Package 'otsad'

July 19, 2018

Type Package

Title Online Time Serie Anomaly Detectors

Version 0.1.0

Description Implements a set of online fault detector detectors
for time-series, called: EWMA, TSSD-EWMA, PEWMA and KNN-CAD and
KNN-LDCD. The first three algorithms belong to prediction-based
techniques and the last two belong to window-based techniques.
In addition, the SD-EWMA and PEWMA algorithms are algorithms
designed to work in stationary environments, while the other three
are algorithms designed to work in non-stationary environments.

Depends R (>= 3.4.0)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat,
stream,
knitr,
rmarkdown

Imports stats,
digest

VignetteBuilder knitr

R topics documented:

`ContextCrosser.CAD` *Context Crosser for Contextual Anomaly Detector (ContextualAD)*

Description

`ContextCrosser.CAD` is an auxiliar method that makes the necessary changes in the `context.operator` environment, including in the `fact.list` and the corresponding dictionary of facts the new facts.

Usage

```
ContextCrosser.CAD(left.or.right, fact.list, new.ctxt.flag = FALSE,
  potential.new.contexts = list(), context.operator)
```

Arguments

`left.or.right`	Situation from which the method have been called. It says if the new facts have been seen or if they are completely new.
`fact.list`	New fact list.
`new.ctxt.flag`	Flag that indicates if a new context is required.
`potential.new.contexts`	Potential new contexts that might be added.
`context.operator`	Environment with the current status for the context operator.

Details

`left.or.right` must be 0 or 1 to indicate left or right respectively.

Value

If `left.or.right`, number of new contex added tho the `context.operator`. Otherwise, the output from `UpdateContextsAndGetActive.CAD`.

References

Smirnov, M. (2018). CAD: Contextual Anomaly Detector. https://github.com/smirmik/CAD

ContextualAnomalyDetector
Contextual Anomaly Detector - Open Source (CAD)

Description

ContextualAnomalyDetector calculates the anomaly score of a dataset using the notion of contexts conformed by facts and provides probabilistic abnormality scores.

Usage

```
ContextualAnomalyDetector(data, rest.period = max(min(150, round(nrow(data) *
  0.03), 1)), max.left.semicontexts = 7, max.active.neurons = 15,
  num.norm.value.bits = 3, base.threshold = 0.75)
```

Arguments

data	Numerical vector with training and test dataset.
rest.period	Training period after an anomaly.
max.left.semicontexts	Number of semicontexts that should be maintained in memory.
max.active.neurons	Number of neurons of the model.
num.norm.value.bits	Granularity of the transformation into discrete values
base.threshold	Threshold to be considered an anomaly.

Details

data must be a numerical vector without NA values. threshold must be a numeric value between 0 and 1. If the anomaly score obtained for an observation is greater than the threshold, the observation will be considered abnormal.

Value

Anomaly score of data.

References

Smirnov, M. (2018). CAD: Contextual Anomaly Detector. https://github.com/smirmik/CAD

CpKnnCad *Classic processing KNN based Conformal Anomaly Detector (KNN-CAD)*

Description

CpKnnCad calculates the anomalies of a dataset using classical processing based on the KNN-CAD algorithm. KNN-CAD 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.

Usage

```
CpKnnCad(data, n.train, threshold = 1, l = 19, k = 27,
  ncm.type = "ICAD", reducefp = TRUE)
```

Arguments

data	Numerical vector with training and test dataset.
n.train	Number of points of the dataset that correspond to the training set.
threshold	Anomaly threshold.
l	Window length.
k	Number of neighbours to take into account.
ncm.type	Non Conformity Measure to use "ICAD" or "LDCD"
reducefp	If TRUE reduces false positives.

Details

data must be a numerical vector without NA values. threshold must be a numeric value between 0 and 1. If the anomaly score obtained for an observation is greater than the threshold, the observation will be considered abnormal. l must be a numerical value between 1 and 1/n; n being the length of the training data. Take into account that the value of l has a direct impact on the computational cost, so very high values will make the execution time longer. k parameter must be a numerical value less than the n.train value. ncm.type determines the non-conformity measurement to be used. ICAD calculates dissimilarity as the sum of the distances of the nearest k neighbours and LDCD as the average.

Value

dataset conformed by the following columns:

is.anomaly	1 if the value is anomalous, 0 otherwise.
anomaly.score	Probability of anomaly.

References

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

Examples

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

## Set parameters
params.KNN <- list(threshold = 1, n.train = 50, l = 19, k = 17)

## Calculate anomalies
result <- CpKnnCad(
  data = df$value,
  n.train = params.KNN$n.train,
  threshold = params.KNN$threshold,
  l = params.KNN$l,
  k = params.KNN$k,
  ncm.type = "ICAD",
  reducefp = TRUE
)

## Plot results
res <- cbind(df[(params.KNN$n.train + 1):n,],
             is.anomaly = result$is.anomaly[(params.KNN$n.train + 1):n])
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "KNN-CAD ANOMALY DETECTOR")
points(x = res[res$is.anomaly == TRUE, "timestamp"],
       y = res[res$is.anomaly == TRUE, "value"], pch=4, col="red", lwd = 2)
```

`CpPewma` *Classic Processing Probabilistic-EWMA (PEWMA).*

Description

`CpPewma` calculates the anomalies of a dataset using classical processing based on the PEWMA algorithm. 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. See also `OcpPewma`, the optimized and faster function of the this function.

Usage

