Package 'otsad'

April 1, 2019

Type Package

Title Online Time Series Anomaly Detectors

Version 0.1.0

Description Implements a set of online fault detectors for time-series, called: PEWMA see M. Carter et al. (2012) <doi:10.1109/SSP.2012.6319708>, SD-EWMA and TSSD-EWMA see H. Raza et al. (2015) <doi:10.1016/j.patcog.2014.07.028>, KNN-CAD see E. Burnaev et al. (2016) <arXiv:1608.04585>, KNN-LDCD see V. Ishimtsev et al. (2017) <arXiv:1706.03412> and CAD-OSE see M. Smirnov (2018) <https://github.com/smirmik/CAD>. The first three algorithms belong to prediction-based techniques and the last three belong to window-based techniques. In addition, the SD-EWMA and PEWMA algorithms are algorithms designed to work in stationary environments, while the other four are algorithms designed to work in non-stationary environments.

Depends R (>= 3.4.0)

SystemRequirements Python (>= 3.0.1); bencode-python3 (1.0.2)

License AGPL (>= 3)

URL https://github.com/alaineiturria/otsad

BugReports https://github.com/alaineiturria/otsad/issues

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests testthat, stream, knitr, rmarkdown

Imports stats, ggplot2, plotly, sigmoid, reticulate

VignetteBuilder knitr

NeedsCompilation no

Author Alaiñe Iturria [aut, cre],
Jacinto Carrasco [aut],
Francisco Herrera [aut],
Santiago Charramendieta [aut],
Karmele Intxausti [aut]

Maintainer Alaiñe Iturria <aiturria@ikerlan.es>

Repository CRAN

Date/Publication 2019-04-01 16:40:03 UTC

R topics documented:

ambient_temperature_system_failure . . . 3
art_daily_flatmiddle . . . 4
art_daily_jumpsdown . . . 4
art_daily_jumpsup . . . 5
art_daily_nojump . . . 5
art_increase_spike_density . . . 6
art_load_balancer_spikes . . . 6
ContextualAnomalyDetector . . . 7
CpKnnCad . . . 8
CpPewma . . . 10
CpSdEwma . . . 11
CpTsSdEwma . . . 13
cpu_utilization_asg_misconfiguration . . . 14
ec2_cpu_utilization_24ae8d . . . 15
ec2_cpu_utilization_53ea38 . . . 15
ec2_cpu_utilization_5f5533 . . . 16
ec2_cpu_utilization_77c1ca . . . 16
ec2_cpu_utilization_825cc2 . . . 17
ec2_cpu_utilization_ac20cd . . . 17
ec2_cpu_utilization_fe7f93 . . . 18
ec2_disk_write_bytes_1ef3de . . . 18
ec2_disk_write_bytes_c0d644 . . . 19
ec2_network_in_257a54 . . . 19
ec2_network_in_5abac7 . . . 20
ec2_request_latency_system_failure . . . 20
elb_request_count_8c0756 . . . 21
exchange_2_cpc_results . . . 21
exchange_2_cpm_results . . . 22
exchange_3_cpc_results . . . 22
exchange_3_cpm_results . . . 23
exchange_4_cpc_results . . . 23
exchange_4_cpm_results . . . 24
GetDetectorScore . . . 24
GetLabels . . . 26
GetNullAndPerfectScores . . . 27
GetNumTrainingValues . . . 28
GetWindowLength . . . 29
GetWindowsLimits . . . 30
grok_asg_anomaly . . . 31
iio_us_east1_i_a2eb1cd9_NetworkIn . . . 32
IpKnnCad . . . 32
IpPewma . . . 35

IpSdEwma 38
IpTsSdEwma 41
machine_temperature_system_failure 44
NormalizeScore 44
nyc_taxi 46
occupancy_6005 46
occupancy_t4013 47
OcpPewma 47
OcpSdEwma 49
OcpTsSdEwma 50
OipPewma 52
OipSdEwma 54
OipTsSdEwma 57
PlotDetections 60
rds_cpu_utilization_cc0c53 62
rds_cpu_utilization_e47b3b 62
ReduceAnomalies 63
rogue_agent_key_hold 64
rogue_agent_key_updown 64
speed_6005 65
speed_7578 65
speed_t4013 66
TravelTime_387 66
TravelTime_451 67
Twitter_volume_AAPL 67
Twitter_volume_AMZN 68
Twitter_volume_CRM 68
Twitter_volume_CVS 69
Twitter_volume_FB 69
Twitter_volume_GOOG 70
Twitter_volume_IBM 70
Twitter_volume_KO 71
Twitter_volume_PFE 71
Twitter_volume_UPS 72

Index **73**

`ambient_temperature_system_failure`
ambient_temperature_system_failure.

