	0.7143	0.3977
2-4	0.2539	0.6230	0.3608	3-9	0.0832	0.8551	0.1517
2-5	0.6602	0.4518	0.5365	3-10	0.2434	0.4681	0.3202
2-6	0.1608	0.8364	0.2698	3-11	0.0504	**0.9773**	0.0959

additional experiments or developing a new model for reconstruction to capture the complex dependencies in multivariate time series data.

3 Implementation Details of Flink Operators

3.1 *dataSource* Operator

dataSource operator reads a raw sensor data and transmit it to the output stream `ds`. Since our program reads values from multiple sensors, we let `ds` have a data type <Tuple> containing a sensor id, a value, and a timestamp. We implement our customized source function `CustomSrcFunc` for reading sensor values by inheriting `RichSourceFunction` provided by Apache Flink. `env` gets the environmental information of Flink program and registers the `CustomSrcFunc` to execute from source s every time interval t. After `env` calls `execution()` method to start the program, for every time t, `CustomSrcFunc` reads a value from s and creates `tuple` containing an id and a value of s, and a timestamp. Then, `tuple` is sent to the output stream `ds` by calling `collect()` method in `ds`.

Algorithm 1: *dataSource* operator

```
1  s ← data source, t ← time interval
2  ds ← DataStream<Tuple>
3  CustomSrcFunc ← RichSourceFunction <Tuple>
4  env ← getExecutionEnvironment()
5  env.register(CunstomSrcFunc(s, t))
6  if env calls execution() then
7      while every time t do
8          CustomSrcFunc reads a value from s and creates tuple containing an id
           and a value of s
9          ds.collect(tuple)
10     end
11 end
```

3.2 *streamWindowing* Operator

streamingWindowing operator collects multiple `tuple` from multiple stream `ds` to build a window with shape $[l, dim]$ and send it to the output stream of windows `ws`. We first define a parameter value `trigger` that is the number of `tuple` to collect and initialize `tempwin` window by calling `createGlobalWindow()` method in Apache Flink. When `ds` collects a `tuple`, `ds` puts `tuple` into `tempwin`. If the number of `tuple` in `tempwin` meets `trigger`, `tempwin` calls the `sortAggregate()` method to sort multiple `tuple` via timestamps in `tuple`. After that, `tempwin` is sent to the output window stream `ws` by calling `collect()` method in `ws`.

3.3 *anomalyDetection* Operator

anomalyDetection operator takes charge of two roles. One is to connect two heterogeneous platforms and the other is to detect anomalous sequences. Our two preceding operators consist of Java but we employ Pytorch to implement a LSTM encoder-decoder model. We need to make it possible to communicate between two different platforms so that we send `tempwin` from `ws` to the LSTM encoder-decoder model for reconstruction. Thus, we adopt a local client-server architecture for communication. We describe the details of our local client-server architecture as follows.

Algorithm 2: *streamWindowing* operator

```
1  trigger ← dim * l
2  tempwin ← createGlobalWindow()
3  ws ← WindowedDataStream<Tuple>
4  while true do
5      if ds collects tuple then
6          ds puts tuple to tempwin
7          if the numbers of tuples in tempwin == trigger then
8              tempwin ← tempwin.sortAggregate()
9              ws.collect(tempwin)
10         end
11     end
12 end
```

The Details of Local Client-Server Architecture. The local client is implemented on a spring boot framework and it behaves asynchronous and event-driven way. On the other hand, the local server is implemented on a Django rest framework. Once receiving `tempwin` from the windowed stream `ws`, the local client makes a JSON file containing the data of `tempwin`. The local client then posts the JSON file to the local server, subscribes to the response from the local server, and waits for the next `tempwin` from `ws`. When the local server receives the JSON file, it calls the `serialize()` to extract `tempwin` in the file. Since `tempwin` already has a tensor of a fixed size (*i.e.*, $[l, dim]$), it copies the value of the JSON file to the tensor. The tensor is defined by $\texttt{Tensor}_{in}$ corresponding to the input sequence. The $\texttt{Tensor}_{in}$ is sent to the LSTM encoder-decoder model and the model outputs the reconstructed tensor $\texttt{Tensor}_{re}$. The local server conducts the anomaly detection algorithm to detect anomalies by the procedure of the detection algorithm, as described in Sect. 2.2. After that, `Webserver.create(`*localhost*`)` calls `response(`$result_d$`)` to response the detection result to `Webclient.create(`*localhost*`)` which reports the detection result.