```
CpPewma(data, n.train = 5, alpha0 = 0.8, beta = 0.3, l = 3)
```

Arguments

`data`	Numerical vector with training and test dataset.

n.train	Number of points of the dataset that correspond to the training set.
alpha0	Maximal weighting parameter.
beta	Weight placed on the probability of the given observation.
l	Control limit multiplier.

Details

data must be a numerical vector without NA values. alpha0 must be a numeric value where 0 < alpha0 < 1. If a faster adjustment to the initial shift is desirable, simply lowering alpha0 will suffice. beta is the weight placed on the probability of the given observation. It must be a numeric value where 0 <= beta <= 1. Note that if beta equals 0, PEWMA converges to a standard EWMA. 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

M. Carter, Kevin y W. Streilein. Probabilistic reasoning for streaming anomaly detection. 2012 IEEE Statistical Signal Processing Workshop (SSP), pp. 377-380, Aug 2012.

Examples

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

## Calculate anomalies
result <- CpPewma(
  data = df$value,
  n.train = 5,
  alpha0 = 0.8,
  beta = 0.1,
  l = 3
)

## Plot results
res <- cbind(df, result)
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "PEWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
```

```
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="green", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="green", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

CpSdEwma *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
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[320] <- 170
df <- data.frame(timestamp=1:n,value=x)

## Calculate anomalies
result <- CpSdEwma(
  train.data = df[1:5,"value"],
  test.data = df[6:n,"value"],
  threshold = 0.01,
  l = 3
)
res <- cbind(df[6:n,], result)
rownames(res) <- 1:(n-5)

## Plot results
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "SD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

`CpTsSdEwma` *Classic Processing Two-Stage Shift-Detection based on EWMA*

Description

`CpTsSdEwma` 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. This algorithm works in two phases. In the first phase, it detects anomalies using the SD-EWMA `CpSdEwma` algorithm. In the second phase, it checks the veracity of the anomalies using the Kolmogorov-Simirnov test to reduce false alarms. See also `OcpTsSdEwma`, the optimized and faster function of this function.

Usage

```
CpTsSdEwma(train.data, test.data, threshold, l = 3, m = 5)
```

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.
m	Length of the subsequences for applying the Kolmogorov-Smirnov test.

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. m is the length of the subsequences for applying the Kolmogorov-Smirnov test. By default, 5 is used. It should be noted that the last m values will not been verified because another m values are needed to be able to perform the verification.

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
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[320] <- 170
df=data.frame(timestamp=1:n,value=x)

## Calculate anomalies
result <- CpTsSdEwma(
  train.data = df[1:5,"value"],
  test.data = df[6:n,"value"],
  threshold = 0.01,
  l = 3,
  m = 20
)
res <- cbind(df[6:n,], result)
rownames(res) <- 1:(n-5)

## Plot results
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "TSSD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
```

```
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

`GetAnomalyScore.CAD` *Get Anomaly Score for Contextual Anomaly Detector*

Description

`GetAnomalyScore.CAD` calculates the anomaly score of a single datum using the notion of contexts conformed by facts and provides probabilistic abnormality scores.

Usage

```
GetAnomalyScore.CAD(datum, local.environment)
```

Arguments

`datum`	Integer vector of activated bits.
`local.environment`	Local environment with the variables of the Context Anomaly Detector.

Details

`datum` must be a numerical vector of the transformation made by `IntToBinarySens`.

Value

Anomaly score of `datum`.

References

Smirnov, M. (2018). CAD: Contextual Anomaly Detector. https://github.com/smirmik/CAD

`GetContextByFacts.CAD` *Get Context By Facts for Contextual Anomaly Detector*

Description

`GetContextByFacts.CAD` is an auxiliary method that determine by the complete facts list whether the context is already saved to the memory. If the context is not found the function immediately creates such. To optimize speed and volume of the occupied memory the contexts are divided into semi-contexts as several contexts can contain the same facts set in its left and right parts.

Usage

```
GetContextByFacts.CAD(context.list, context.operator, zero.level = 0)
```

Arguments

context.list	List of potentially new contexts
context.operator	Environment with the current status for the context operator.
zero.level	Flag indicating the context type in transmitted list

Value

Depending on the type of potentially new context transmitted as an input parameters the function returns either: a) flag indicating that the transmitted zero-level context is a new/existing one; or: b) number of the really new contexts that have been saved to the context memory.

References

Smirnov, M. (2018). CAD: Contextual Anomaly Detector. https://github.com/smirmik/CAD

IntToBinarySens *Int To Binary Sens*

Description

IntToBinarySens Auxiliar function that transforms integer to binary bins.

Usage

```
IntToBinarySens(x, num.norm.value.bits)
```

Arguments

x	input data
num.norm.value.bits	Expected number of binary bins

Value

Vector with activated numeric bits.

Examples

```
otsad:::IntToBinarySens(2, 3)
otsad:::IntToBinarySens(10, 5)
```

IpKnnCad *Incremental processing KNN based Conformal Anomaly Detector (KNN-CAD).*

Description

IpKnnCad allows the calculation of anomalies using SD-EWMA in an incremental processing mode. KNN-CAD 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.

Usage

