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

Alaiñe Iturria[1,2], Jacinto Carraso[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

Anomalies in time-series data give essential and often actionable information to be extracted from critical situations. 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 six of the most recent detection algorithms capable of dealing with various challenges, such as online and non-stationary time-series anomaly detection.

1 Introduction

Anomalies in time-series data give essential and often actionable information to be extracted for critical situations. This is an important task in many domains such as fault detection in manufacture, intrusion detection in cyber security or fraud detection in banks.

Anomaly detection is a wide field that has been studied for years. Chandola et al. [3], Hodge et al. [5] and Zhang et al. [10] provide a comprehensive overview of anomaly detection techniques. The detection of anomalies involves the identification of patterns in the data that differ from expected behavior. In the classical literature the anomaly detection methods can be broadly categorized in six families: classification-based, clustering-based, nearest neighbors-based, statistics, information-theoretic based and spectral techniques.

Due to the evolution of new technologies, the amount of data is increasing and it 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. As opposed to classic techniques, online time-series anomaly detection techniques do not have the complete data set to work with and must take time into account. In recent years, most of the work has focused on the evolution of predictive and moving window-based techniques [4].

In R statistical software, there are few packages available from *Comprehensive R Archive Network (CRAN)* to address the problem of time-series anomaly detection. The first and most popular is the **tsoutliers** package. There is also the **qicharts** package that implements basic control chart algorithms. Finally, another package that is becoming increasingly popular is **Twitter's AnomalyDetection** package (only available on GitHub). One of the main disadvantages of these three packages is that the implemented algorithms are not suitable to work online. In addition to the aforementioned, there are two other packages, **SmartSifter** and **EnergyOnlineCPM** available for online time-series anomaly detection.

In conclusion, there are only a few packages for online anomaly detection available in CRAN. For this reason, we propose the **otsad** pacakge that implements 6 recent and powerful algorithms for the detection of anomalies in univariate time-series able to work online.

In the following section 2 we briefly introduce what online time-series anomaly detection is all about. In in the following section 3 we show how to use the **otsad** package using two easy examples. Finally, in section 4, we present an overview of this work.

2 Online anomaly detection in time-series

The detection of anomalies in online time-series is a difficult task because it must be able to consider the time (or the position of each observation) and it does not have entire dataset available. As mentioned earlier, in the last few years it is focusing on the evolution of predictive-based and moving window-based detection techniques.

Predictive techniques are models in which the parameters or components of the model are modified as new data arrive to capture better normal data trends. They calculate if a value is anomalous based on the dissimilarity or the error obtained between the observation and its prediction. On the other hand, techniques based on moving windows are are commonly used to improve distance-based techniques. Distance calculation is an expensive process and proportional to the number of observations to be considered, the use of moving windows allows reducing the set of observations to be considered maintaining the most recent subset of data.

Another major challenge with the introduction of online processing is that time-series can be stationary and non-stationary. A time-series is said to be stationary when the mean and variance remain constant over time. In addition, they must be neither seasonal nor trendy. Most statistical or predictive techniques assume that time-series are stationary, when in most real cases they never are. It is true that there are techniques to turn a non-stationary time-series into a stationary one, however, this is not a simple task and there are cases this can not be done stationary.

With the **ostad** package we provide six algorithms that address these three challenges. All algorithms can be used online. Furthermore, two of the six algorithms can be used in stationary environments and the other four in non-stationary environments. Finally, three of the techniques are based on predictive methods while the other three are based on moving windows techniques.

3 The otsad package

In our opinion, this is the first review and package for R that develops a set of current and powerful anomaly detectors for the online detection of anomalies in a univariate time series. The **otsad** package implements six anomaly detection algorithms. But also other functionalities and contents that can be interesting.

3.1 package contents

For an easier understanding of this section, we divided it into four subsections. Subsection 3.1.1 describes each of the implemented algorithms. In subsection 3.1.2 we present a novel detector evaluation technique. In Subsection 3.1.3 we introduce included data sets and the function to visualize the results. Finally, in subsection 3.1.4 we describe a simple but effective technique for reducing false positives.

