The **otsad** Package: Online Time-Series Anomaly Detectors

Alaiñe Iturria[1,2], Jacinto Carrasco[2], Francisco Herrera[2], Santi Charramendieta[1], and Karmele Intxausti[1]

[1]IK4-Ikerlan, Big Data Architectures Team, Paseo J.Mª Arizmediarrieta, 2, 20500 Arrasate-Mondragon. Spain
[2]Department of Computer Science and Artificial Intelligence, University of Granada, Granada, 18071, Spain

Abstract

Anomaly detection gives valuable information that can be used to solve malfunctions and prevent future problems. Although this field has been widely studied, due to the continuous evolution of technology, new challenges arise that require further improvement and evolution of anomaly detection techniques. In this paper, we present the **ostad** package that implements seven of the most recent detection algorithms capable of dealing with various challenges, such as online and non-stationary univariate time-series anomaly detection.

1 Introduction

Anomaly detection consists of the identification of patterns in the data that differ from expected behavior and is a relevant task in many domains such as fault detection in the manufacturing industry, intrusion detection in cybersecurity and fraud detection in banks. Anomaly detection is an extensive field that has been studied for years. Chandola et al. [8], Hodge et al. [12] and Zhang et al. [23] provide a comprehensive overview of anomaly detection techniques.

Due to the evolution of new technologies, the amount of data is increasing and is collected faster. For this reason, detection techniques must face new challenges such as an increase in the amount of data and online processing capacity. Unlike classic techniques, online time-series anomaly detection techniques do not have the complete data set to work with, and time must be considered. Another major challenge with the introduction of online time-series processing is that time-series can be stationary and non-stationary.

In recent years, most of the work has focused on the evolution of evolving prediction models and sliding window-based techniques [9]. Evolving techniques are models in which parameters or components are modified as new data arrives to capture normal data trends in a better way. On the other hand, techniques based on sliding windows are commonly used to improve distance-based techniques. The distance calculation is an expensive method and proportional to the number of observations to be considered. The use of sliding windows allows reducing the set of observations to be considered but also maintaining the most recent subset of data.

Some works [1][3], suggest that prediction-based algorithms are suitable to adapt to online time-series anomaly detection since only the past data is required during the training phase. Moreover, real-time time series prediction is an independent domain whose research is currently more advanced than TSOD. However, the continuous change of the training data set, the non-stationarity of the data, and the lack of the entire data set make it challenging to adapt the online prediction algorithms into an outlier detection. Hence, little research has been done in this domain [2][22].

When adapting prediction algorithms to anomaly detection, two challenges must be addressed. The first challenge is the online anomaly scoring based on the prediction error. The techniques used in offline processing algorithms rely on the entire data set to compute the score from the prediction errors, and very few are proposed for online processing [2][5]. The second challenge is the streaming input data normalization. Standardization or normalization of the data becomes an essential requirement for some online prediction algorithms, such as those based on neural networks [15][18]. However, none of them address online normalization, and they assume that data is already normalized. This is not a realistic assumption, as, in a real scenario where data arrives in real-time, the entry data set

is not available. Hence, the statistics needed for normalization, e.g., minimum, maximum, mean, or deviation, may change over time, making these algorithms not suitable for real online learning tasks.

Although anomaly detection is a highly studied field, there is little theory and open-source software able to address these new challenges. For python, we have found a few algorithms in GitHub of *Numenta Anomaly Benchmark (NAB)* [14] including **Numenta HTM**, **CAD-OSE**, **KNN-CAD**, and others. For R, there are few CRAN packages. The first and most popular is the **tsoutliers** package. There is also the **qicharts** package that implements basic control chart algorithms. One of the main disadvantages of these packages is that the implemented algorithms are not suitable to work online. In addition to the above, there are few other packages available for online time-series anomaly detection: **SmartSifter** and **EnergyOnlineCPM**.

This paper presents a novel R package that includes up-to-date and powerful anomaly detection algorithms for univariate time series. Named as **ostad**, it aims to provide algorithms that cover different current needs such as online processing and the ability to work in stationary and non-stationary environments. This package intends to address both evolving and sliding window-based techniques that are gaining strength, including algorithms of both types. Moreover, to help increase the number of proposed algorithms and further progress in this field, this work aims to provide a novel framework that allows an easy adaptation of any online prediction algorithm into an online TSOD. The framework covers these two crucial aspects: streaming normalization and outlier scoring. It contains two main components. The first component is oriented to online data normalization or standardization, while the second focuses on online anomaly scoring based on prediction errors. Both components implement several online normalization and outlier scoring methods already available in state-of-the-art and new proposals.

The rest of the paper is organized as follows. The following section introduces the content of the **otsad** package, while section 3 describes in detail how to use implemented detectors and reduce false positives. Finally, we conclude the paper with some conclusions and future lines for continuing the work, section 4.

2 The otsad package

This section introduces the first R package that develops a set of up-to-date and effective online anomaly detectors for univariate time series. The **otsad** package implements eight anomaly detection algorithms along with other functionalities and contents that can be interesting to adapt online time-series forecasting algorithms and perform the best results.

For an easier understanding of this section, we divided it into five subsections. In the first subsection, we described each implemented algorithms. A novel detector evaluation technique is presented in section 2.2. Then, we introduce included datasets and the function to visualize the results in section 2.3. After that, section 2.4 describes a simple but effective technique for reducing false positives. Finally, in section 2.5 a framework for easily adapting any online prediction model to outlier detection is introduced.

2.1 Anomaly detection algorithms

This package implements and documents the set of detectors listed below. The first three algorithms belong to evolving-based techniques, and the last four belong to window-based ones. From a stationary environment perspective, the first two algorithms can be used with stationary data, while the other five are suitable for non-stationary data. Table 1 shows the most important characteristics of each algorithm.

- *PEWMA* or *Probabilistic reasoning for streaming anomaly detection* [7]. This algorithm is a probabilistic method of EWMA which dynamically adjusts the parameters based on the probability of the given observation. This method produces dynamic, data-driven anomaly thresholds robust to abrupt transient changes yet quickly adjust to long-term distributional shifts.
- *SD-EWMA* or *Shift-Detection based on EWMA* [19]. This algorithm is a novel method for covariate shift-detection tests using univariate time-series. It uses an exponentially weighted moving average (EWMA) model based control chart to detect the covariate shift-point in stationary time-series. SD-EWMA algorithm works in an online mode.
- *TSSD-EWMA* or *Two-Stage Shift-Detection based on EWMA* [19]. This algorithm is a novel method for covariate shift-detection tests based on a two-stage structure for univariate time-series. It uses an exponentially weighted moving average (EWMA) model-based control chart to detect the covariate shift-point in non-stationary time-series. The algorithm works in two phases and processes the stream in an online mode. In the first phase, it applies SD-EWMA. The second phase checks the anomalies' authenticity using the Kolmogorov-Simirnov test to reduce false alarms.