```
IpKnnCad(data, n.train, threshold = 1, l = 19, k = 27,
  ncm.type = "ICAD", reducefp = TRUE, to.next.iteration = NULL)
```

Arguments

data	Numerical vector with training and test dataset.
n.train	Number of points of the dataset that correspond to the training set.
threshold	Anomaly threshold.
l	Window length.
k	Number of neighbours to take into account.
ncm.type	Non Conformity Measure to use "ICAD" or "LDCD"
reducefp	If TRUE reduces false positives.
to.next.iteration	list with the necessary parameters to execute in the next iteration.

Details

data must be a numerical vector without NA values. threshold must be a numeric value between 0 and 1. If the anomaly score obtained for an observation is greater than the threshold, the observation will be considered abnormal. l must be a numerical value between 1 and 1/n; n being the length of the training data. Take into account that the value of l has a direct impact on the computational cost, so very high values will make the execution time longer. k parameter must be a numerical value less than the n.train value. ncm.type determines the non-conformity measurement to be used. ICAD calculates dissimilarity as the sum of the distances of the nearest k neighbours and LDCD as the average. to.next.iteration is the last result returned by some previous execution of this algorithm. The first time the algorithm is executed its value is NULL. However, to run a new batch of data without having to include it in the old dataset and restart the process, this parameter returned by the last run is only needed.

This algorithm can be used for both classical and incremental processing. It should be noted that in case of having a finite dataset, the CpKnnCad algorithm is faster. Incremental processing can be used in two ways. 1) Processing all available data and saving calibration.alpha and last.data for future runs with new data. 2) Using the stream library for when there is much data and it does not fit into the memory. An example has been made for this use case.

Value

dataset conformed by the following columns:

`is.anomaly` 1 if the value is anomalous 0, otherwise.

`anomaly.score` Probability of anomaly.

`to.next.iteration`
Last result returned by the algorithm. It is a list containing the following items.

- `training.set` Last training set values used in the previous iteration and required for the next run.
- `calibration.set` Last calibration set values used in the previous iteration and required for the next run.
- `sigma` Last covariance matrix calculated in the previous iteration and required for the next run.
- `alphas` Last calibration alpha values calculated in the previous iteration and required for the next run.
- `last.data` Last values of the dataset converted into multi-dimensional vectors..
- `pred` Parameter that is used to reduce false positives. Only necessary in case of reducefp is TRUE.
- `record.count` Number of observations that have been processed up to the last iteration.

References

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

Examples

```
## EXAMPLE 1: ----------------------
## It can be used in the same way as with CpKnnCad passing the whole dataset as
## an argument.

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

## Set parameters
params.KNN <- list(threshold = 1, n.train = 50, l = 19, k = 17)

## Calculate anomalies
result <- IpKnnCad(
  data = df$value,
  n.train = params.KNN$n.train,
  threshold = params.KNN$threshold,
  l = params.KNN$l,
  k = params.KNN$k,
  ncm.type = "ICAD",
```

```
  reducefp = TRUE
)

## Plot results
res <- cbind(df[(params.KNN$n.train + 1):n,],
  is.anomaly = result$is.anomaly[(params.KNN$n.train + 1):n])
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "KNN-CAD ANOMALY DETECTOR")
points(x = res[res$is.anomaly == TRUE, "timestamp"],
  y = res[res$is.anomaly == TRUE, "value"], pch=4, col="red", lwd = 2)

## EXAMPLE 2: ----------------------
## You can use it in an incremental way. This is an example using the stream
## library. This library allows the simulation of streaming operation.

# install.packages("stream")
library("stream")

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

## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 100
numIter <- n%/%nread

## Set parameters
params.KNN <- list(threshold = 1, n.train = 50, l = 19, k = 17)

## Calculate anomalies
for(i in 1:numIter) {
  # read new data
  newRow <- get_points(dsd_df, n = nread, outofpoints = "ignore")
  # calculate if it's an anomaly
  last.res <- IpKnnCad(
    data = newRow$value,
    n.train = params.KNN$n.train,
    threshold = params.KNN$threshold,
    l = params.KNN$l,
    k = params.KNN$k,
    ncm.type = "ICAD",
    reducefp = TRUE,
    to.next.iteration = last.res$to.next.iteration
  )
  # prepare the result
  if(!is.null(last.res$is.anomaly)){
    res <- rbind(res, cbind(newRow, is.anomaly = last.res$is.anomaly))
  }
```

```
}

## Plot results
res <- res[(params.KNN$n.train + 1):n,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "KNN-CAD ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
  y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
```

IpPewma *Incremental Processing Probabilistic-EWMA (PEWMA).*

Description

IpPewma allows the calculation of anomalies using PEWMA in an incremental processing mode. See also OipPewma, the optimized and faster function of this function 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.

Usage