3.1.1 Anomaly detection algorithms

This package implements, documents, explains and provides references for a set of detectors listed below. The first three algorithms belong to prediction based techniques and the last three belong to the window-based ones. The first two algorithms are for stationary environments and the other four for non-stationary environments. Table 1 shows the most important characteristics of each algorithm.

- *PEWMA* or *Probabilistic reasoning for streaming anomaly detection* [2]. The probabilistic reasoning for streaming anomaly detection. This algorithm is a probabilistic method of EWMA which dynamically adjusts the parameterization based on the probability of the given observation. This method produces dynamic, data-driven anomaly thresholds which are robust to abrupt transient changes, yet quickly adjust to long-term distributional shifts.
- *SD-EWMA* or *Shift-Detection based on EWMA* [8]. This algorithm is a novel method for covariate shift-detection tests for univariate time-series. It works in an online mode. It uses an exponentially weighted moving average (EWMA) model based control chart to detect the covariate shift-point in stationary time-series.
- *TSSD-EWMA* or *Two-Stage Shift-Detection based on EWMA* [8]. 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. It uses an exponentially weighted moving average (EWMA) model based control chart to detect the covariate shift-point in non-stationary time-series. It works in two phases. In the first phase, it applies SD-EWMA. In the second phase, it checks the authenticity of the anomalies using the Kolmogorov-Simirnov test to reduce false alarms.

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

Table 1: Features of the algorithms

- *KNN-CAD* or *Conformal k-NN Anomaly Detector* [1]. This algorithm is a model-free anomaly detection method for univariate time-series which 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* [6]. 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 techniques, KNN, and statistical techniques to determine the degree of anomaly.
- *CAD-OSE* or *Contextual Anomaly Detector* [9]. This algorithm discretizes the search space and creates contexts with aggrupations of observations. When a new instance arrives, the algorithm searches in the previous storaged contexts if the instance has been seen and creates a new context if it has not. The creation of new contexts and the storaged ones determine if the instance is an anomaly.

Each of these algorithms was implemented to work in two different scenarios. On the one hand, classical processing used when the complete data set (train and test) is available. On the other hand, the incremental processing used when no dataset is available. It allows calculating the abnormality of the new observation(s) with the parameters updated in the last run performed.

3.1.2 Detector measurement technique

The **Otsad** package implements the method used in NAB [7] to measure the detector. This method considers the time elapsed until the anomalies are detected.

This metric uses windows to determine the label of the detected anomaly, i.e. True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). To determine this label, it is centered a window on the real anomaly, so if the detected anomaly falls inside, it is considered as a TP, if it falls outside as a FP and if there are no anomalies inside the window as a FN. The window size is calculated as the division between 10% of the number of observations and the number of real anomalies in the time series. The package includes three functions to perform these tasks. On the one hand, **GetWindowLength** and **GetWindowsLimits** that gets the size of the window and the start and end limits of each of them after being focused on the real anomalies. On the other hand, **GetLabels** to get the TP, FP, TN and FN labels.

It also allows setting a weight for each of these labels. These weights make 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. The authors propose 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 FP and the 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 normalized using the Max-Min normalization as in Eq. 1. The maximum value used, is the score of a perfect detector, i.e. one that detect all anomaly with maximum score and 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 a value between -inf and 100. For these latest objectives, we have three more functions. First one is **GetDetectorScore**, that calculates the detector score without normalizing. This function uses the three earlier functions. 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.

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

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

This package incorporates an additional function to be able to reproduce the results of the benchmarking. Named as **GetNumTrainingValues**, this function allows obtaining the number of instances used as a training set in NAB [7]. The number of training set values is calculated as 15% of 5000 and for smaller datasets of 5000 instances such as 15% of the dataset instances.