Online anomaly detectors	Features	
	Stationarity	Technique
PEWMA	Stationary	Evolving
SD-EWMA	Stationary	Evolving
TSSD-EWMA	Non-stationary	Evolving
KNN-CAD	Non-stationary	Window-based
KNN-LDCD	Non-stationary	Window-based
CAD-OSE	Non-stationary	Window-based
EORELM-AD	Non-stationary	Window-based

Table 1: Features of the algorithms

- *KNN-CAD* or *Conformal k-NN Anomaly Detector* [6]. This algorithm is a model-free anomaly detection method for univariate time-series. It adapts itself to non-stationarity in the data stream and provides probabilistic abnormality scores based on the conformal prediction paradigm.
- *KNN-LDCD* or *KNN - Lazy Drifting Conformal Detector* [13]. This algorithm is a variant of the KNN-CAD algorithm. The differences between both rely on the dissimilarity measure calculation and conformity measure calculation methods. The KNN-CAD and KNN-LDCD algorithms use both distance-based and statistical techniques to determine the degree of anomaly.
- *CAD-OSE* or *Contextual Anomaly Detector* [20]. This algorithm discretizes the search space and creates contexts with aggrupations of observations. When a new instance arrives, the algorithm searches in the previous storage contexts if the instance has been seen and creates a new context if it has not. The creation of new and storage contexts determines if the instance is an anomaly.
- *EORELM-AD* or *Ensemble based Online Recurrent Extreme Learning Machine Anomaly Detector*. This is an ensemble of the OR-ELM prediction algorithm. The OR-ELM [18] is an improvement of the popular Online Sequential Extreme Learning Machine (OS-ELM)[6] that trains single-hidden layer feedforward recurrent neural networks in an online manner. EORELM-AD combines the resultant predictions of different instances of the OR-ELM algorithm for posterior anomaly scoring and detection based on prediction errors.

Each of these algorithms was implemented to work in two different scenarios. On the one hand, classical processing is used when the complete data set (train and test) is available. On the other hand, incremental (or online) processing is used when the entire dataset is unavailable. This approach allows calculating the abnormality of new observation(s) with the parameters updated in the last run performed.

2.2 Detector measurement technique

The **otsad** package implements the method used in NAB [14] to measure the detector. This method considers the time elapsed until the anomalies are detected.

This metric uses time windows to determine the label of the detected anomaly, i.e., True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). The label is determined by centering a window on the real anomaly. The detected anomaly is labeled as TP when it falls inside the window and as FP when it is outside. An FN label is considered when there are no detected anomalies inside the window. TN labels are not considered. The window size is calculated as the division between 10% of the number of observations and real anomalies in the time series. The package includes three functions to perform these tasks. On the one hand, **GetWindowLength** and **GetWindowsLimits** get the size and the start and end limits of the window after being focused on the real anomalies. On the other hand, **GetLabels** gets the TP, FP, TN, and FN labels.

This measurement technique also allows setting a weight for each of these labels. These weights make it possible to penalize FN and FP and reward early detections. In this way, the FN and FP will get a negative score, and the early detections will get a higher score than the late ones, as is shown in Figure 1. NAB [14] proposes to use the weights given in Table 2. There are three different profiles *Standard*, *Reward low FP rate* and *Reward low FN rate*. Each one of these profiles penalizes more or less the score of FP and FN. Finally, the total score is calculated as the sum of the scores assigned to each detected anomaly and the cumulative scores of missed anomalies. This score is then scaled using the Max-Min normalization as in Equation 1. The maximum value used is the score of a perfect detector, i.e., detects all anomalies with maximum score. The minimum value is the score of a null detector, i.e., one that does not detect any anomaly (and no FP). The final score is in the interval $(-\infty, 100]$.

Table 2: Label weights per profile

Label	Profile		
	Standard	**Reward low FP rate**	**Reward low FN**
ATP	1.0	1.0	1.0
AFP	-0.11	-0.22	-0.11
ATN	1.0	1.0	1.0
AFN	-1.0	-1.0	-2.0

For these latest objectives, we have extended the package capabilities in three more functions. First one is **GetDetectorScore**, that calculates the detector score without normalizing. The second one is **GetNullAndPerfectScores** function to obtain the scores of the perfect and null detectors for the dataset. The third one is the **NormalizeScore** function which allows normalizing detector scores.

$$Score_{final} = 100\frac{Score_{detector} - Score_{null}}{Score_{perfect} - Score_{null}} \tag{1}$$

This package incorporates an additional function to allow the user to reproduce the results of the benchmarking. Named as **GetNumTrainingValues**, this function enables obtaining the number of instances used as a training set in NAB [14]. The number of training set values is calculated as 15% of 5000. and for smaller datasets of 5000 instances as 15% of the dataset instances.

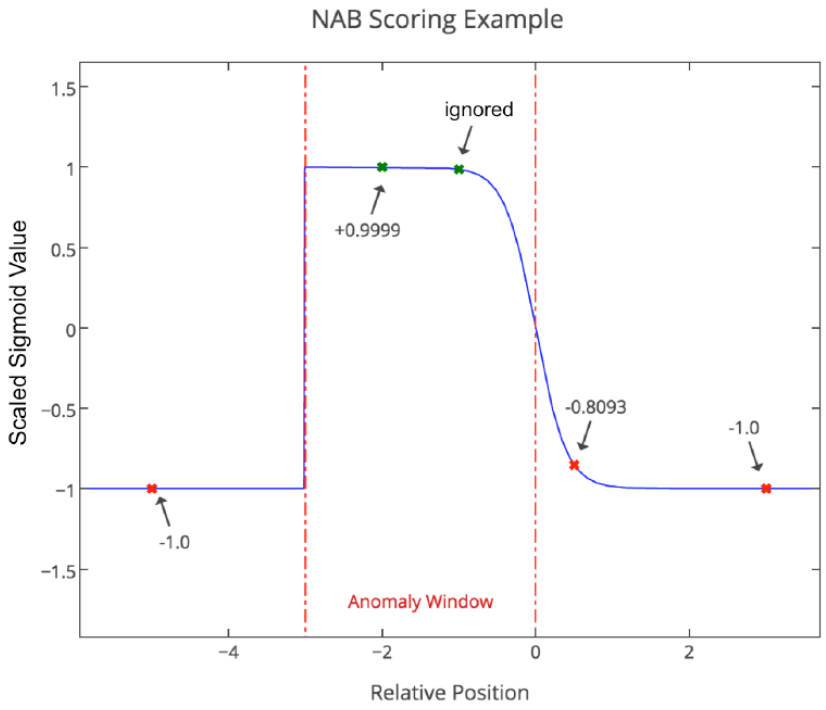


Figure 1: Scoring example for a sample anomaly window, where the values represent the scaled sigmoid function. The first point is an FP preceding the anomaly window (red dashed lines) and contributes -1.0 to the score. Within the window, we see two detections, and only count the earliest TP for the score. 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 $-1.0A_{FP}+0.9999A_{TP}-0.8093A_{FP}-1.0A_{FP}$. With the standard application profile, this would result in a total score of 0.6909 [14]

.

