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 six 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. [**?**], Hodge et al. [**?**] and Zhang et al. [**?**] provide a comprehensive overview of anomaly detection techniques. In the classical literature anomaly detection methods can be categorized into six families: classification-based, clustering-based, nearest neighbours-based, statistics, information-theoretic based and spectral techniques.

Unlike classical techniques, techniques for anomaly detection in time series must be able to take time (or the position of each observation) into account. In also, most of the existing statistical techniques assume that the time-series is stationary. A time-series is said to be stationary when the mean and variance remain constant over time, and it hasn't got a trend or seasonal component.

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 time must be taken into account. Another major challenge with the introduction of online time series processing is that time-series can be stationary and non-stationary. As we already previously commented, most statistical or predictive techniques assume that time-series are stationary, but in most real cases they never are. Although there are techniques to make time-series stationary, it may be difficult to make it stationary because in online processing the full data set is not available.

In recent years, most of the work has focused on the evolution of evolving prediction models and sliding window-based techniques [**?**]. 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.

Despite the detection of anomalies 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)* [**?**] including **Numenta HTM**, **CAD-OSE**, **KNN-CAD**, and others. For R, there are few CRAN packages 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. 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**. Moreover, we only found two CRAN packages able to address these challenges. The general conclusion is that few algorithms and packages are available and ready to use for online anomaly detection.

For all explained reasons, in this paper, we present 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. With this package, we intend to address both evolving and sliding window-based techniques that are gaining strength, including algorithms of both.

The rest of the paper is organized as follows. The next 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

In this section we introduce the first package for R that develops a set of current and effective online anomaly detectors for univariate time series. The **otsad** package implements six anomaly detection algorithms along with other functionalities and contents that can be interesting in order to perform the best results.

For an easier understanding of this section, we divided it into four 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. Finally, section 2.4 describes a simple but effective technique for reducing false positives.

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 three belong to the window-based ones. From a stationary environment perspective, the first two algorithms can be use with stationary data, while the other four are suitable for using with non-stationary data. Table 1 shows the most important characteristics of each algorithm.

- *PEWMA* or *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* [**?**]. 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* [**?**]. 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. In the second phase, it checks the authenticity of the anomalies using the Kolmogorov-Simirnov test to reduce false alarms.
- *KNN-CAD* or *Conformal k-NN Anomaly Detector* [**?**]. 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* [**?**]. 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* [**?**]. 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 and storaged contexts determine if the instance is an anomaly.

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

Table 1: Features of the algorithms

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 (or online) processing used when complete dataset is not available. 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 [**?**] 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). To determine this label, a window is centered on the real anomaly. The detected anomaly is labeled as TP when falls inside the window and as FP when it falls 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 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** 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 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 [**?**] 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. one that detect all anomaly 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 $(-\infty, 100]$.

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 allows obtaining the number of instances used as a training set in NAB [**?**]. 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.

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 [**?**] 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.

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

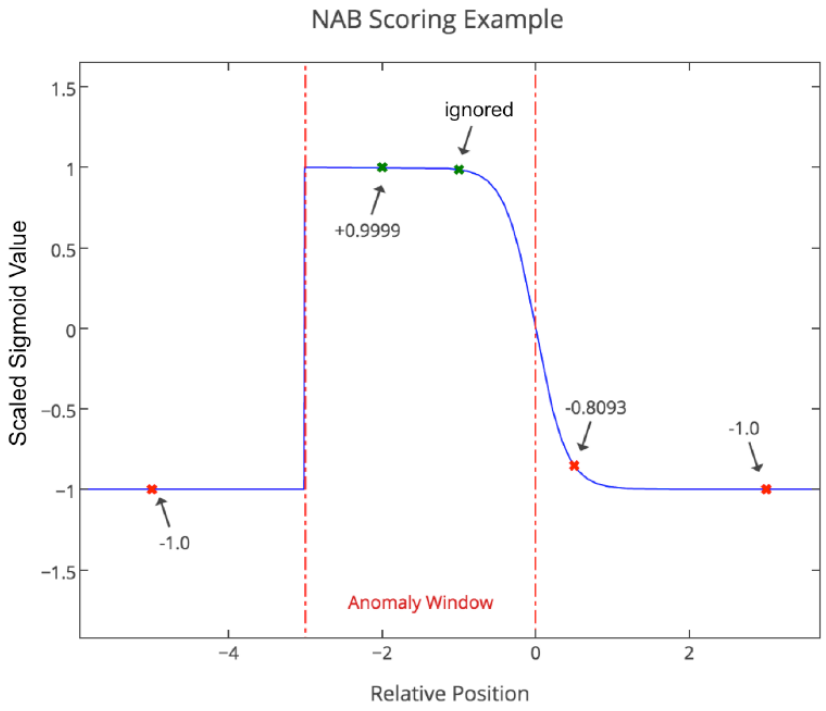


Figure 1: Scoring example for a sample anomaly window, where the values represent the scaled sigmoid function. The first point is an FP preceding the anomaly window (red dashed lines) and contributes -1.0 to the score. Within the window, we see two detections, and only count the earliest TP for the score.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

.