Description

The ambient temperature in an office setting.

Usage

```
ambient_temperature_system_failure
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`art_daily_flatmiddle` *art_daily_flatmiddle.*

Description

Artificially-generated data with varying types of anomalies

Usage

```
art_daily_flatmiddle
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`art_daily_jumpsdown` *art_daily_jumpsdown*

Description

Artificially-generated data with varying types of anomalies

Usage

```
art_daily_jumpsdown
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

art_daily_jumpsup *art_daily_jumpsup*

Description

Artificially-generated data with varying types of anomalies

Usage

```
art_daily_jumpsup
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

art_daily_nojump *art_daily_nojump*

Description

Artificially-generated data with varying types of anomalies

Usage

```
art_daily_nojump
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`art_increase_spike_density`
art_increase_spike_density

Description

Artificially-generated data with varying types of anomalies

Usage

```
art_increase_spike_density
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`art_load_balancer_spikes`
art_load_balancer_spikes.

Description

Artificially-generated data with varying types of anomalies

Usage

```
art_load_balancer_spikes
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`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(length(data) * 0.03), 1)), max.left.semicontexts = 7,
  max.active.neurons = 15, num.norm.value.bits = 3,
  base.threshold = 0.75, min.value = min(data, na.rm = T),
  max.value = max(data, na.rm = T), python.object = NULL, lib = 0)
```

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.
`min.value`	Minimum expected value.
`max.value`	Maximum expected value.
`python.object`	Python object for incremental processing.
`lib`	0 to run the original python script, 1 to get the same results on all operating systems.

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. Requires hashlib (included in python installation) and bencode-python3 (which can be installed using pip) python libraries.

Value

List

result	Data frame with anomaly.score and is.anomaly comparing the anomaly score with base.threshold.
python.object	ContextualAnomalyDetector Python object used in online anomaly detection

References

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

Examples

```
## Generate data
set.seed(100)
n <- 200
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

## Calculate anomalies

  result <- ContextualAnomalyDetector(data = df$value, rest.period = 10, base.threshold = 0.9)

   ## Plot results
   res <- cbind(df, result$result)
   PlotDetections(res, title = "CAD_OSE ANOMALY DETECTOR")
```

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 <- 350
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, result)
PlotDetections(res, title = "KNN-CAD ANOMALY DETECTOR")
```

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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 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)
PlotDetections(res, title = "PEWMA ANOMALY DETECTOR")
```

`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(data, n.train, threshold = 0.01, 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.
threshold	Error smoothing constant.
l	Control limit multiplier.

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.

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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

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

```
## Plot results
PlotDetections(res, title = "KNN-CAD ANOMALY DETECTOR")
```

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

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.

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 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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

## Calculate anomalies
result <- CpTsSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3,
  m = 20
)
res <- cbind(df, result)

## Plot results
PlotDetections(res, title = "TSSD_EWMA ANOMALY DETECTOR")
```

---

cpu_utilization_asg_misconfiguration

cpu_utilization_asg_misconfiguration.

Description

From Amazon Web Services (AWS) monitoring CPU usage – i.e. average CPU usage across a given cluster. When usage is high, AWS spins up a new machine, and uses fewer machines when usage is low.

Usage

```
cpu_utilization_asg_misconfiguration
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_cpu_utilization_24ae8d`
ec2_cpu_utilization_24ae8d.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_cpu_utilization_24ae8d
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_cpu_utilization_53ea38`
ec2_cpu_utilization_53ea38.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_cpu_utilization_53ea38
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_cpu_utilization_5f5533`
ec2_cpu_utilization_5f5533.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_cpu_utilization_5f5533
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_cpu_utilization_77c1ca`
ec2_cpu_utilization_77c1ca.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_cpu_utilization_77c1ca
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_cpu_utilization_825cc2`
ec2_cpu_utilization_825cc2.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_cpu_utilization_825cc2
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_cpu_utilization_ac20cd`
ec2_cpu_utilization_ac20cd.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_cpu_utilization_ac20cd
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_cpu_utilization_fe7f93`
ec2_cpu_utilization_fe7f93.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_cpu_utilization_fe7f93
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_disk_write_bytes_1ef3de`
ec2_disk_write_bytes_1ef3de.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_disk_write_bytes_1ef3de
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_disk_write_bytes_c0d644` *ec2_disk_write_bytes_c0d644.*