```
IpPewma(data, n.train = 5, alpha0 = 0.8, beta = 0, l = 3,
  last.res = NULL)
```

Arguments

data	Numerical vector with training and test dataset.
n.train	Number of points of the dataset that correspond to the training set.
alpha0	Maximal weighting parameter.
beta	Weight placed on the probability of the given observation.
l	Control limit multiplier.
last.res	Last result returned by the algorithm.

Details

data must be a numerical vector without NA values. alpha0 must be a numeric value where 0 < alpha0 < 1. If a faster adjustment to the initial shift is desirable, simply lowering alpha0 will suffice. beta is the weight placed on the probability of the given observation. it must be a numeric value where 0 <= beta <= 1. Note that beta equals 0, PEWMA converges to a standard EWMA. Finally l is the parameter that determines the control limits. By default, 3 is used. last.res is the last result returned by some previous execution of this algorithm. The first time the algorithm is executed its value is NULL. However, to run a new batch of data without having to include it in the old dataset and restart the process, the two parameters returned by the last run are only needed.

This algorithm can be used for both classical and incremental processing. It should be noted that in case of having a finite dataset the CpPewma or OcpPewma algorithms are faster. Incremental processing can be used in two ways. 1) Processing all available data and saving last.res for future runs in which there is new data. 2) Using the stream library for when there is too much data and it does not fit into the memory. An example has been made for this use case.

Value

A list of the following items.

result dataset conformed by the following columns.

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

last.res Last result returned by the algorithm. Is a dataset containing the parameters calculated in the last iteration and necessary for the next one.

References

M. Carter, Kevin y W. Streilein. Probabilistic reasoning for streaming anomaly detection. 2012 IEEE Statistical Signal Processing Workshop (SSP), pp. 377-380, Aug 2012.

Examples

```
## EXAMPLE 1: ----------------------
## It can be used in the same way as with CpPewma passing the whole dataset as
## an argument.

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

## Calculate anomalies
result <- IpPewma(
  data = df$value,
  alpha0 = 0.8,
  beta = 0,
  n.train = 5,
  l = 3,
  last.res = NULL
)
res <- cbind(df, result$result)

## Plot results
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "PEWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
```

```
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="green", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="green", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")

## EXAMPLE 2: ----------------------
## You can use it in an incremental way. This is an example using the stream
## library. This library allows the simulation of streaming operation.

# install.packages("stream")
library("stream")

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

## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 100
numIter <- n%/%nread

## Calculate anomalies
for(i in 1:numIter) {
  # read new data
  newRow <- get_points(dsd_df, n = nread, outofpoints = "ignore")
  # calculate if it's an anomaly
  last.res <- IpPewma(
    data = newRow$value,
    n.train = 5,
    alpha0 = 0.8,
    beta = 0,
    l = 3,
    last.res = last.res$last.res
  )
  # prepare the result
  if(!is.null(last.res$result)){
    res <- rbind(res, cbind(newRow, last.res$result))
  }
}

## Plot results
res <- res[6:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "PEWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
```

```
plot(x = res$timestamp, y = res$ucl, type="l", col="green", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="green", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

`IpSdEwma` *Incremental Processing Shift-Detection based on EWMA (SD-EWMA).*

Description

`IpSdEwma` allows the calculation of anomalies using SD-EWMA in an incremental processing mode. See also `OipSdEwma`, the optimized and faster function of this function SD-EWMA 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.

Usage

```
IpSdEwma(data, n.train, threshold = 0.01, l = 3, last.res = NULL)
```

Arguments

`data`	Numerical vector with training and test dataset.
`n.train`	Number of points of the dataset that correspond to the training set.
`threshold`	Error smoothing constant.
`l`	Control limit multiplier.
`last.res`	Last result returned by the algorithm.

Details

`data` must be a numerical vector 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. `l` is the parameter that determines the control limits. By default, 3 is used. Finally `last.res` is the last result returned by some previous execution of this algorithm. The first time the algorithm is executed its value is NULL. However, to run a new batch of data without having to include it in the old dataset and restart the process, the two parameters returned by the last run are only needed.

This algorithm can be used for both classical and incremental processing. It should be noted that in case of having a finite dataset the `CpSdEwma` or `OcpSdEwma` algorithms are faster. Incremental processing can be used in two ways. 1) Processing all available data and saving `last.res` for future runs in which there is new data. 2) Using the stream library for when there is too much data and it does not fit into memory. An example has been made for this use case.

Value

A list of the following items.

`result` dataset conformed by the following columns.

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

`last.res` Last result returned by the algorithm. Is a dataset containing the parameters calculated in the last iteration and necessary for the next one.

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

```
## EXAMPLE 1: ----------------------
## It can be used in the same way as with CpSdEwma passing the whole dataset as
## an argument.

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

## Calculate anomalies
result <- IpSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3
)
res <- cbind(df[6:n,], result$result)
rownames(res) <- 1:(n-5)

## Plot results
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "SD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

```
## EXAMPLE 2: ----------------------
## You can use it in an incremental way. This is an example using the stream
## library. This library allows the simulation of streaming operation.

# install.packages("stream")
library("stream")

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

## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 100
numIter <- n%/%nread

## Calculate anomalies
for(i in 1:numIter) {
  # read new data
  newRow <- get_points(dsd_df, n = nread, outofpoints = "ignore")
  # 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$result)){
    res <- rbind(res, cbind(newRow[(nread-nrow(last.res$result)+1):nread,],
                 last.res$result))
  }
}
print(res)

## Plot results
n.train <- 5
rownames(res) <- 1:(n-n.train)
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "SD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

IpTsSdEwma *Incremental Processing Two-Stage Shift-Detection based on EWMA*

Description

IpTsSdEwma allows the calculation of anomalies using TSSD-EWMA in an incremental processing mode. See also OipTsSdEwma, the optimized and faster function of this function. This algorithm is a novel method for covariate shift-detection tests based on a two-stage structure for univariate time-series. TSSD-EWMA works in two phases. In the first phase, it detects anomalies using the SD-EWMA CpSdEwma algorithm. In the second phase, it checks the veracity of the anomalies using the Kolmogorov-Simirnov test to reduce false alarms.