Algorithm 3: *anomalyDetection* operator

```
client_local ← Webclient.create(localhost)
server_local ← Webserver.create(localhost)
client_local calls post(tempwin)
if tempwin is sent then
    client_local subscribes response from server_local and waits for the next tempwin
    from ws
    Tensor_in ← server_local calls serialize(tempwin)
    Tensor_re ← LSTM encoder-decoder(Tensor_in)
    cnt ← 0 to count anomalous data points
    forall the time step t in length T do
        x_t ← Tensor_in (t)
        x'_t ← Tensor_re (t)
        e_t = |x_t − x'_t|
        a_t = (e_t − μ)^T Σ^{-1} (e_t − μ)
        if a_t > τ then
            cnt ← cnt +1
        end
    end
    if cnt > δ then
        result_d ← tempwin is an anomalous sequence
    else
        result_d ← tempwin is a normal sequence
    end
    server_local calls response(result_d)
    if result_d is replied then
        client_local gets result_d and reports result_d
    end
end
```

4 Conclusion

In this paper, we presented a real-time deep learning-based anomaly detection approach of multivariate data streams with Apache Flink. While many deep learning-based anomaly detection algorithms have been proposed in the literature, there has not been no work on how to perform these detection algorithms for multivariate data streams with a stream processing framework. In order to address this issue, we first train a LSTM encoder-decoder model to reconstruct an input sequence and develop a detection algorithm that uses anomaly scores based on errors between the input sequence and the reconstructed sequence. We show that our detection algorithm have promising performance by experimental results on a real-world dataset. Then, we develop a Flink program by implementing three operators which process and transform the multivariate data streams in a specific order. The developed Flink program can provide anomaly notifications in real time, making system experts can easily identify critical issues and maintain the health of the complex systems by providing appropriate solutions.

Acknowledgement. This work was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2020-0-01795) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation), and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1A2C1004032).

References

1. Chalapathy, R., Chawla, S.: Deep learning for anomaly detection: a survey. arXiv preprint arXiv:1901.03407 (2019)
2. Malhotra, P., et al.: LSTM-based encoder-decoder for multi-sensor anomaly detection. arXiv preprint arXiv:1607.00148 (2016)
3. Zhang, C., et al.: A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01 (2019)
4. Su, Y., et al.: Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
5. Shen, L., Li, Z., Kwok, J.: Timeseries anomaly detection using temporal hierarchical one-class network. In: Advances in Neural Information Processing Systems, vol. 33 (2020)
6. Audibert, J., et al.: USAD: unsupervised anomaly detection on multivariate time series. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2020)
7. Hundman, K., et al.: Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2018)
8. Tariq, S., et al.: Detecting anomalies in space using multivariate convolutional LSTM with mixtures of probabilistic PCA. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
9. Zenati, H., et al.: Adversarially learned anomaly detection. In: 2018 IEEE International Conference on Data Mining (ICDM). IEEE (2018)
10. Zong, B., et al.: Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In: International Conference on Learning Representations (2018)
11. Park, D., Hoshi, Y., Kemp, C.C.: A multimodal anomaly detector for robot-assisted feeding using an LSTM-Based variational autoencoder. IEEE Robot. Autom. Lett. **3**(3), 1544–1551 (2018)
12. Park, D., et al.: A multimodal execution monitor with anomaly classification for robot-assisted feeding. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE (2017)
13. Rodriguez, A., et al.: Failure detection in assembly: force signature analysis. In: 2010 IEEE International Conference on Automation Science and Engineering. IEEE (2010)
14. Görnitz, N., et al.: Toward supervised anomaly detection. J. Artif. Intell. Res. **46**, 235–262 (2013)

15. Hautamaki, V., Karkkainen, I., Franti, P.: Outlier detection using k-nearest neighbour graph. In: Proceedings of the 17th International Conference on Pattern Recognition. ICPR 2004, vol. 3. IEEE (2004)
16. He, Z., Xiaofei, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recogn. Lett. **24**(9–10), 1641–1650 (2003)
17. Manevitz, L.M., Yousef, M.: One-class SVMs for document classification. J. Mach. Learn. Res. **2**, 139–154 (2001)
18. Brockwell, P.J., Davis, R.A., Fienberg, S.E.: Time Series: Theory and Methods. SSS. Springer Science & Business Media, New York (1991). https://doi.org/10.1007/978-1-4419-0320-4