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

In this work, we have implemented the algorithms as were originally proposed by their authors. However, the results are not as expected or could be better because all of them give many false positives. KNN-CAD [**?**], KNN-LDCD [**?**] and CAD-OSE [**?**] authors suggested some techniques to reduce false positives.

These techniques can be only applied to above algorithms, so the rest of the algorithms selected in this paper are at a disadvantage. For this reason, we introduce and implement a simple algorithm to reduce false positives, ensuring that it can be applied in all the algorithms. The function's name in the package is **ReduceAnomalies**. Our algorithm is based on the real life situation where a time-lapse exists between an alarm is triggered and until action is taken. The time-lapse is modeled by a window sized by the user. This window is only focused on the first anomaly detected inside the window and all the others are discarded. Because the need to keep algorithms in the manner authors proposed, our function is now limited to classical processing and wherever a detector has been previously applied.

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. Finally, section 3.3 includes some examples to show how to use the package.

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 python libraries.

3.2 Documentation

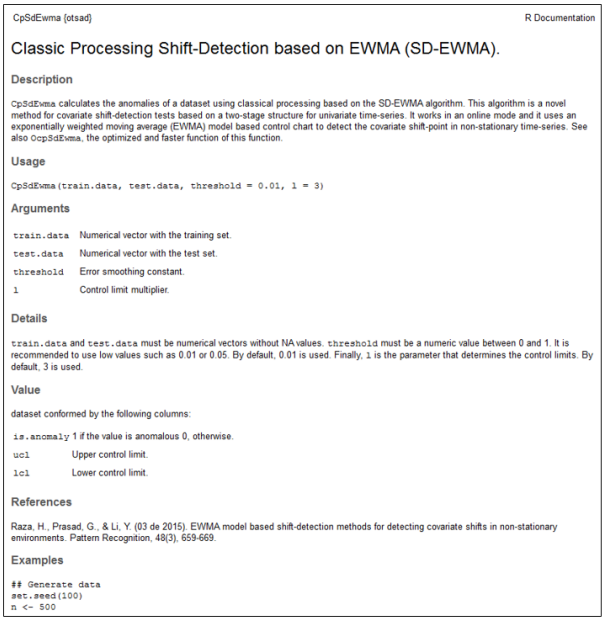
Considering that this vignette provides the user with an overview of the **otsad** package, it is also important to have access to the specific information of each of the available algorithms. This information can be checked in the documentation page corresponding to each algorithm. In all cases, the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

3.3 Examples

Here is an example of 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 3 by typing:

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

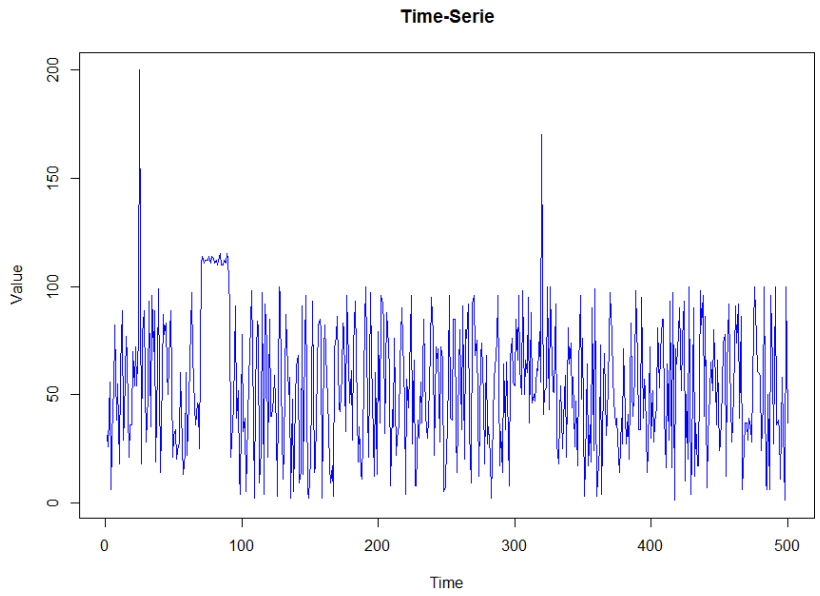


Figure 3: Our time-series visualization

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)
```

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 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.

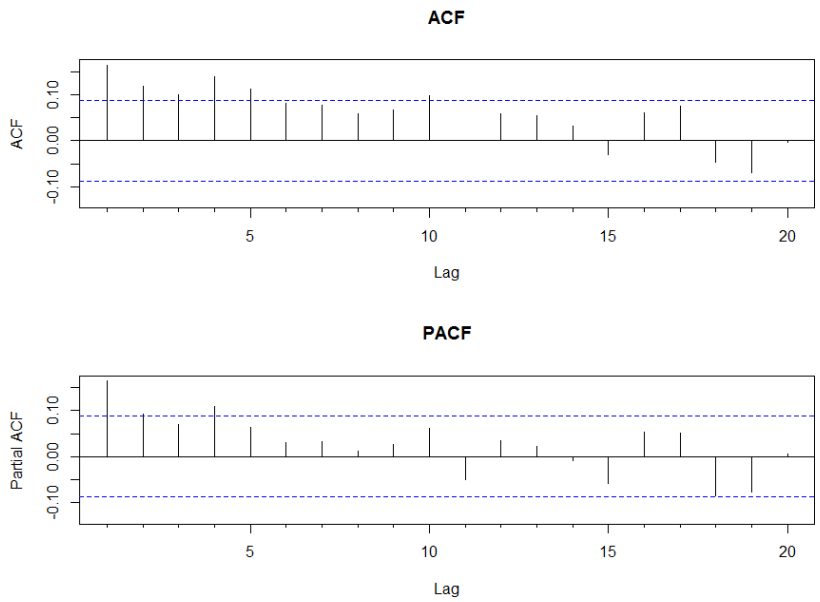


Figure 4: ACF and PACF graphs

```
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  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 plot the results (Figure 5) by writing the following code:

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

In Figure 5 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:

```
## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 250
numIter <- n%/%nread
```

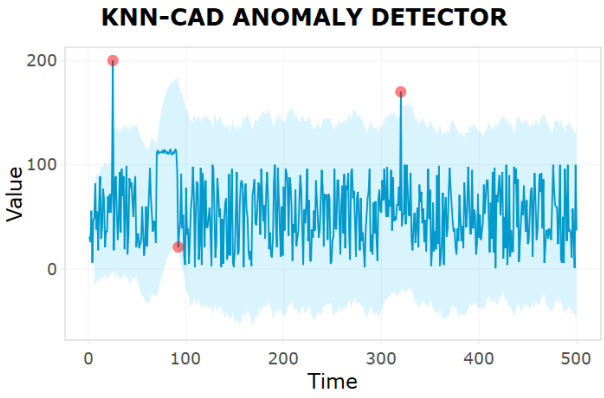


Figure 5: Sd-Ewma Anomaly detector results

```
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:

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

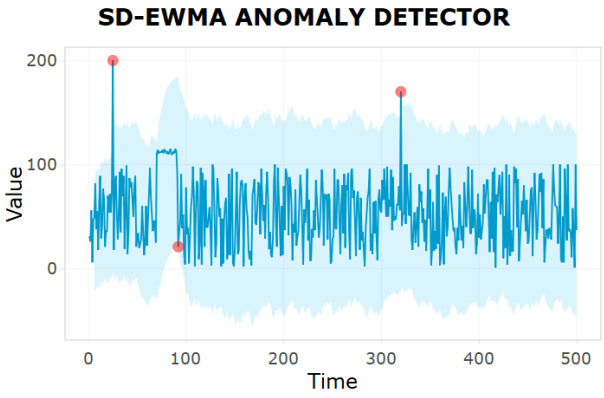


Figure 6: Online Sd-Ewma Anomaly detector results

The use of the Contextual Anomaly detector is similar to the use of the previous algorithms. 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 library with `install_bencode` function of `otsad` package.

```r
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")
```

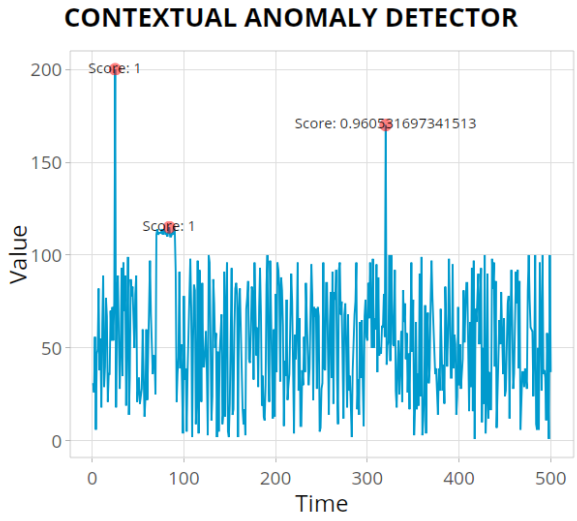


Figure 7: Contextual Anomaly Detector results

Finally we show a complete 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.