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_disk_write_bytes_c0d644
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_network_in_257a54` *ec2_network_in_257a54.*

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_network_in_257a54
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_network_in_5abac7` *ec2_network_in_5abac7.*

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
ec2_network_in_5abac7
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`ec2_request_latency_system_failure`
ec2_request_latency_system_failure.

Description

CPU usage data from a server in Amazon's East Coast datacenter. The dataset ends with complete system failure resulting from a documented failure of AWS API servers.

Usage

```
ec2_request_latency_system_failure
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`elb_request_count_8c0756`
elb_request_count_8c0756.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
elb_request_count_8c0756

elb_request_count_8c0756
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`exchange_2_cpc_results`
exchange_2_cpc_results.

Description

Online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost per thousand impressions (CPM). One of the files is normal, without anomalies.

Usage

```
exchange_2_cpc_results
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`exchange_2_cpm_results` *exchange_2_cpm_results.*

Description

Online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost per thousand impressions (CPM). One of the files is normal, without anomalies.

Usage

```
exchange_2_cpm_results
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`exchange_3_cpc_results` *exchange_3_cpc_results.*

Description

Online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost per thousand impressions (CPM). One of the files is normal, without anomalies.

Usage

```
exchange_3_cpc_results
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`exchange_3_cpm_results` *exchange_3_cpm_results.*

Description

Online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost per thousand impressions (CPM). One of the files is normal, without anomalies.

Usage

```
exchange_3_cpm_results
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`exchange_4_cpc_results` *exchange_4_cpc_results.*

Description

Online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost per thousand impressions (CPM). One of the files is normal, without anomalies.

Usage

```
exchange_4_cpc_results
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

exchange_4_cpm_results *exchange_4_cpm_results.*

Description

Online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost per thousand impressions (CPM). One of the files is normal, without anomalies.

Usage

```
exchange_4_cpm_results
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

GetDetectorScore *Get detector score*

Description

`GetDetectorScore` Calculates the start and end positions of each window that are focused on the real anomalies. This windows can be used to know if the detected anomaly is a true positive or not.

Usage

```
GetDetectorScore(data, print = FALSE, title = "")
```

Arguments

`data`	All dataset with training and test datasets and with at least `timestamp`, `value`, `is.anomaly` and `is.real.anomaly` columns.
`print`	If TRUE shows a graph with results.
`title`	Title of the graph.

Details

`data` must be a data.frame with `timestamp`, `value`, `is.anomaly` and `is.real.anomaly` columns. `timestamp` column can be numeric, of type POSIXct, or a character type date convertible to POSIXct.

This function calculates the scores based on three different profiles. Each label tp, fp, tn, fn is associated with a weight to give a more realistic score. For the standard profile weights are tp = 1, tn = 1, fp, = 0.11, and fn = 1. For the reward_low_FP_rate profile weights are tp = 1, tn = 1, fp, = 0.22, and fn = 1. For the reward_low_FN_rate profile weights are tp = 1, tn = 1, fp, = 0.11, and fn = 2.

Value

List conformed by the following items:

`data`	Same data set with additional columns such as `label`, `start.limit`, `end.limit`, `standard.score` and etc.
`standard`	Total score obtained by the detector using the weights of the standard profile.
`low_FP_rate`	Total score obtained by the detector using the weights of the reward_low_FP_rate profile.
`low_FN_rate`	Total score obtained by the detector using the weights of the reward_low_FN_rate profile.
`tp`	Number of true positives
`tn`	Number of true negatives
`fp`	Number of false positives
`fn`	Number of false negatives

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA'15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

# Add is.real.anomaly column
df$is.real.anomaly <- 0
df[c(25,80,150), "is.real.anomaly"] <- 1

## Calculate anomalies
result <- CpSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3
)
res <- cbind(df, result)

# Get detector score
GetDetectorScore(res, print = FALSE, title = "")
```

GetLabels *Get Lables*

Description

GetLabels Calculates the start and end positions of each window that are focused on the real anomalies. This windows can be used to know if the detected anomaly is a true positive or not.