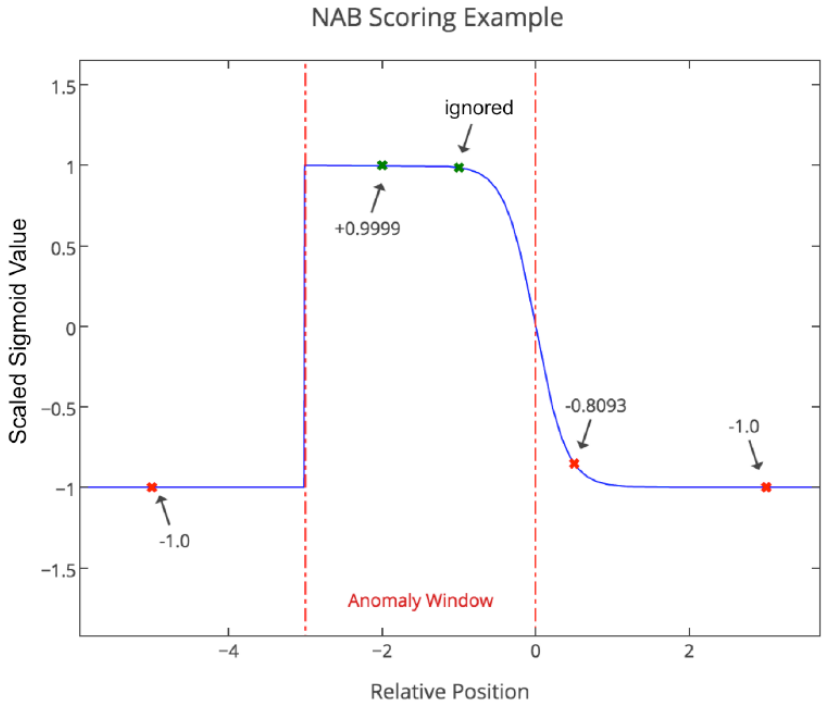


Figure 1: Scoring example for a sample anomaly window, where the values represent the scaled sigmoid function. The first point is an FP preceding the anomaly window (red dashed lines) and contributes -1.0 to the score. Within the window, we see two detections, and only count the earliest TP for the score. 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 [7]

.

3.1.3 Data sets and function for displaying the results

Otsad includes some data sets from the NAB repository, which can be used to perform experiments. In [7] authors provide a publicly available set of 58 labeled one-dimensional time-series from different fields. This package includes 51 of the 58 datasets available in the NAB 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.

It 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 be able to edit it.

3.1.4 False positive reduction technique

To finish this section, we introduce a new technique for the reduction of false positives. In this work, we have tried to implement the original algorithm proposed by the authors. However, all of them give many false positives. For KNN-CAD [1] and KNN-LDCD [6] algorithms, authors suggest a technique to reduce false positives. So that all the algorithms have the same opportunities, we introduce and implement a simple algorithm to reduce false positives. This algorithm is based on the idea that on a practical case, exists a time lapse between an alarm is triggered and until action is taken. This time-lapse is modeled by a window which length is defined by the user. This window is focused on the first anomaly detected and all the others that fall inside, are discarded. The function's name available in the package is **ReduceAnomalies.** It is available only for classical processing and after the detector has been applied.

3.2 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 python libraries.

3.3 Documentation

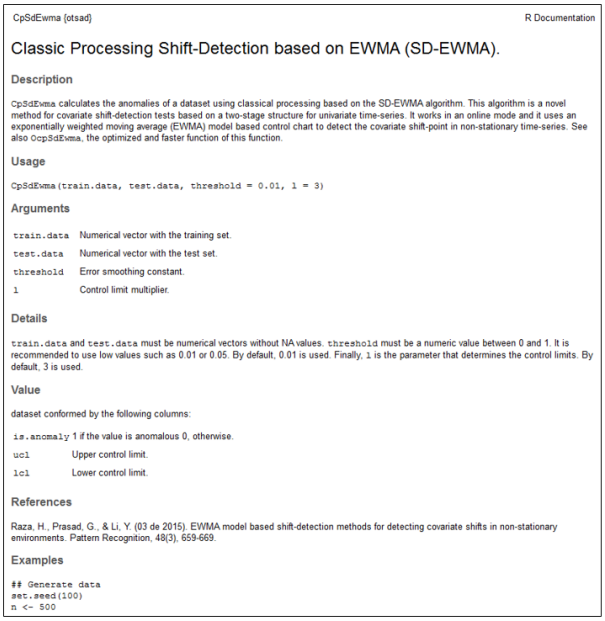
Considering that this vignette provides the user with an overview of the **otsad** package, it is also important to have access to the specific information of each of the available algorithms. This information can be checked in the documentation page corresponding to each algorithm. In all cases, the documentation has the same structure, consisting of the following sections (see Figure 2 for an example):

- A *description* section, which gives a brief description of what the algorithm consists of (like those given in Section 3).
- 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 2: Extract from CpSdEwma detectors documentation page, showing the highlighted above aspects.