```r
# 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 8 we show a capture of it.

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

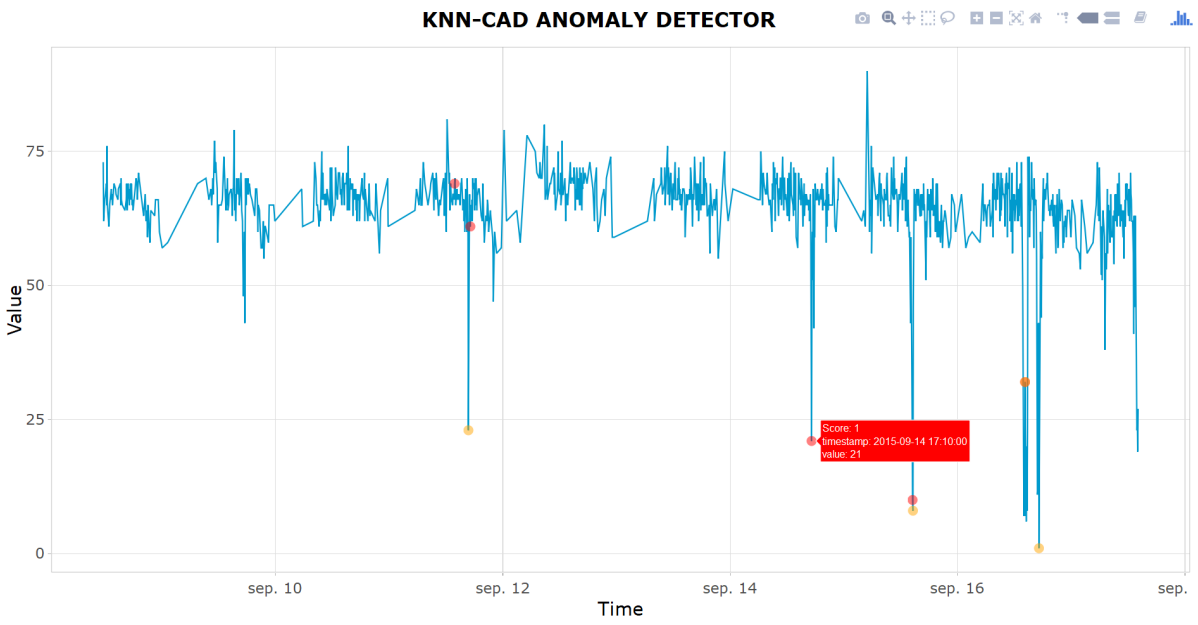


Figure 8: Anomalies detected in speed_7578 dataset by KNN_CAD detector.

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)

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"],
```

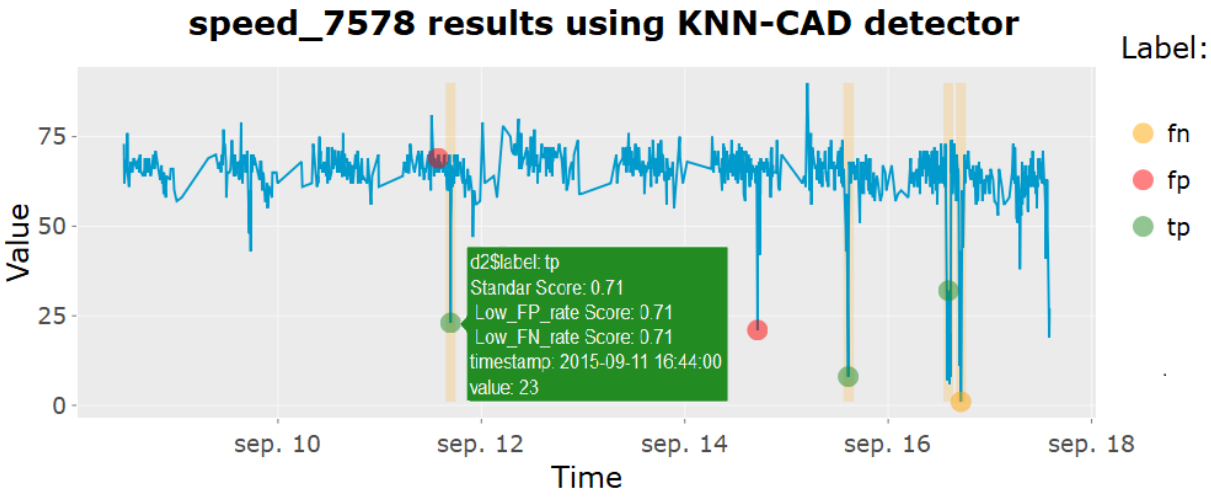


Figure 9: KNN-CAD detector measurement results for speed_7578 dataset.

```
  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 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.