Usage

```
GetLabels(data)
```

Arguments

data	All dataset with training and test datasets with at least timestamp, value, is.anomaly, is.real.anomaly, start.limit and end.limit columns.

Details

data must be a data.frame with timestamp, value, is.anomaly and is.real.anomaly columns. timestamp column can be numeric, of type POSIXct, or a character type date convertible to POSIXct. see GetWindowsLimits to know more about how to get start.limit and end.limit columns.

Value

Same data set with two additional columns label and first.tp. first.tp indicates for each window Which is the position of first true positive. label indicates for each detection if it is a TP, FP, TN or FN.

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA'15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

# Add is.real.anomaly column
df$is.real.anomaly <- 0
```

```
df[c(25,80,150), "is.real.anomaly"] <- 1

## Calculate anomalies
result <- CpSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3
)
res <- cbind(df, result)

# Get Window Limits
data <- GetWindowsLimits(res)
data[data$is.real.anomaly == 1,]

# Get labels
data <- GetLabels(data)
data[data$is.real.anomaly == 1 | data$is.anomaly == 1,]

# Plot results
PlotDetections(res, print.real.anomaly = TRUE, print.time.window = TRUE)
```

`GetNullAndPerfectScores` *Get Null And Perfect Scores*

Description

`GetNullAndPerfectScores` Calculates the score of Perfect and Null detectors scores. Perfect detector is one that outputs all true positives and no false positives. And Null detector is one that outputs no anomaly detections.

Usage

```
GetNullAndPerfectScores(data)
```

Arguments

`data`	All dataset with training and test datasets and with at least `timestamp`, `value` and `is.real.anomaly` columns.

Details

This function calculates the scores based on three different profiles. Each tp, fp, tn, fn label is associated with a weight to give a more realistic score. For the standard profile weights are tp = 1, tn = 1, fp, = 0.11, and fn = 1. For the reward_low_FP_rate profile weights are tp = 1, tn = 1, fp, = 0.22, and fn = 1. For the reward_low_FN_rate profile weights are tp = 1, tn = 1, fp, = 0.11, and fn = 2.

Value

data.frame with null and perfect detectors scores for each profile.

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA'15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

# Add is.real.anomaly column
df$is.real.anomaly <- 0
df[c(25,80,150), "is.real.anomaly"] <- 1

# Get null and perfect scores
GetNullAndPerfectScores(df)
```

GetNumTrainingValues *Get Number of Training Values*

Description

`GetNumTrainingValues` Calculates the number of values to be used as a training set.

Usage

```
GetNumTrainingValues(n.row, prob.percent = 0.15)
```

Arguments

`n.row`	Number of rows of the all dataset with training and test values.
`prob.percent`	Percentage of training values

Details

the number of values to be used as a training set is calculated as a minimum between 15% of the number of rows in the dataset and 15% of 5000.

Value

Number of training values.

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA'15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

# Get number of instances to train phase
GetNumTrainingValues(nrow(df))
```

GetWindowLength *Get Window Length*

Description

`GetWindowLength` Calculates the size of the window. This window focuses on the real anomaly and it can be used to know if the detected anomaly is a true positive or not.

Usage

```
GetWindowLength(data.length, num.real.anomaly, window.length.perc = 0.1)
```

Arguments

`data.length`	Dataset length.
`num.real.anomaly`	Number of real anomalies contained in the data set.
`window.length.perc`	Window length in percentage of the total data

Details

`nrow.data` and `num.real.anomaly` must be numeric. Window length is calculated by default as 10% of the length of the data set divided by the number of real anomalies contained in it.

Value

Window length as numeric.

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA 15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

# Add is.real.anomaly column
df$is.real.anomaly <- 0
df[c(25,80,150), "is.real.anomaly"] <- 1

# Get window length
GetWindowLength(data.length = nrow(df), num.real.anomaly = 3)
```

GetWindowsLimits *Get windows limits*

Description

`GetWindowsLimits` Calculates the start and end positions of each window that are focused on the real anomalies. This windows can be used to know if the detected anomaly is a true positive or not.