2.3 Data sets and function for displaying the results

Otsad includes 51 of the 58 labeled one-dimensional time-series from different fields available in the NAB [14] repository. Each of the datasets included in **otsad** is composed of three columns: the *timestamp* column, the *value* column and a third column called *is.real.anomaly* containing the labeled truth anomalies.

Our package also includes a function called **PlotDetections** to display the detections results on a chart. This function, by default, returns an interactive graph, but it also has a choice to get a ggplot object, and it could be edited.

2.4 False positive reduction technique

Some algorithms already included techniques to reduce false positives. These can be only applied to the above algorithms. For this reason, we introduce and implement our own algorithm, *ReduceAnomalies*, to reduce false positives, ensuring that it can be applied in all the algorithms. This algorithm is inspired by the real-life situation where a time-lapse exists between an alarm being triggered, and corrective action is taken. In other words, the minimum time lapse between the first and the second alarm. Our algorithm uses the number of processed data points between two detected anomalies to reduce false positives. When the first anomaly x_t is detected a new window of length w is created, with x_{t+1} as the starter point and x_{t+w} as the endpoint. For each newly detected anomaly, its relative position compared to the window is evaluated. Detected anomalies inside the window are excluded. If a detected anomaly is outside the window, it is considered a real anomaly, and the new window is calculated in the same way as mentioned above.

2.5 From prediction to anomaly detection framework

Prediction based outlier detectors are very popular in batch learning, however, adapting them to streaming processing is not an easy task. Besides, several streaming prediction algorithms need normalized data sets to work well. Nevertheless, in most cases, normalization is not considered along with the algorithm, and batch normalization techniques are used in their experiments. For those reasons, we describe our framework to adapt any prediction algorithm into an outlier detector in the following.

The framework comprises two main components, one for online data normalization and the other for streaming anomaly scoring based on prediction error. The procedure to adapt an online prediction model into outlier detection using this framework is shown in Figure 2 and it is as follows. First, if the prediction model requires it, the current data point is normalized incrementally and then used to train and predict the expected value using the chosen prediction model. After that, to compute the outlierness, the prediction error is computed and passed to the outlier scoring function. Finally, a predefined threshold is used to determine if the current sample is or is not an outlier.

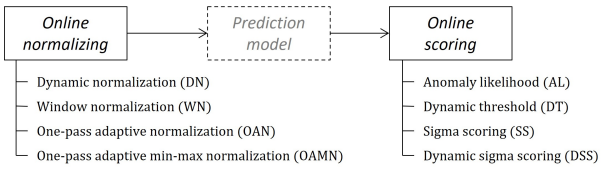


Figure 2: Procedure to adapt online prediction models to outlier detection using the proposed framework.

In Figure 2, in addition to the procedure for transforming a prediction algorithm into an anomaly detection algorithm, the normalization and scoring functions contained in each component are also shown.

2.5.1 Online normalization component

- *DN* or *Dynamic nomalization*. This method is introduced by Bollegala [4]. In there, the classical z-score standardization Eq. (2) is applied using dynamically calculated mean Eq. (3) and deviation Eq. (4).

$$x' = \frac{x - \mu}{\sigma} \tag{2}$$

$$\mu_t = \mu_{t-1} + \frac{x_t - \mu_{t-1}}{t} \tag{3}$$

$$
\begin{aligned}
s_t &= s_{t-1} + (x_t - \mu_{t-1})(x_t - \mu_t) \\
\sigma_t &= \sqrt{s_t/(t-1)}
\end{aligned}
\tag{4}
$$

- *WN* or *Window nomalization*. This is the traditional approach, used by Lin and Keogh [16] and Haykin [11], that consists of dividing time series into sliding windows W first, and then normalizing or standardizing them according to the statistical properties (μ and σ) inferred from the last window W. In one-pass normalization, a sliding window W is updated with the last n data instances whenever a new data x_t arrives, i.e., $W = \{x_{t-n+1}, ..., x_t\}$. Then, considering the statics of the sliding window W, the whole W (or only the current data instance x_t) is normalized.
- *OAN* or *One-pass adaptive nomalization*. This is our adaptation for one-pass learning of the adaptive normalization introduced by Ogasawara et al. [17]. This method is specifically designed to normalize non-stationary heteroscedastic (non-uniform volatility) time series and is used for neural network learning. Furthermore, it is important to note that the original method is not fully incremental since it is used to train different neural networks (NN) offline. However, it presents some incremental properties, which we exploit to its application in an online environment.
- *OAMN* or *One-pass adaptive min-max nomalization*. This is a one-pass adaptation of the adaptive min-max normalization for Big Data stream proposed by Gupta and Hewett [10]. The original proposal was designed for online chunk-by-chunk (batches of data) processing, where disjoint (non-overlapping) fixed size sliding windows W_i are used. In this case, one-pass overlapping windows are used.

2.5.2 Online outlier scoring component

- *AL* or *Anomaly likelihood*. This is a novel incremental thresholding approach introduced by Ahamd et al. in [2] and used along with the Hierarchical Temporal Memory (HTM) prediction algorithm to provide the HTM anomaly detector. Anomaly likelihood is a generalist method completely independent from the prediction model, which only needs the prediction errors.
- *DT* or *Dynamic threshold*. This method was introduced by Buda et al. [5] and computes the dynamic threshold based on a sliding window of historical normalized prediction errors that are labeled as inliner. The dynamic threshold is calculated as ten times the standard deviation of the historical window. An anomaly is reported if the normalized current error value is equal to or bigger than the dynamic threshold.
- *SS* or *Sigma scoring*. This is our online scoring proposal based on 3-sigma control charts and the scoring method proposed in [21], originally used to score the anomalies detected by One-Class based outlier detectors. In order to adapt this scoring method to online 3-sigma control limits, the radius is defined as $R = 3\sigma_t$ and the center as $o = \mu_t$; where σ_t and μ_t are the standard deviation and the mean of the current window W_t. Hence, this method fulfills these desirable four criteria:

  (1) $0 < score(x) < 1$
  (2) $score(x) = 0.5$ for samples on the boundary
  (3) $score(x) < 0.5$ for samples inside the boundary
  (4) $score(x) > 0.5$ for samples outside the boundary
- *DSS* or *Dynamic sigma scoring*. This is a dynamic version of the previous SS method. In this case, the mean and the standard deviation are computed dynamically whenever a new prediction error arrives instead of a sliding window. The dynamic mean μ_t, and dynamic standard deviation σ_t are computed using the Eq. (3) and Eq. (4) proposed in [4] for dynamic data normalization.

3 Use example

In this section we explain how to use the package. Section 3.1 describes how to install the package. In section 3.2 we show how to load the documentation. Then, section 3.3 includes some examples to show how to use the package. Finally, section 3.4 summarizes the most useful functions.