Usage

```
IpTsSdEwma(data, n.train, threshold, l = 3, m = 5,
  to.next.iteration = list(last.res = NULL, to.check = NULL, last.m = NULL))
```

Arguments

data	Numerical vector with training and test dataset.
n.train	Number of points of the dataset that correspond to the training set.
threshold	Error smoothing constant.
l	Control limit multiplier.
m	Length of the subsequences for applying the Kolmogorov-Smirnov test.
to.next.iteration	list with the necessary parameters to execute in the next iteration

Details

data must be a numerical vector 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. m is the length of the subsequences for applying the Kolmogorov-Smirnov test. By default, 5 is used. It should be noted that the last m values have not been verified because you need other m values to be able to perform the verification. Finally to.next.iteration is the last result returned by some previous execution of this algorithm. The first time the algorithm is executed its value is NULL. However, to run a new batch of data without having to include it in the old dataset and restart the process, the two parameters returned by the last run are only needed.

Value

A list of the following items.

last.data.checked
: Anomaly results of the last m results of the previous iteration. dataset conformed by the following columns.

- is.anomaly 1 if the value is anomalous 0 otherwise.

- ucl Upper control limit.
- lcl Lower control limit.

checked.results
: Anomaly results of the dataset excluding the last m values because they could not be verified. dataset conformed by the following columns: is.anomaly, ucl, lcl.

to.next.iteration
: Last result returned by the algorithm. It is a list containing the following items.

- last.res Last result returned by the aplicaction of SD-EWMA function with the calculations of the parameters of the last run . These are necessary for the next run.
- to.check Subsequence of the last remaining unchecked values to be checked in the next iteration. dataset conformed by the following columns: is.anomaly, ucl, lcl, value.
- last.m Subsequence of the m values prior to the to.check subsecuence necessary to verify the values in to.check.

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

```
## EXAMPLE 1: ----------------------
## It can be used in the same way as with CpTsSdEwma passing the whole dataset
## as an argument.

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

## Calculate anomalies
result <- IpTsSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3,
  m = 20
)
res <- cbind(df[6:n,], rbind(result$last.data.checked, result$checked.results,
             result$to.next.iteration$to.check[, -4]))
rownames(res) <- 1:(n-5)

## Plot results
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "TSSD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
```

```
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")

## EXAMPLE 2: ----------------------
## You can use it in an incremental way. This is an example using the stream
## library. This library allows the simulation of streaming operation.

# install.packages("stream")
library("stream")

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

## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 100
numIter <- n%/%nread
m <- 20

## Calculate anomalies
for(i in 1:numIter) {
  # read new data
  newRow <- get_points(dsd_df, n = nread, outofpoints = "ignore")
  # calculate if it's an anomaly
  last.res <- IpTsSdEwma(
    data = newRow$value,
    n.train = 5,
    threshold = 0.01,
    l = 3,
    m = m,
    to.next.iteration = last.res$to.next.iteration
  )
  # prepare result
  if(!is.null(last.res$last.data.checked)){
    res <- rbind(res, cbind(last.timestamp, last.res$last.data.checked))
  }
  if(!is.null(last.res$checked.results)){
    init <- nread - (nrow(last.res$checked.results) +
            nrow(last.res$to.next.iteration$to.check)) + 1
    end <- init + nrow(last.res$checked.results) - 1
    res <- rbind(res, cbind(newRow[init:end,], last.res$checked.results))
  }
  if(i == numIter){
    res <- rbind(res,
```

```
            cbind(timestamp = newRow[
                              (nread - nrow(last.res$to.next.iteration$to.check)
                              + 1):nread, "timestamp"],
                  last.res$to.next.iteration$to.check))
  }
  last.timestamp <- newRow[(nread-m+1):nread,]
}

## Plot results
rownames(res) <- 1:(n-5)
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "TSSD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

`OcpPewma` *Optimized Classic Processing Probabilistic-EWMA (PEWMA).*

Description

`OcpPewma` calculates the anomalies of a dataset using an optimized version of classical processing Probabilistic-EWMA algorithm. It Is an optimized implementation of the `CpPewma` algorithm using environmental variables. It has been shown that in long datasets it can reduce runtime by up to 50%. TThis 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.

Usage

```
OcpPewma(data, alpha0 = 0.2, beta = 0, n.train = 5, l = 3)
```

Arguments

`data`	Numerical vector with training and test datasets.
`alpha0`	Maximal weighting parameter.
`beta`	Weight placed on the probability of the given observation.
`n.train`	Number of points of the dataset that correspond to the training set.
`l`	Control limit multiplier.

Details

`data` must be a numerical vector without NA values. `alpha0` must be a numeric value where 0 < `alpha0` < 1. If a faster adjustment to the initial shift is desirable, simply lowering `alpha0` will suffice. `beta` is the weight placed on the probability of the given observation. It must be a numeric value where 0 <= `beta` <= 1. Note that if `beta` equals 0, PEWMA converges to a standard EWMA. 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

M. Carter, Kevin y W. Streilein. Probabilistic reasoning for streaming anomaly detection. 2012 IEEE Statistical Signal Processing Workshop (SSP), pp. 377-380, Aug 2012.

Examples

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

## Calculate anomalies
result <- OcpPewma(
  data = df$value,
  n.train = 5,
  alpha0 = 0.8,
  beta = 0.1,
  l = 3
)

## Plot results
res <- cbind(df, result)
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "PEWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
  y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="green", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="green", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
```

`OcpSdEwma` *Optimized Classic Processing Shift-Detection based on EWMA (SD-EWMA).*

Description

OcpSdEwma calculates the anomalies of a dataset using an optimized version of classical processing based on the SD-EWMA algorithm. It is an optimized implementation of the CpSdEwma algorithm using environment variables. It has been shown that in long datasets it can reduce runtime by up to 50%. SD-EWMA 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.