Usage

```
GetWindowsLimits(data, windowLength = NULL)
```

Arguments

`data`	All dataset with training and test datasets and with at least `timestamp`, `value` and `is.real.anomaly` columns.
`windowLength`	Window length. See `GetWindowLength`.

Details

data must be a data.frame with timestamp, value, is.anomaly and is.real.anomaly columns. timestamp column can be numeric, of type POSIXct, or a character type date convertible to POSIXct. windowLength must be numeric value.

Value

Same data set with two additional columns start.limit and end.limit where for each is.real.anomaly equal to 1 is indicated the position in the data set where each window starts and ends. If two anomalies fall within the same window, the start and end positions are only indicated on the first of them.

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA'15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

# Add is.real.anomaly column
df$is.real.anomaly <- 0
df[c(25,80,150), "is.real.anomaly"] <- 1

# Get Window Limits
data <- GetWindowsLimits(df)
data[data$is.real.anomaly == 1,]
```

---

grok_asg_anomaly *grok_asg_anomaly.*

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
grok_asg_anomaly
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`iio_us_east1_i_a2eb1cd9_NetworkIn` *iio_us_east1_i_a2eb1cd9_NetworkIn.*

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
iio_us_east1_i_a2eb1cd9_NetworkIn
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`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, is.anomaly = result$is.anomaly)
PlotDetections(res, print.time.window = FALSE, title = "KNN-CAD ANOMALY DETECTOR")

## 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
PlotDetections(res, title = "KNN-CAD ANOMALY DETECTOR")
```

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 <- 350
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$value,
  alpha0 = 0.8,
  beta = 0.1,
  n.train = 5,
  l = 3,
  last.res = NULL
)
res <- cbind(df, result$result)

## Plot results
PlotDetections(res, title = "PEWMA ANOMALY DETECTOR")

## 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$value,
    n.train = 5,
    alpha0 = 0.8,
    beta = 0.1,
    l = 3,
    last.res = last.res$last.res
  )
  # prepare the result
  if(!is.null(last.res$result)){
    res <- rbind(res, cbind(newRow, last.res$result))
  }
}

## Plot results
PlotDetections(res, title = "PEWMA ANOMALY DETECTOR")
```

`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 <- 200
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
```

```
df <- data.frame(timestamp = 1:n, value = x)

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

## Plot results
PlotDetections(res, title = "SD-EWMA ANOMALY DETECTOR")

## 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 <- 350
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$result)){
    res <- rbind(res, cbind(newRow, last.res$result))
  }
```

```
}

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

`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 <- 200
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 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, rbind(result$last.data.checked, result$checked.results,
                           result$to.next.iteration$to.check[, -4]))

## Plot results
PlotDetections(res, print.time.window = FALSE, title = "TSSD-EWMA ANOMALY DETECTOR")

## 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 <- 350
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
PlotDetections(res, title = "TSSD-EWMA ANOMALY DETECTOR")
```

`machine_temperature_system_failure` *machine_temperature_system_failure.*

Description

Temperature sensor data of an internal component of a large, industrial mahcine. The first anomaly is a planned shutdown of the machine. The second anomaly is difficult to detect and directly led to the third anomaly, a catastrophic failure of the machine.

Usage

```
machine_temperature_system_failure
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`NormalizeScore` *Normalize Score using Max and Min normalization*

Description

`ReduceAnomalies` It reduces the number of detected anomalies. This function is designed to reduce the number of false positives keeping only the first detection of all those that are close to each other. This proximity distance is defined by a window

Usage

```
NormalizeScore(real.score, perfect.score, null.score)
```

Arguments

real.score	Detector score. See GetDetectorScore.
perfect.score	Perfect detector score; one that outputs all true positives and no false positives. See GetNullAndPerfectScores.
null.score	Perfect detector score; one that outputs all true positives and no false positives. See GetNullAndPerfectScores.

Value

Normalized score.

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA'15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

# Add is.real.anomaly column
df$is.real.anomaly <- 0
df[c(25,80,150), "is.real.anomaly"] <- 1

## Calculate anomalies
result <- CpSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3
)
res <- cbind(df, result)

# Get null and perfect scores
np.scores <- GetNullAndPerfectScores(df)
np.standard <- np.scores[1,]
np.fp <- np.scores[2,]
np.fn <- np.scores[3,]

# Get detector score
scores <- GetDetectorScore(res, print = FALSE, title = "")

# Normalize standard score
```

```
NormalizeScore(scores$standard, np.standard$perfect.score, np.standard$null.score)

# Normalize low_FP_rate score
NormalizeScore(scores$low_FP_rate, np.fp$perfect.score, np.fp$null.score)

# Normalize low_FN_rate score
NormalizeScore(scores$low_FN_rate, np.fn$perfect.score, np.fn$null.score)
```