3.1 Installation

The **otsad** package is available at GitHub repository, so it can be downloaded and installed directly from the R command line by typing:

```
install.packages("devtools")
devtools::install_github("alaineiturria/otsad")
```

To easily access all the package's functions, it must be attached in the usual way:

```
library(otsad)
```

Special note:
CAD-OSE executes a python script so it is necessary to have python2 or python3 installed. In addition, to get the same results in all operating systems it is necessary to have installed the hashlib and bencode-python3 python libraries.

3.2 Documentation

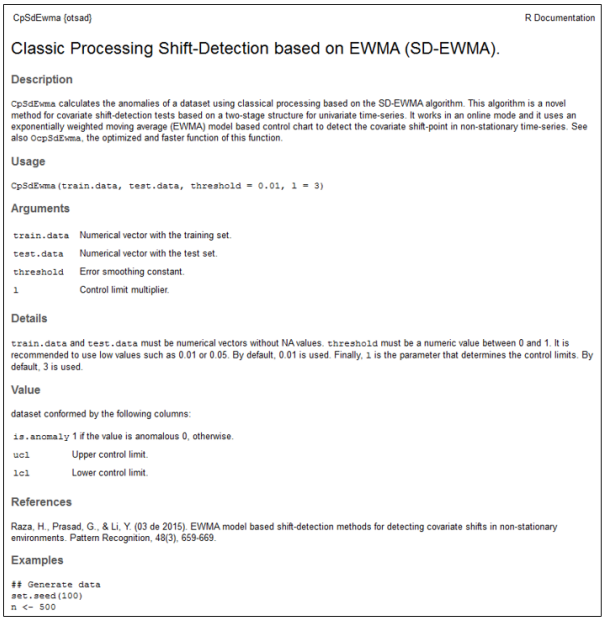
Considering that this vignette provides the user with an overview of the **otsad** package, it is also important to have access to the specific information of each of the available algorithms. This information can be checked in the documentation page corresponding to each algorithm. In all cases, the documentation has the same structure, consisting of the following sections (see Figure 3 for an example):

- A *description* section, which gives a brief description of what the algorithm consists of (like those given in Section 2).
- A *usage* section, where an overview of the function with the available parameters is given.
- An *arguments* section, where each of the input parameters is described.
- A *details* section, which provides the user with more details on the algorithm, conditions and recommendations on the values that can be taken by each of the input parameters.
- A *value* section, where the output parameters of the function are described.
- A *references* section that points to the original contribution where the detector(s) was proposed, where further details, motivations or contextualization can be found.
- An *examples* section, where one or more examples of the use of the functions are shown.

As usual in R, the documentation pages for each function can be loaded from the command line with the commands `?` or `help`:

```
?CpSdEwma
help(CpSdEwma)
```

In addition, a user manual is available in the github repository of the **otsad** package, which contains the complete documentation of the package and its functions.


CpSdEwma {otsad} R Documentation

Classic Processing Shift-Detection based on EWMA (SD-EWMA).

Description

CpSdEwma calculates the anomalies of a dataset using classical processing based on the SD-EWMA algorithm. This algorithm is a novel method for covariate shift-detection tests based on a two-stage structure for univariate time-series. It works in an online mode and it uses an exponentially weighted moving average (EWMA) model based control chart to detect the covariate shift-point in non-stationary time-series. See also OcpSdEwma, the optimized and faster function of this function.

Usage

```
CpSdEwma(train.data, test.data, threshold = 0.01, l = 3)
```

Arguments

train.data	Numerical vector with the training set.
test.data	Numerical vector with the test set.
threshold	Error smoothing constant.
l	Control limit multiplier.

Details

train.data and test.data must be numerical vectors without NA values. threshold must be a numeric value between 0 and 1. It is recommended to use low values such as 0.01 or 0.05. By default, 0.01 is used. Finally, l is the parameter that determines the control limits. By default, 3 is used.

Value

dataset conformed by the following columns:

is.anomaly	1 if the value is anomalous 0, otherwise.
ucl	Upper control limit.
lcl	Lower control limit.

References

Raza, H., Prasad, G., & Li, Y. (03 de 2015). EWMA model based shift-detection methods for detecting covariate shifts in non-stationary environments. Pattern Recognition, 48(3), 659-669.

Examples

```
## Generate data
set.seed(100)
n <- 500
```

Figure 3: Extract from CpSdEwma detectors documentation page, showing the highlighted above aspects.

3.3 Examples

For the sake of clarity we have divided the examples into three blocks. In Section 3.3.1 we introduce how to detect anomalies and visualize the results. Then Section introduces how to measure anomaly detectors. Finally, in Section we show an example of using the framework to adapt a prediction algorithm to outlier detection.

3.3.1 Anomaly detection and visualization

- **Example 1:** *How to solve a simple problem using SD-EWMA algorithm.*
  The data has been generated as follows:

```
## Generate data
set.seed(100)
n <- 500
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE) # distributional shift
x[25] <- 200 # abrupt transient anomaly
x[320] <- 170 # abrupt transient anomaly
df <- data.frame(timestamp = 1:n, value = x)
```

We can visualize the time-series as in Figure 4 by typing:

```
plot(x = df$timestamp, y = df$value, type = "l",
     main = "Time-Serie", col = "blue", xlab = "Time", ylab = "Value")
```

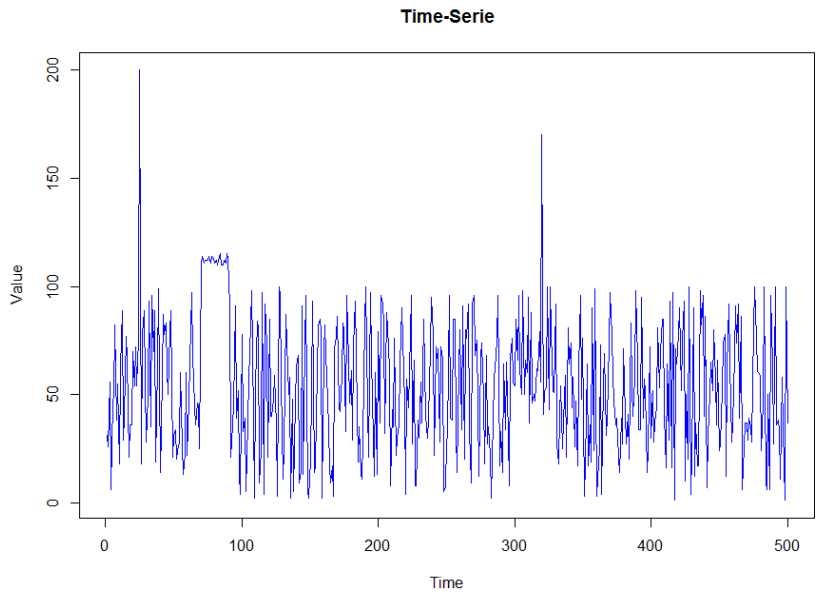


Figure 4: Our time-series visualization

The SD-EWMA algorithm is designed for stationary time-series. Therefore, we must first check that the time-series is stationary. To do this, we can display the acf and pacf graphics (Figure 5) by:

```
forecast::Acf(ts(df$value), main = "ACF", lag = 20)
forecast::Pacf(ts(df$value), main = "PACF", lag = 20)
```