Usage

```
OcpSdEwma(train.data, test.data, threshold, 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
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[320] <- 170
df <- data.frame(timestamp = 1:n, value = x)

## Calculate anomalies
result <- OcpSdEwma(
```

```
  train.data = df[1:5,"value"],
  test.data = df[6:n,"value"],
  threshold = 0.01,
  l = 3
)
res <- cbind(df[6:n,], result)
rownames(res) <- 1:(n-5)

## Plot results
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "SD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

`OcpTsSdEwma` *Optimized Classic Processing Two-Stage Shift-Detection based on EWMA*

Description

`OcpTsSdEwma` calculates the anomalies of a dataset using an optimized verision of classical processing based on the SD-EWMA algorithm. It is an optimized implementation of the `CpTsSdEwma` algorithm using environment variables. It has been shown that in long datasets it can reduce runtime by up to 50%. This algorithm is a novel method for covariate shift-detection tests based on a two-stage structure for univariate time-series. This algorithm works in two phases. In the first phase, it detects anomalies using the SD-EWMA `CpSdEwma` algorithm. In the second phase, it checks the veracity of the anomalies using the Kolmogorov-Simirnov test to reduce false alarms.

Usage

```
OcpTsSdEwma(train.data, test.data, threshold, l = 3, m = 5)
```

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.
`m`	Length of the subsequences for applying the Kolmogorov-Smirnov test.

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. `m` is the length of the subsequences for applying the Kolmogorov-Smirnov test. By default, 5 is used. It should be noted that the last `m` values will not been verified because another `m` values are needed to be able to perform the verification.

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
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[320] <- 170
df=data.frame(timestamp=1:n,value=x)

## Calculate anomalies
result <- OcpTsSdEwma(
  train.data = df[1:5,"value"],
  test.data = df[6:n,"value"],
  threshold = 0.01,
  l = 3,
  m = 20
)
res <- cbind(df[6:n,], result)
rownames(res) <- 1:(n-5)

## Plot results
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "TSSD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
  y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
```

```
ylim = y.limits, xlab = "", ylab = "")
```

OipPewma *Optimized Incremental Processing Probabilistic-EWMA (PEWMA).*

Description

OipPewma is the optimized implementation of the IpPewma function using environmental variables. It has been shown that in long datasets it can reduce runtime by up to 50%. This function allows the calculation of anomalies using PEWMA in an incremental processing mode. 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.

Usage

```
OipPewma(data, alpha0 = 0.2, beta = 0, n.train = 5, l = 3,
  last.res = NULL)
```

Arguments

data	Numerical vector with training and test dataset.
alpha0	Maximal weighting parameter.
beta	Weight placed on the probability of the given observation.
n.train	Number of points of the dataset that correspond to the training set.
l	Control limit multiplier.
last.res	Last result returned by the algorithm.

Details

data must be a numerical vector without NA values. alpha0 must be a numeric value where 0 < alpha0 < 1. If a faster adjustment to the initial shift is desirable, simply lowering alpha0 will suffice. beta is the weight placed on the probability of the given observation. it must be a numeric value where 0 <= beta <= 1. Note that beta equals 0, PEWMA converges to a standard EWMA. Finally l is the parameter that determines the control limits. By default, 3 is used. last.res is the last result returned by some previous execution of this algorithm. The first time the algorithm is executed its value is NULL. However, to run a new batch of data without having to include it in the old dataset and restart the process, the two parameters returned by the last run are only needed.

This algorithm can be used for both classical and incremental processing. It should be noted that in case of having a finite dataset the CpPewma or OcpPewma algorithms are faster. Incremental processing can be used in two ways. 1) Processing all available data and saving last.res for future runs in which there is new data. 2) Using the stream library for when there is too much data and it does not fit into the memory. An example has been made for this use case.

Value

A list of the following items.

`result` dataset conformed by the following columns.

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

`last.res` Last result returned by the algorithm. Is a dataset containing the parameters calculated in the last iteration and necessary for the next one.

References

M. Carter, Kevin y W. Streilein. Probabilistic reasoning for streaming anomaly detection. 2012 IEEE Statistical Signal Processing Workshop (SSP), pp. 377-380, Aug 2012.