3.4 Use example

Here is an example of how to solve a simple problem. The example will be made with the 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 3 by typing:

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

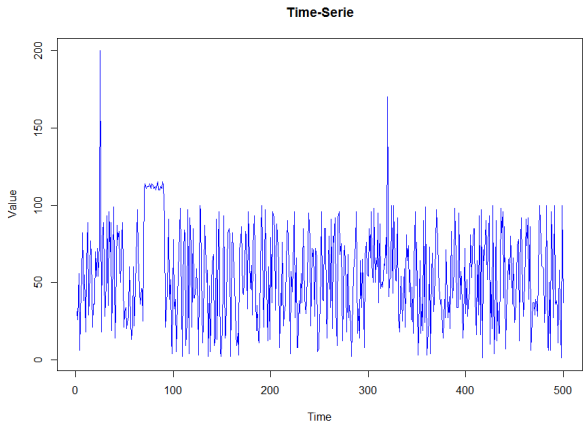


Figure 3: Our time-series visualization

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 4) by:

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

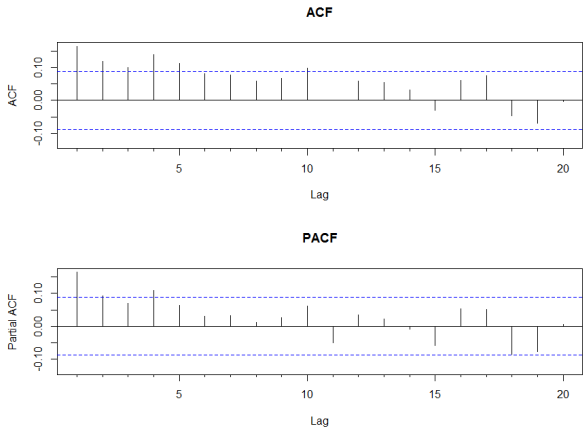


Figure 4: ACF and PACF graphs

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)
```

On the recommendation of the algorithm authors, also indicated in the details section of this function, 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)
```

If we print the results we can observe that the 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  31.000000 31.00000
## 2           0  26.000000 26.00000
## 3           0  56.000000 56.00000
## 4           0   6.000000  6.00000
## 5           0  47.000000 47.00000
## 6           0 -16.718459 83.33762
## 7           0 -15.120715 84.87796
## 8           0 -12.127443 91.30896
## 9           0 -12.029491 90.89286
## 10          0 -10.427275 92.40430
## 11          0 -12.931167 90.31049
## 12          0 -10.756584 92.99798
## 13          0  -7.669897 99.48715
## 14          0  -9.332986 97.76851
## 15          0  -9.501358 97.09333
```

Finally, we can see the results (Figure 5) by writing the following code:

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

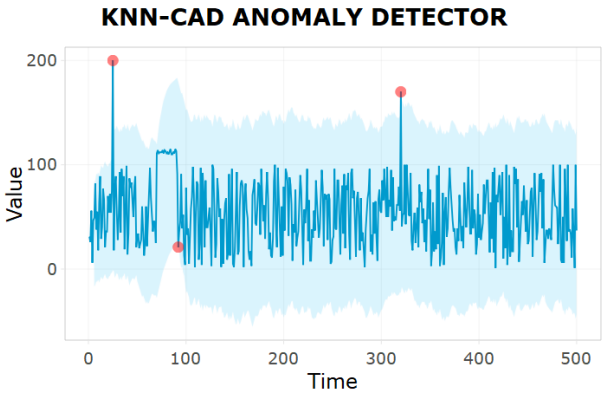


Figure 5: Sd-Ewma Anomaly detector results

In Figure 5 we can see that the detector has detected both abrupt transient anomalies well and the third one, distributional shift anomaly, a little late.

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

```
## 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 view the results (Figure reffig:result2) by writing:

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

The use of the Contextual Anomaly detector is similar to the combined use of the Classic and Incremental use of the previous algorithms.