We can observe that since almost all the lags are within the limits, the time-series is stationary. In addition, we can use two statistical tests, adf test and kpss test. The tests also indicate that the time-series is stationary.

```
library(tseries)
adf.test(df$value, alternative = 'stationary', k = 0)
kpss.test(df$value)
```

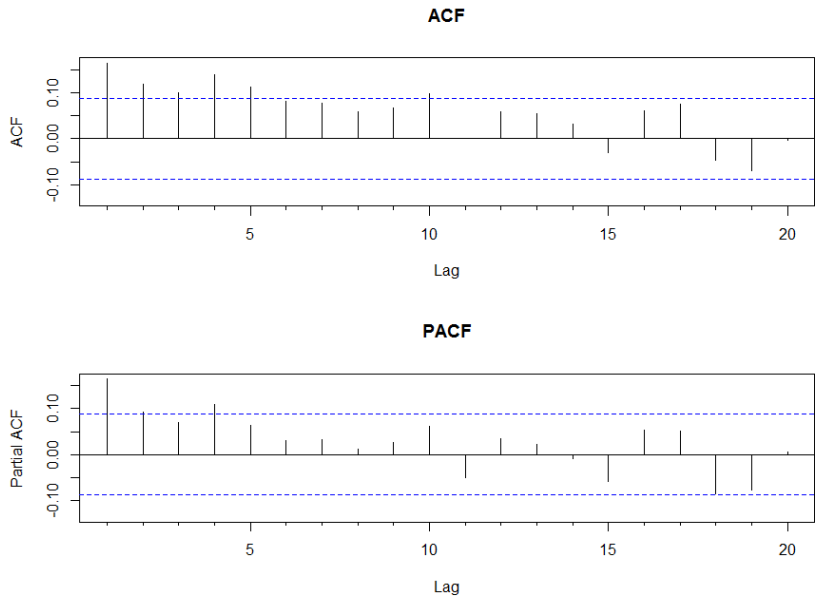


Figure 5: ACF and PACF graphs

On the recommendation of the algorithm authors we set **threshold** to 0.01.One of the usual values for the σ multiplier `l` is usually 3. Next, we apply the SD-EWMA anomaly detector, as the example is simple we use the classical processing algorithm. Note that the optimized algorithm could be used in the same way. Finally, we separate the training and test sets, using the first five values for training and the rest values for testing.

```
result <- CpSdEwma(data = df$value, n.train = 5, threshold = 0.01, l = 3)
```

This results are of data.frame type and have three columns: *is.anomaly* indicating whether or not the test observation is anomalous and the columns *ucl* and *lcl* with the upper and lower control limits, used to determine whether or not the observation is anomalous.

```
head(result, n = 15)
```

```
##    is.anomaly         lcl      ucl
## 1           0  74.0000000  74.0000
## 2           0  89.0000000  89.0000
## 3           0  78.0000000  78.0000
## 4           0  23.0000000  23.0000
## 5           0  86.0000000  86.0000
## 6           0   0.2998149 139.0313
## 7           0   0.6808881 138.7171
## 8           0  -8.3155413 134.5737
## 9           0  -8.8121491 133.4445
## 10          0  -7.7247629 133.8939
## 11          0  -4.6526720 137.8049
## 12          0 -12.4721436 133.7091
## 13          0 -19.2253248 129.7386
## 14          0 -18.8776837 129.3396
## 15          0 -19.8205462 127.8363
```

Finally, we can plot the results (Figure 6) by writing the following code:

```
res <- cbind(df, result)
PlotDetections(res, title = "SD-EWMA ANOMALY DETECTOR")
```

In Figure 6 we can see that the detector has detected well both abrupt transient anomalies and distributional shift anomaly.

This example has used classical processing, but by using the incremental function we can simulate online processing as follows:

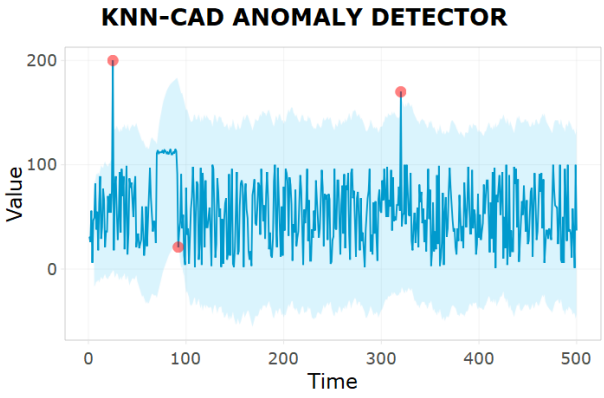


Figure 6: Sd-Ewma Anomaly detector results

```r
## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 250
numIter <- n%/%nread
iter <- seq(1, nread * numIter, nread)

## Calculate anomalies
for(i in iter) {
  # read new data
  newRow <- df[i:(i + nread - 1),]
  # calculate if it's an anomaly
  last.res <- IpSdEwma(
    data = newRow$value,
    n.train = 5,
    threshold = 0.01,
    l = 3,
    last.res = last.res$last.res
  )
  # prepare the result
  if(!is.null(last.res$result)){
    res <- rbind(res, cbind(newRow, last.res$result))
  }
}
```

In the same way we can plot the results (Figure reffig:result2) by writing:

```r
PlotDetections(res, title = "SD-EWMA ANOMALY DETECTOR")
```

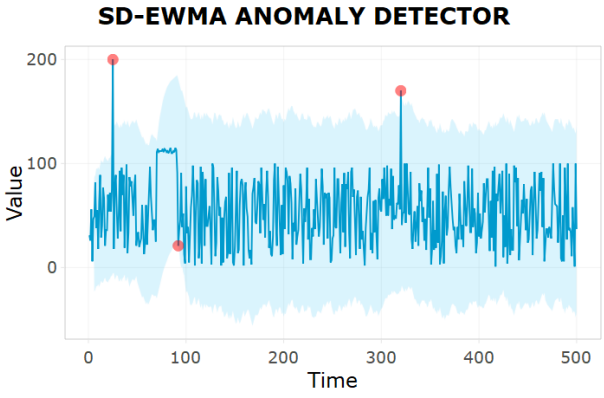


Figure 7: Online Sd-Ewma Anomaly detector results

- **Example 2:** *How to use Contextual Anomaly Detector.* The use of the Contextual Anomaly Detector is similar to the use of the previous algorithm. It is important to note that if the bencode python module is not installed, this method will rise an error. This can be prevented installing the bencode-pyhthon3 library with Python package manager `pip`.

```
cad.results <- ContextualAnomalyDetector(
  df$value[1:250],
  base.threshold = 0.75,
  rest.period = 5,
  max.value = 200,
  min.value = 0
)
cad.results.online <- ContextualAnomalyDetector(
  df$value[251:500],
  python.object = cad.results$python.Object
)

res <- cbind(df, rbind(cad.results$result, cad.results.online$result))
PlotDetections(res, title = "CONTEXTUAL ANOMALY DETECTOR")
```