`nyc_taxi` *nyc_taxi.*

Description

Number of NYC taxi passengers, where the five anomalies occur during the NYC marathon, Thanksgiving, Christmas, New Years day, and a snow storm. The raw data is from the NYC Taxi and Limousine Commission. The data file included here consists of aggregating the total number of taxi passengers into 30 minute buckets.

Usage

```
nyc_taxi
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`occupancy_6005` *occupancy_6005.*

Description

Real time traffic data from the Twin Cities Metro area in Minnesota, collected by the Minnesota Department of Transportation. Included metrics include occupancy, speed, and travel time from specific sensors.

Usage

```
occupancy_6005
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

occupancy_t4013 *occupancy_t4013.*

Description

Real time traffic data from the Twin Cities Metro area in Minnesota, collected by the Minnesota Department of Transportation. Included metrics include occupancy, speed, and travel time from specific sensors.

Usage

```
occupancy_t4013
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 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)
PlotDetections(res, title = "PEWMA ANOMALY DETECTOR")
```

`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(data, n.train, threshold, 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.
`threshold`	Error smoothing constant.
`l`	Control limit multiplier.

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.

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 <- 200
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

## Calculate anomalies
result <- OcpSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3
)
res <- cbind(df, result)

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

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

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.

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 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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

## Calculate anomalies
result <- OcpTsSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3,
  m = 20
)
res <- cbind(df, result)

## Plot results
PlotDetections(res, title = "TSSD-EWMA ANOMALY DETECTOR")
```

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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

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

## Plot results
PlotDetections(res, title = "PEWMA ANOMALY DETECTOR")

## 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$value,
    n.train = 5,
    alpha0 = 0.8,
    beta = 0.1,
    l = 3,
    last.res = last.res$last.res
  )
  # prepare the result
  if(!is.null(last.res$result)){
    res <- rbind(res, cbind(newRow, last.res$result))
  }
}

## Plot results
PlotDetections(res, print.time.window = FALSE, title = "PEWMA ANOMALY DETECTOR")
```

`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 incre-

mental 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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

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

## Plot results
PlotDetections(res, print.time.window = FALSE, title = "SD-EWMA ANOMALY DETECTOR")

## 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$result)){
    res <- rbind(res, cbind(newRow, last.res$result))
  }
}

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

`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 <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

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

## Plot results
PlotDetections(res, print.time.window = FALSE, title = "TSSD-EWMA ANOMALY DETECTOR")

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

`PlotDetections` *PLOT DETECTIONS*

Description

`PlotDetections` shows in a graph the results obtained after the application of one of the anomaly detectors included in this package.

Usage

```
PlotDetections(data, print.real.anomaly = FALSE,
  print.time.window = FALSE, title = "", xlab = "Time",
  ylab = "Value", return.ggplot = FALSE)
```

Arguments

`data`	data.frame composed of at least one column called timestamp and another column called value. You can also include other columns such as is.anomaly, is.real.anomaly, ucl, lcl, anomaly.score. Any of these columns except is.real.anomaly that are included in the dataset will be shown in the graph automatically.
`print.real.anomaly`	If TRUE adds the real anomalies to the graph.

print.time.window
: If TRUE shows a time band centered on the real anomaly. According to the article shown in the reference, if the detected anomaly remains within it would be considered a true positive.

title
: Title of the graph.

xlab
: X Axis Name.

ylab
: Y Axis Name.

return.ggplot
: If TRUE the function returns a ggplot object.

Details

data must be a data.frame. The timestamp column can be numeric, of type POSIXlt, or a character type date convertible to POSIXlt. The value column must be numeric. is.anomaly, is.real.anomaly, ucl, lcl, anomaly.score are some of the variables returned by the algorithms included in this package and must be numeric or boolean in the case of columns is.anomaly, is.real.anomaly .

Value

plotly object.