Examples

```
## EXAMPLE 1: ----------------------
## It can be used in the same way as with OcpPewma passing the whole dataset as
## an argument.

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

## Calculate anomalies
result <- OipPewma(
  data = df$value,
  alpha0 = 0.8,
  beta = 0,
  n.train = 5,
  l = 3,
  last.res = NULL
)
res <- cbind(df, result$result)

## Plot results
res <- res[6:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "PEWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
  y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="green", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="green", xaxt="n",
```

```
  ylim = y.limits, xlab = "", ylab = "")

## EXAMPLE 2: ----------------------
## You can use it in an incremental way. This is an example using the stream
## library. This library allows the simulation of streaming operation.

# install.packages("stream")
library("stream")

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

## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 100
numIter <- n%/%nread

## Calculate anomalies
for(i in 1:numIter) {
  # read new data
  newRow <- get_points(dsd_df, n = nread, outofpoints = "ignore")
  # calculate if it's an anomaly
  last.res <- OipPewma(
    data = newRow$value,
    n.train = 5,
    alpha0 = 0.8,
    beta = 0,
    l = 3,
    last.res = last.res$last.res
  )
  # prepare the result
  if(!is.null(last.res$result)){
    res <- rbind(res, cbind(newRow, last.res$result))
  }
}

## Plot results
res <- res[6:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "PEWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
  y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="green", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="green", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
```

`OipSdEwma` *Optimized Incremental Processing Shift-Detection based on EWMA (SD-EWMA).*

Description

`OipSdEwma` is the optimized implementation of the `IpSdEwma` function using environmental variables. This function allows the calculation of anomalies using SD-EWMA alogrithm in an incremental processing mode. It has been shown that in long datasets it can reduce runtime by up to 50%. SD-EWMA 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.

Usage

```
OipSdEwma(data, n.train, threshold, l = 3, last.res = NULL)
```

Arguments

`data`	Numerical vector with training and test datasets.
`n.train`	Number of points of the dataset that correspond to the training set.
`threshold`	Error smoothing constant.
`l`	Control limit multiplier.
`last.res`	Last result returned by the algorithm.

Details

`data` must be a numerical vector 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. `l` is the parameter that determines the control limits. By default, 3 is used. Finally `last.res` is the last result returned by some previous execution of this algorithm. The first time the algorithm is executed its value is NULL. However, to run a new batch of data without having to include it in the old dataset and restart the process, the two parameters returned by the last run are only needed.

This algorithm can be used for both classical and incremental processing. It should be noted that in case of having a finite dataset the `CpSdEwma` or `OcpSdEwma` algorithms are faster. Incremental processing can be used in two ways. 1) Processing all available data and saving `last.res` for future runs in which there is new data. 2) Using the stream library for when there is too much data and it does not fit into memory. An example has been made for this use case.

Value

A list of the following items.

`result` dataset conformed by the following columns.

- `is.anomaly` 1 if the value is anomalous 0, otherwise.

- ucl Upper control limit.
- lcl Lower control limit.

last.res Last result returned by the algorithm. Is a dataset containing the parameters calculated in the last iteration and necessary for the next one.

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

```
## EXAMPLE 1: ----------------------
## It can be used in the same way as with OcpSdEwma passing the whole dataset as
## an argument.

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

## Calculate anomalies
result <- OipSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3
)
res <- cbind(df[6:n,], result$result)
rownames(res) <- 1:(n-5)

## Plot results
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
     xlab = "timestamp", ylab = "value", main = "SD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")

## EXAMPLE 2: ----------------------
## You can use it in an incremental way. This is an example using the stream
## library. This library allows the simulation of streaming operation.

# install.packages("stream")
library("stream")
```

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

## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 100
numIter <- n%/%nread

## Calculate anomalies
for(i in 1:numIter) {
  # read new data
  newRow <- get_points(dsd_df, n = nread, outofpoints = "ignore")
  # calculate if it's an anomaly
  last.res <- OipSdEwma(
    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$result)){
    res <- rbind(res, cbind(newRow[(nread-nrow(last.res$result)+1):nread,],
                last.res$result))
  }
}
print(res)
# plot
n.train <- 5
rownames(res) <- 1:(n-n.train)
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l",
     ylim = y.limits, xlab = "timestamp", ylab = "value",
     main = "SD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
       y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
     ylim = y.limits, xlab = "", ylab = "")
```

`OipTsSdEwma` *Optimized Incremental Processing Two-Stage Shift-Detection based on EWMA*

Description

`OipTsSdEwma` is the optimized implementation of the `IpTsSdEwma` function using environmental variables. This function allows the calculation of anomalies using TSSD-EWMA in an incremental processing mode. It has been shown that in long datasets it can reduce runtime by up to 50%. This algorithm is a novel method for covariate shift-detection tests based on a two-stage structure for univariate time-series. TSSD-EWMA works in two phases. In the first phase, it detects anomalies using the SD-EWMA `CpSdEwma` algorithm. In the second phase, it checks the veracity of the anomalies using the Kolmogorov-Simirnov test to reduce false alarms.

Usage

```
OipTsSdEwma(data, n.train, threshold, l = 3, m = 5,
  to.next.iteration = list(last.res = NULL, to.check = NULL, last.m = NULL))
```

Arguments

`data`	Numerical vector with training and test dataset.
`n.train`	Number of points of the dataset that correspond to the training set.
`threshold`	Error smoothing constant.
`l`	Control limit multiplier.
`m`	Length of the subsequences for applying the Kolmogorov-Smirnov test.
`to.next.iteration`	list with the necessary parameters to execute in the next iteration

Details

`data` must be a numerical vector 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. `m` is the length of the subsequences for applying the Kolmogorov-Smirnov test. By default, 5 is used. It should be noted that the last m values have not been verified because you need other m values to be able to perform the verification. Finally `to.next.iteration` is the last result returned by some previous execution of this algorithm. The first time the algorithm is executed its value is NULL. However, to run a new batch of data without having to include it in the old dataset and restart the process, the two parameters returned by the last run are only needed.

Value

A list of the following items.

`last.data.checked`
: Anomaly results of the last `m` results of the previous iteration. dataset conformed by the following columns.

- `is.anomaly` 1 if the value is anomalous 0 otherwise.
- `ucl` Upper control limit.

- lcl Lower control limit.

checked.results
: Anomaly results of the dataset excluding the last m values because they could not be verified. dataset conformed by the following columns: is.anomaly, ucl, lcl.

to.next.iteration
: Last result returned by the algorithm. It is a list containing the following items.

- last.res Last result returned by the aplicaction of SD-EWMA function with the calculations of the parameters of the last run . These are necessary for the next run.
- to.check Subsequence of the last remaining unchecked values to be checked in the next iteration. dataset conformed by the following columns: is.anomaly, ucl, lcl, value.
- last.m Subsequence of the m values prior to the to.check subsecuence necessary to verify the values in to.check.

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