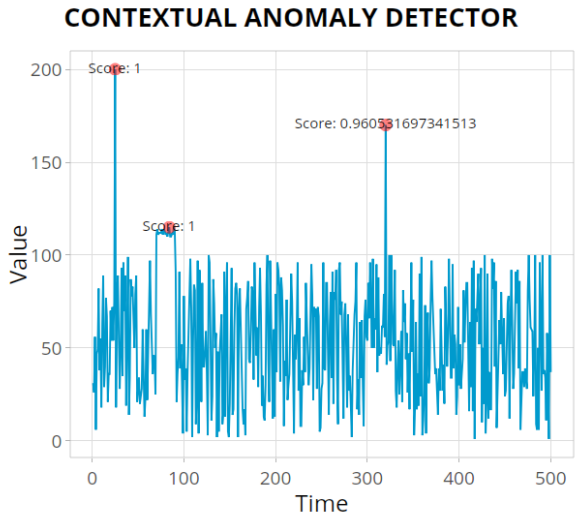


Figure 8: Contextual Anomaly Detector results

- **Example 3:** *How to use EORELM-AD.* The use of the EORELM-AD algorithm is quite like the use of previous algorithms. However, for its easier use, it is implemented as an R6 class. Because of that, unlike the other algorithms, first, the EorelmAD object must be created as follow.

```
detector <- EorelmAD$new(
  n.train = 20,
  numLags = 10,
  numHiddenNeurons = c(20),
  normMethod = NULL,
  outlierMethod = "DSS",
  threshold = 0.5
)
```

Now the anomaly detection is carried out. To do this, the prediction method is used. As an argument, it takes a single value or a set of values in which anomalies should be detected. In this case, the whole dataset is used.

```
result <- detector$predict(df$value)

## Plot results
res <- cbind(df, result)
```

```
PlotDetections(res, title = "EORELM-AD ANOMALY DETECTOR", return.ggplot = TRUE)
```

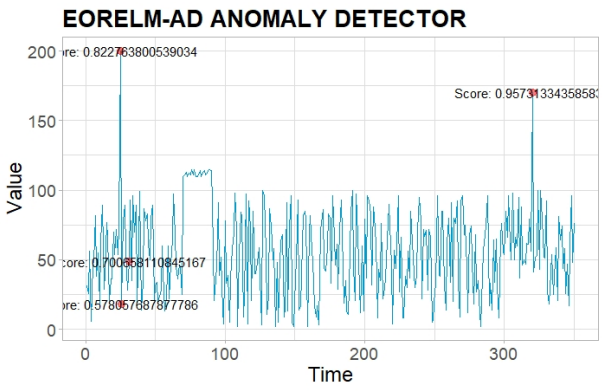


Figure 9: EORELM-AD results

- **Example 4:** *How to use KNN.* In this example KNN is used to detect the five anomalies of the *Speed_7578* dataset.

```
# Load the previously installed otsad package
library("otsad")

# Load the dataset speed_7578, included in otsad
myData <- speed_7578

# Initialize parameters
n <- nrow(myData)
n.train <- GetNumTrainingValues(n)

# Calculate anomalies using KNN-CAD
result <- CpKnnCad(
  data = myData$value,
  n.train = n.train,
  threshold = 1,
  l = 19,
  k = 27,
  ncm.type = "ICAD",
  reducefp = TRUE
)

# Add results to dataset
myData <- cbind(myData, result)
```

Once the algorithm is applied, we could show the results using the PlotDetections function. Since the resulting graph is interactive, in Figure 10 we show a capture of it.

```
# Plot Results
PlotDetections(myData, title = "KNN-CAD ANOMALY DETECTOR")
```

3.3.2 Detector measuerement

Example 4 is continued to show how to measure the performance of the detector. Hence, the detector's score is obtained as follow:

```
# Get detector score
score <- GetDetectorScore(myData)

# See this results
data.frame(score[-1])
```

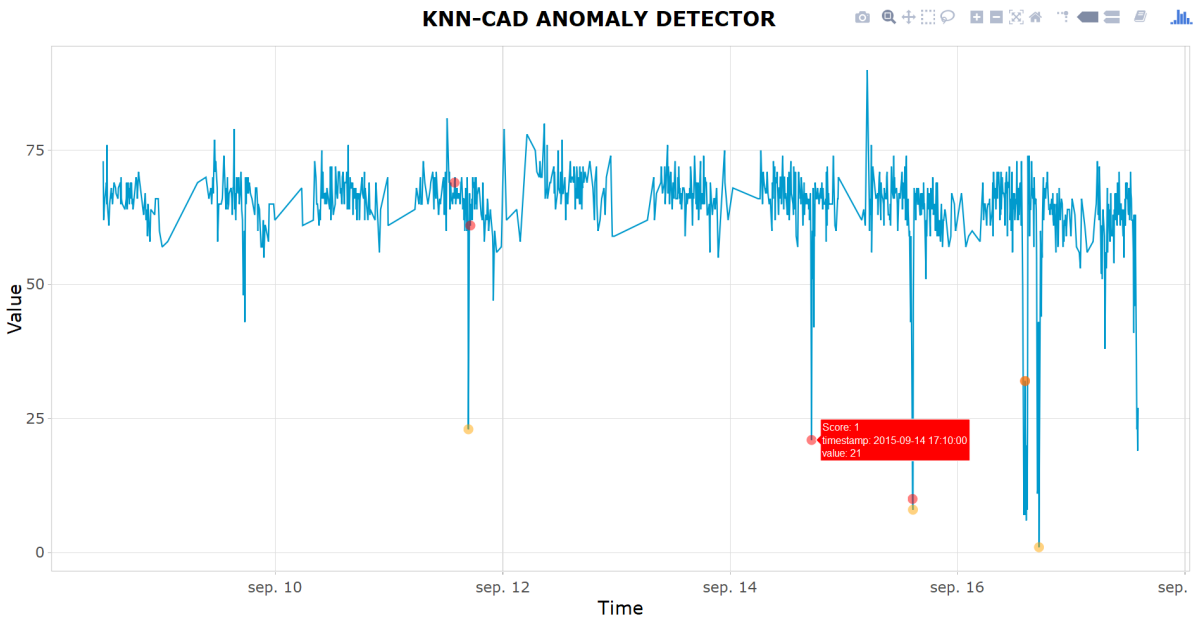


Figure 10: Anomalies detected in speed_7578 dataset by KNN_CAD detector.