References

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in 14th International Conference on Machine Learning and Applications (IEEE ICMLA'15), 2015.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

## Calculate anomalies
result <- CpSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 3
)
res <- cbind(df, result)

## Plot results
PlotDetections(res, title = "KNN-CAD ANOMALY DETECTOR")
```

`rds_cpu_utilization_cc0c53`
rds_cpu_utilization_cc0c53.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
rds_cpu_utilization_cc0c53
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`rds_cpu_utilization_e47b3b`
rds_cpu_utilization_e47b3b.

Description

AWS server metrics as collected by the AmazonCloudwatch service. Example metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes..

Usage

```
rds_cpu_utilization_e47b3b
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

ReduceAnomalies *Reduce Anomalies*

Description

ReduceAnomalies It reduces the number of detected anomalies. This function is designed to reduce the number of false positives keeping only the first detection of all those that are close to each other. This proximity distance is defined by a window

Usage

```
ReduceAnomalies(data, windowLength)
```

Arguments

data	Numerical vector with anomaly labels.
windowLength	Window length.

Value

New Numerical vector with reduced anomaly labels.

Examples

```
## Generate data
set.seed(100)
n <- 180
x <- sample(1:100, n, replace = TRUE)
x[70:90] <- sample(110:115, 21, replace = TRUE)
x[25] <- 200
x[150] <- 170
df <- data.frame(timestamp = 1:n, value = x)

## Calculate anomalies
result <- CpSdEwma(
  data = df$value,
  n.train = 5,
  threshold = 0.01,
  l = 2
)
res <- cbind(df, result)

## Plot results
PlotDetections(res, title = "KNN-CAD ANOMALY DETECTOR")

## Reduce anomalies
res$is.anomaly <- ReduceAnomalies(res$is.anomaly, windowLength = 5)

## Plot results
PlotDetections(res, title = "KNN-CAD ANOMALY DETECTOR")
```

`rogue_agent_key_hold` *rogue_agent_key_hold.*

Description

Timing the key holds for several users of a computer, where the anomalies represent a change in the user.

Usage

```
rogue_agent_key_hold
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

---

`rogue_agent_key_updown`
rogue_agent_key_updown.

Description

Timing the key strokes for several users of a computer, where the anomalies represent a change in the user.

Usage

```
rogue_agent_key_updown
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

speed_6005 *speed_6005.*

Description

Real time traffic data from the Twin Cities Metro area in Minnesota, collected by the Minnesota Department of Transportation. Included metrics include occupancy, speed, and travel time from specific sensors.

Usage

```
speed_6005
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

speed_7578 *speed_7578.*

Description

Real time traffic data from the Twin Cities Metro area in Minnesota, collected by the Minnesota Department of Transportation. Included metrics include occupancy, speed, and travel time from specific sensors.

Usage

```
speed_7578
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

speed_t4013 *speed_t4013.*

Description

Real time traffic data from the Twin Cities Metro area in Minnesota, collected by the Minnesota Department of Transportation. Included metrics include occupancy, speed, and travel time from specific sensors.

Usage

```
speed_t4013
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

TravelTime_387 *TravelTime_387.*

Description

Real time traffic data from the Twin Cities Metro area in Minnesota, collected by the Minnesota Department of Transportation. Included metrics include occupancy, speed, and travel time from specific sensors.

Usage

```
TravelTime_387
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

TravelTime_451 *TravelTime_451.*

Description

Real time traffic data from the Twin Cities Metro area in Minnesota, collected by the Minnesota Department of Transportation. Included metrics include occupancy, speed, and travel time from specific sensors.

Usage

```
TravelTime_451
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

Twitter_volume_AAPL *Twitter_volume_AAPL.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_AAPL
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

Twitter_volume_AMZN *Twitter_volume_AMZN.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_AMZN
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

Twitter_volume_CRM *Twitter_volume_CRM.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_CRM
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`Twitter_volume_CVS` *Twitter_volume_CVS.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_CVS
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`Twitter_volume_FB` *Twitter_volume_FB.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_FB
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

Twitter_volume_GOOG *Twitter_volume_GOOG.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_GOOG
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

Twitter_volume_IBM *Twitter_volume_IBM.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_IBM
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`Twitter_volume_KO` *Twitter_volume_KO.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_KO
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`Twitter_volume_PFE` *Twitter_volume_PFE.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_PFE
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

`Twitter_volume_UPS` *Twitter_volume_UPS.*

Description

A collection of Twitter mentions of large publicly-traded companies such as Google and IBM. The metric value represents the number of mentions for a given ticker symbol every 5 minutes.

Usage