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

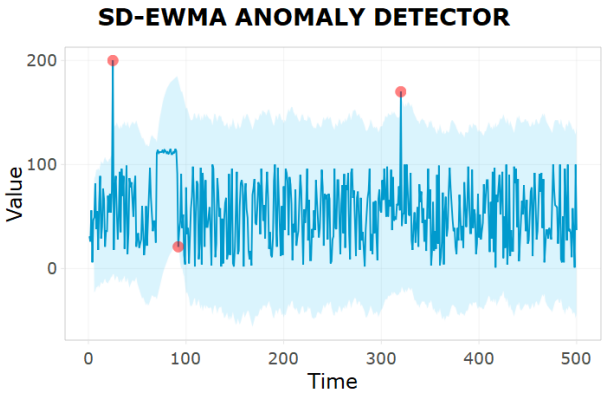


Figure 6: Online Sd-Ewma Anomaly detector results

```
  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")
```

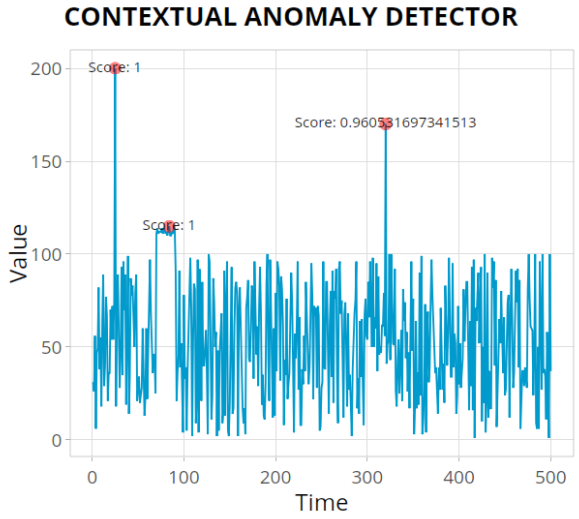


Figure 7: Contextual Anomaly Detector results

Finally we show a last example using a data set included in the package. For this example, we used the *Speed_7578* dataset included in the package which has five anomalies. We used the KNN algorithm to try to find those anomalies.

```
# 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 will show the results using the PlotDetections function. Since the resulting graph is interactive, in Figure 8 we show a capture of it.

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

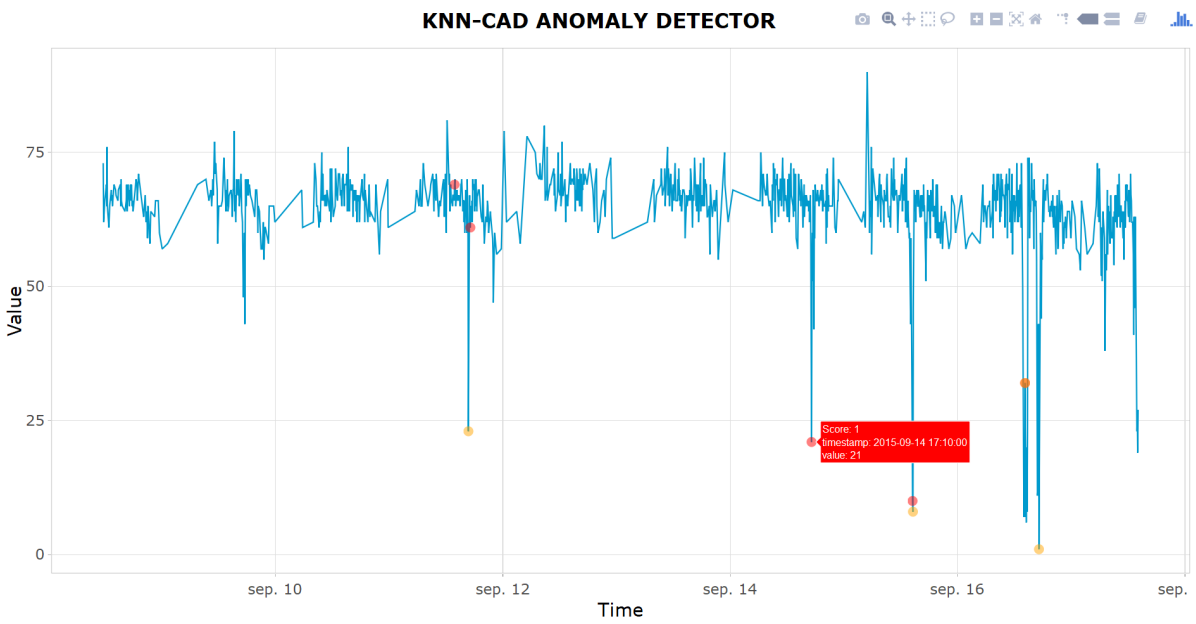


Figure 8: Anomalies detected in speed$_7$578$dataset by KNN_CAD detector$.

Next, we get the detector's score.

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

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

This function has an option to show the results on a chart by adding the following parameters, see 9.

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

Finally, we normalize the scores and show them.

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

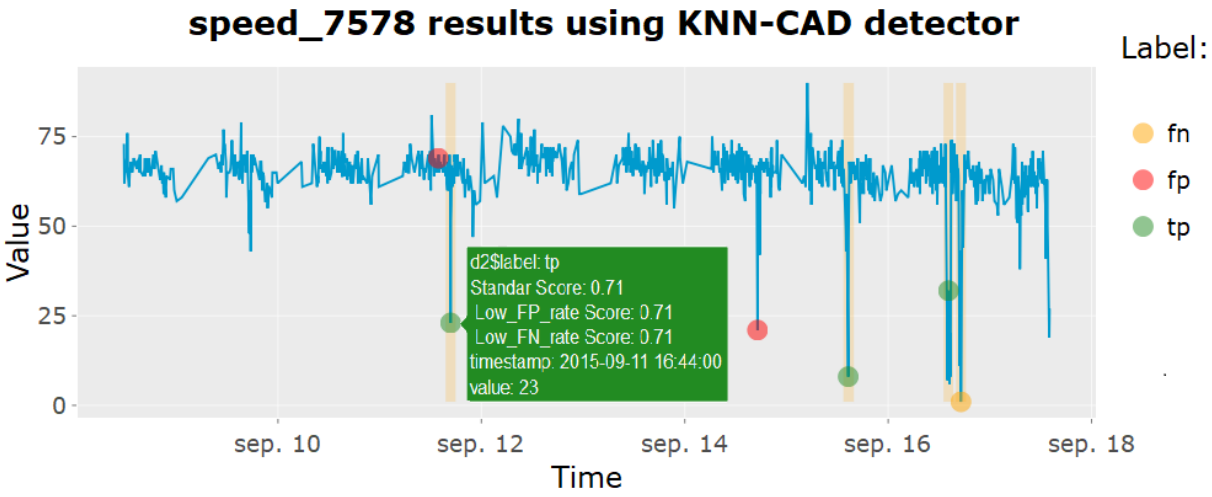


Figure 9: KNN-CAD detector measurement results for speed_7578 dataset.

```
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)
```

4 Summary

In this vignette we introduce the **otsad** package, which implements five recent algorithms for online anomaly detection. In order to the context and motivation for this work we present the new challenges faced by time-series anomaly detection techniques with the introduction of the need for online processing. As explained in section 2, the package proposes three algorithms based on the most recent prediction techniques (derived from EWMA) and two, based on moving window techniques (derived from KNN-CAD). Likewise, we cover the problem of the stationarity of time-series since two of the proposed algorithms have been designed to work in stationary environments and the other three can work in non-stationary environments. Finally, all algorithms can be used in two different scenarios: classical processing and incremental processing.

Regarding the future extensions of this package, there are several aspects that may be addressed in the future (e.g., functions for displaying the results). In addition, a new algorithm and a new metric are already being developed for the evaluation of detector quality. Several pre-labelled time-series data sets will also be included.

References

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

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

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

[4] 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, 9 2014.

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

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

[7] 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, Dec 2015.

[8] 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, March 2015.

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

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