This function has an option to show the results on a chart by adding the following parameters, see 11.

```
# Get detector score
score <- GetDetectorScore(
  myData,
  print = TRUE,
  title = "speed_7578 results using KNN-CAD detector"
)
```

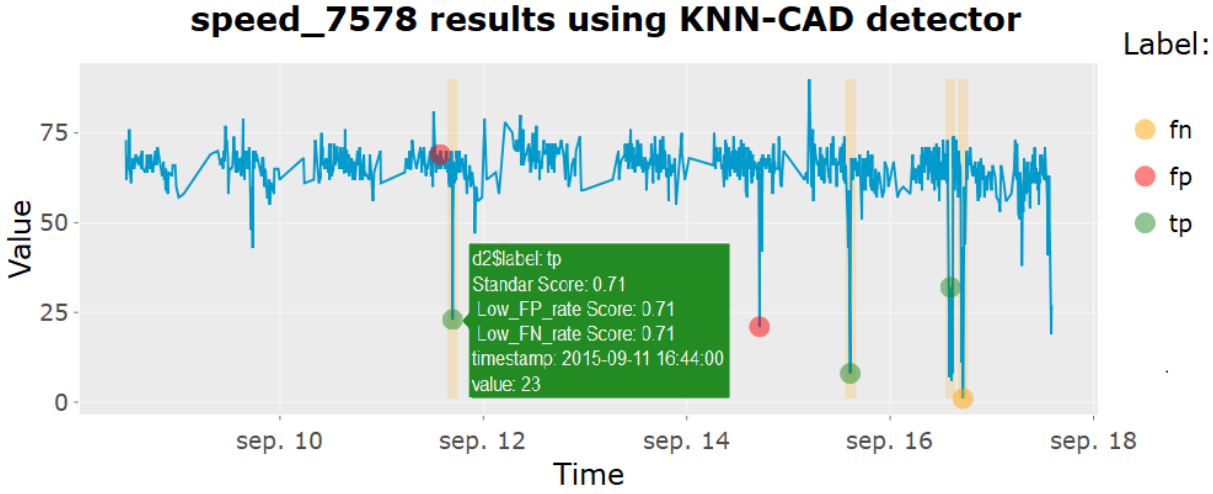


Figure 11: KNN-CAD detector measurement results for speed_7578 dataset.

Finally, we normalize the scores and show them.

```
# Normalize results
null.perfect <- GetNullAndPerfectScores(myData)

standar.score <- NormalizeScore(
  score$standard,
  perfect.score = null.perfect[1, "perfect.score"],
```

```
  null.score = null.perfect[1, "null.score"]
)

low_FP_rate.score <- NormalizeScore(
  score$low_FP_rate,
  perfect.score = null.perfect[2, "perfect.score"],
  null.score = null.perfect[2, "null.score"]
)

low_FN_rate.score <- NormalizeScore(
  score$low_FN_rate,
  perfect.score = null.perfect[3, "perfect.score"],
  null.score = null.perfect[3, "null.score"]
)

# Show normalized scores
cbind(standar.score, low_FP_rate.score, low_FN_rate.score)
```

3.3.3 Adapting prediction algorithm to anomaly detection

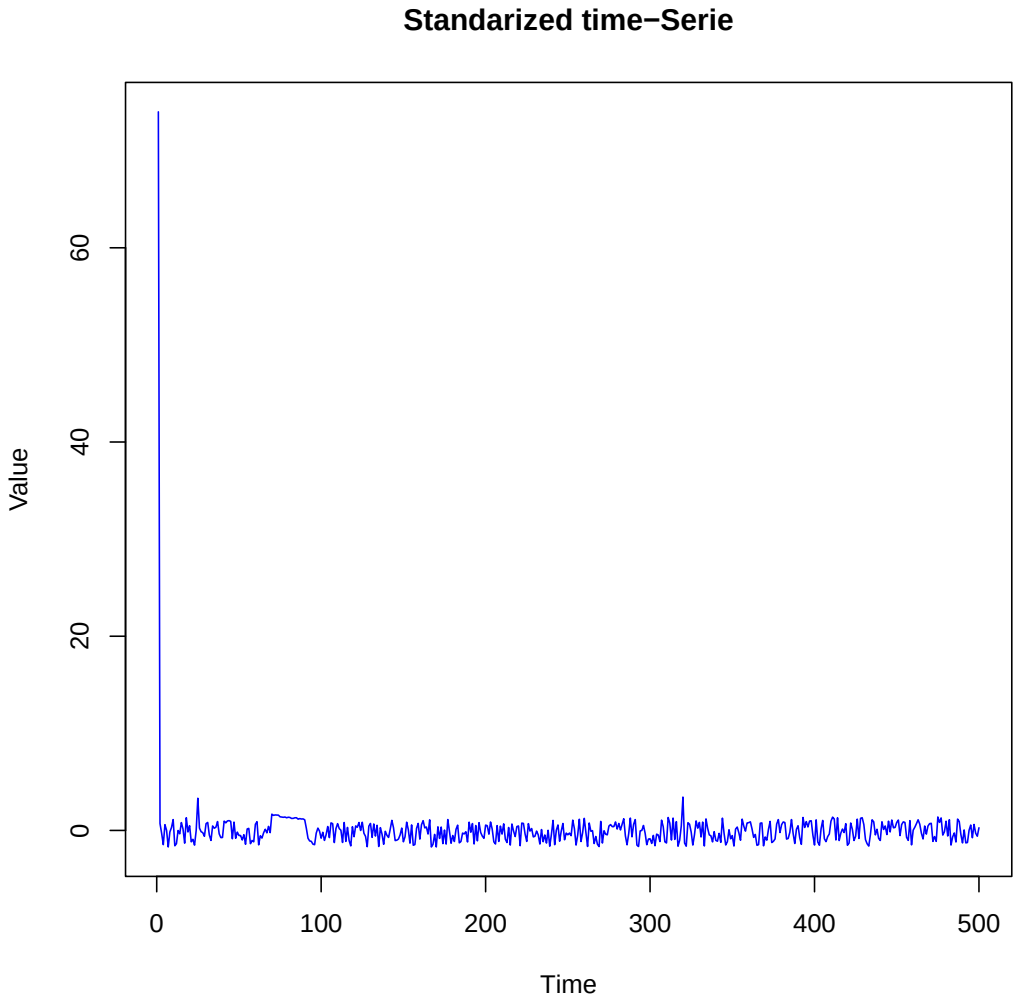
The data set used for this example is the following one:

```
## Generate data
set.seed(100)
n <- 500
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE) # distributional shift
x[25] <- 200 # abrupt transient anomaly
x[320] <- 170 # abrupt transient anomaly
df <- data.frame(timestamp = 1:n, value = x)
```