```
## EXAMPLE 1: ----------------------
## It can be used in the same way as with OcpTsSdEwma passing the whole dataset
## as an argument.

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

## Calculate anomalies
result <- OipTsSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3,
  m = 20
)
res <- cbind(df[6:n,], rbind(result$last.data.checked, result$checked.results,
  result$to.next.iteration$to.check[, -4]))
rownames(res) <- 1:(n-5)

## Plot results
res <- res[1:500,]
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "TSSD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
  y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
```

```
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")

## EXAMPLE 2: ----------------------
## You can use it in an incremental way. This is an example using the stream
## library. This library allows the simulation of streaming operation.

# install.packages("stream")
library("stream")

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

## Initialize parameters for the loop
last.res <- NULL
res <- NULL
nread <- 100
numIter <- n%/%nread
m <- 20

## Calculate anomalies
for(i in 1:numIter) {
  # read new data
  newRow <- get_points(dsd_df, n = nread, outofpoints = "ignore")
  # calculate if it's an anomaly
  last.res <- OipTsSdEwma(
    data = newRow$value,
    n.train = 5,
    threshold = 0.01,
    l = 3,
    m = m,
    to.next.iteration = last.res$to.next.iteration
  )
  # prepare result
  if(!is.null(last.res$last.data.checked)){
    res <- rbind(res, cbind(last.timestamp, last.res$last.data.checked))
  }
  if(!is.null(last.res$checked.results)){
    init <- nread - (nrow(last.res$checked.results) +
        nrow(last.res$to.next.iteration$to.check)) + 1
    end <- init + nrow(last.res$checked.results) - 1
    res <- rbind(res, cbind(newRow[init:end,], last.res$checked.results))
  }
  if(i == numIter){
    res <- rbind(res,
      cbind(timestamp = newRow[
        (nread - nrow(last.res$to.next.iteration$to.check)
```

```
          + 1):nread, "timestamp"],
        last.res$to.next.iteration$to.check))
  }
  last.timestamp <- newRow[(nread-m+1):nread,]
}

## Plot results
rownames(res) <- 1:(n-5)
y.limits <- c(-150,250)
plot(x = res$timestamp, y = res$value, type = "l", ylim = y.limits,
  xlab = "timestamp", ylab = "value", main = "TSSD-EWMA ANOMALY DETECTOR")
points(x = res[res$is.anomaly == 1, "timestamp"],
  y = res[res$is.anomaly == 1, "value"], pch=4, col="red", lwd = 2)
par(new=TRUE)
plot(x = res$timestamp, y = res$ucl, type="l", col="red", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
par(new=TRUE)
plot(x = res$timestamp, y = res$lcl, type="l", col="red", xaxt="n",
  ylim = y.limits, xlab = "", ylab = "")
```

`Step.CAD` *Step operator for Contextual Anomaly Detector (ContextualAD)*

Description

`Step.CAD` is an auxiliar method that makes the necessary changes in the local environment of the anomaly detector

Usage

```
Step.CAD(datum, local.environment)
```

Arguments

`datum`	New datum in form of numeric vector corresponding to the bits that have been activated.
`local.environment`	Local environment with the variables of the Contextual Anomaly Detector.

Details

`datum` must be a numeric vector.

Value

If `left.or.right`, number of new contex added tho the `context.operator`. Otherwise, the output from `UpdateContextsAndGetActive.CAD`.

References

Smirnov, M. (2018). CAD: Contextual Anomaly Detector. https://github.com/smirmik/CAD

`UpdateContextsAndGetActive.CAD` *Update Contexts and Get Active - Contextual Anomaly Detector*

Description

`UpdateContextsAndGetActive.CAD` is an auxiliar method for the Contextual Anomaly Detector that reviews the list of previously selected left semi-contexts, creates the list of potentially new contexts resulted from intersection between zero-level contexts, determines the contexts that coincide with the input data and require activation.

Usage

```
UpdateContextsAndGetActive.CAD(new.ctxt.flag, context.operator)
```

Arguments

`new.ctxt.flag`	flag indicating that a new zero-level context is not recorded at the current step, which means that all contexts already exist and there is no need to create new ones.
`context.operator`	Environment with the current status for the context operator.

Value

List of:

`activeContexts`	List of identifiers of the contexts which completely coincide with the input stream, should be considered active and be recorded to the input stream of "neurons"
`potentialNewContextList`	List of contexts based on intersection between the left and the right zero-level semi-contexts, which are potentially new contexts requiring saving to the context memory

References

Smirnov, M. (2018). CAD: Contextual Anomaly Detector. https://github.com/smirmik/CAD

Index