Now we standardize the data set, using the dynamic normalization method.

```
## Generate data
set.seed(100)
normalizer <- DynamicNormalizer$new()
online_norm_value <- sapply(df$value, normalizer$normalize)

plot(online_norm_value, type = "l",
       main = "Standarized time-Serie", col = "blue", xlab = "Time", ylab = "Value")
```



As a prediction model EORELM-AD is used and it is initialized as in Example 3.

```
detector <- EorelmAD$new(
  n.train = 20,
  numLags = 10,
  numHiddenNeurons = c(20),
  normMethod = NULL,
  outlierMethod = "DSS",
  threshold = 0.5
)
```

Then the prediction on the the whole dataset is carried out.

```
result <- detector$predict(online_norm_value)
res <- cbind(df, result)
```

To show how to use the online scoring function in the following, we calculate the anomaly score from the prediction error obtained in the previous run.

```
  scorer <- DynamicSigmaScorer$new()
  anomaly.score <- sapply(res$error, scorer$computeScore)
```

Finally, it can be seen that the score obtained by DynamicSigmaScoring and the score obtained by EORELM-AD are exactly the same.

```
 setdiff(anomaly.score, res$anomaly.score)
```

```
## numeric(0)
```

```r
cbind("score" = anomaly.score, "res-score" = res$anomaly.score)[100:105,]
```

```
##              score    res-score
## [1,] 0.0044822023 0.0044822023
## [2,] 0.0110590130 0.0110590130
## [3,] 0.0240821506 0.0240821506
## [4,] 0.0135302591 0.0135302591
## [5,] 0.0037784959 0.0037784959
## [6,] 0.0007961603 0.0007961603
```

3.4 Most useful functions

Functionality	Description	Offline	Online
Detectors	PEWMA	CpPewma	IpPewma
	SD-EWMA	CpSdEwma	IpSdEwma
	TSSD-EWMA	CpTsSdEwma	IpTsSdEwma
	KNN-ICAD	CpKnnCad(ncm.type = "ICAD")	IpKnnCad(ncm.type = "ICAD")
	KNN-LDCD	CpKnnCad(ncm.type = "LDCD")	IpKnnCad(ncm.type = "LDCD")
	CAD-OSE	ContextualAnomalyDetector	ContextualAnomalyDetector
	EORELM-AD (R6)	EorelmAD	EorelmAD
NAB score	Get score	GetDetectorScore	-
	Normalize score:	NormalizeScore GetNullAndPerfectScores	-
False Positve Reduction	False positive reduction	ReduceAnomalies	ReduceAnomalies
Visualization	Static or interactive visualizations	PlotDetections	-
Online normalization	Dynamic normalization (R6)	DynamicNormalizer	DynamicNormalizer
	Window normalization (R6)	WindowNormalizer	WindowNormalizer
	One-pass adaptive normalization (R6)	AdaptiveNormalizer	AdaptiveNormalizer
	One-pass adaptive min-max normalization (R6)	AdaptiveNormalizer2	AdaptiveNormalizer2
Online scoring	Anomaly likelihood (R6)	AnomalyLikelihoodScorer	AnomalyLikelihoodScorer
	Dynamic threshold (R6)	DynamicThresholdScorer	DynamicThresholdScorer
	Sigma scoring (R6)	SigmaScorer	SigmaScorer
	Dynamic sigma scoring (R6)	DynamicSigmaScorer	DynamicSigmaScorer

4 Summary

In this paper, we present the **otsad** package for R. This package fully meets the demand for anomaly detection algorithms over univariate time series in online environments. With this package we tackle with the ability to work with stationary and non-stationary data. We also provide algorithms of the two detection techniques (evolving and window based) that are gaining strength on research.

As a future job, we propose to maintain and add functionalities to our **otsad** package, i.e. provide more sophisticated false positive reduction techniques and incorporate them into algorithms.

References

[1] Charu C Aggarwal. *Outlier analysis*. Springer, Cham, New York, NY, 2nd edition edition, 2017.

[2] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[3] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Computing Surveys*, 54(3), 2021.

[4] Danushka Bollegala. Dynamic feature scaling for online learning of binary classifiers. *Knowledge-Based Systems*, 129:97–105, 2017.

[5] Teodora Sandra Buda, Bora Caglayan, and Haytham Assem. DeepAD: A generic framework based on deep learning for time series anomaly detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10937 LNAI, pages 577–588, 2018.

[6] E. Burnaev and V. Ishimtsev. Conformalized density- and distance-based anomaly detection in time-series data. *ArXiv e-prints*, 2016.

[7] K. M. Carter and W. W. Streilein. Probabilistic reasoning for streaming anomaly detection. In *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pages 377–380, 2012.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.

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

[10] Vibhuti Gupta and Rattikorn Hewett. Adaptive Normalization in Streaming Data. In *Proceedings of the 2019 3rd International Conference on Big Data Research*, pages 12–17, New York, NY, USA, 2019. ACM.

[11] Simon S. Haykin. *Neural networks and learning machines.* Pearson Education, Upper Saddle River, 3 edition, 2009.

[12] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[13] V. Ishimtsev, I. Nazarov, A. Bernstein, and E. Burnaev. Conformal k-NN Anomaly Detector for Univariate Data Streams. *ArXiv e-prints*, 2017.

[14] 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)*, pages 38–44, 2015.

[15] Nan Ying Liang, Guang Bin Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, 2006.

[16] Jessica Lin and Eamonn Keogh. FINDING OR NOT FINDING RULES IN TIME SERIES. In *Advances in Econometrics*, volume 19, pages 175–201, 2004.

[17] Eduardo Ogasawara, Leonardo C. Martinez, Daniel De Oliveira, Geraldo Zimbrão, Gisele L. Pappa, and Marta Mattoso. Adaptive Normalization: A novel data normalization approach for non-stationary time series. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 2010.

[18] Jin Man Park and Jong Hwan Kim. Online recurrent extreme learning machine and its application to time-series prediction. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1983–1990, 2017.

[19] Haider Raza, Girijesh Prasad, and Yuhua Li. Ewma model based shift-detection methods for detecting covariate shifts in non-stationary environments. *Pattern Recogn.*, 48(3):659–669, 2015.

[20] Mikhail Smirnov. Contextual Anomaly Detector. Contribute to smirmik/CAD development by creating an account on GitHub, September 2018.

[21] Biao Wang and Zhizhong Mao. Outlier detection based on a dynamic ensemble model: Applied to process monitoring. *Information Fusion*, 51:244–258, 2019.

[22] Zhao Xu, Kristian Kersting, and Lorenzo Von Ritter. Stochastic online anomaly analysis for streaming time series. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, (IJCAI-17)*, pages 3189–3195, 2017.

[23] Yang Zhang, N. Meratnia, and P. Havinga. Outlier detection techniques for wireless sensor networks: A survey. *Commun. Surveys Tuts.*, 12(2):159–170, 2010.