```
Twitter_volume_UPS
```

Format

A data frame with three variables: `timestamp`, `value`, `is.real.anomaly`.

For further details, see https://github.com/numenta/NAB/blob/master/data/README.md

Index

∗Topic **datasets**
ambient_temperature_system_failure, 3
art_daily_flatmiddle, 4
art_daily_jumpsdown, 4
art_daily_jumpsup, 5
art_daily_nojump, 5
art_increase_spike_density, 6
art_load_balancer_spikes, 6
cpu_utilization_asg_misconfiguration, 14
ec2_cpu_utilization_24ae8d, 15
ec2_cpu_utilization_53ea38, 15
ec2_cpu_utilization_5f5533, 16
ec2_cpu_utilization_77c1ca, 16
ec2_cpu_utilization_825cc2, 17
ec2_cpu_utilization_ac20cd, 17
ec2_cpu_utilization_fe7f93, 18
ec2_disk_write_bytes_1ef3de, 18
ec2_disk_write_bytes_c0d644, 19
ec2_network_in_257a54, 19
ec2_network_in_5abac7, 20
ec2_request_latency_system_failure, 20
elb_request_count_8c0756, 21
exchange_2_cpc_results, 21
exchange_2_cpm_results, 22
exchange_3_cpc_results, 22
exchange_3_cpm_results, 23
exchange_4_cpc_results, 23
exchange_4_cpm_results, 24
grok_asg_anomaly, 31
iio_us_east1_i_a2eb1cd9_NetworkIn, 32
machine_temperature_system_failure, 44
nyc_taxi, 46
occupancy_6005, 46
occupancy_t4013, 47
rds_cpu_utilization_cc0c53, 62
rds_cpu_utilization_e47b3b, 62
rogue_agent_key_hold, 64
rogue_agent_key_updown, 64
speed_6005, 65
speed_7578, 65
speed_t4013, 66
TravelTime_387, 66
TravelTime_451, 67
Twitter_volume_AAPL, 67
Twitter_volume_AMZN, 68
Twitter_volume_CRM, 68
Twitter_volume_CVS, 69
Twitter_volume_FB, 69
Twitter_volume_GOOG, 70
Twitter_volume_IBM, 70
Twitter_volume_KO, 71
Twitter_volume_PFE, 71
Twitter_volume_UPS, 72

ambient_temperature_system_failure, 3
art_daily_flatmiddle, 4
art_daily_jumpsdown, 4
art_daily_jumpsup, 5
art_daily_nojump, 5
art_increase_spike_density, 6
art_load_balancer_spikes, 6

ContextualAnomalyDetector, 7
CpKnnCad, 8, *33*
CpPewma, 10, *36*, *47*, *52*
CpSdEwma, 11, *13*, *39*, *41*, *49*, *50*, *55*, *57*
CpTsSdEwma, 13, *50*
cpu_utilization_asg_misconfiguration, 14

ec2_cpu_utilization_24ae8d, 15
ec2_cpu_utilization_53ea38, 15
ec2_cpu_utilization_5f5533, 16
ec2_cpu_utilization_77c1ca, 16

ec2_cpu_utilization_825cc2, 17
ec2_cpu_utilization_ac20cd, 17
ec2_cpu_utilization_fe7f93, 18
ec2_disk_write_bytes_1ef3de, 18
ec2_disk_write_bytes_c0d644, 19
ec2_network_in_257a54, 19
ec2_network_in_5abac7, 20
ec2_request_latency_system_failure, 20
elb_request_count_8c0756, 21
exchange_2_cpc_results, 21
exchange_2_cpm_results, 22
exchange_3_cpc_results, 22
exchange_3_cpm_results, 23
exchange_4_cpc_results, 23
exchange_4_cpm_results, 24

GetDetectorScore, 24, *45*
GetLabels, 26
GetNullAndPerfectScores, 27, *45*
GetNumTrainingValues, 28
GetWindowLength, 29, *30*
GetWindowsLimits, *26*, 30
grok_asg_anomaly, 31

iio_us_east1_i_a2eb1cd9_NetworkIn, 32
IpKnnCad, 32
IpPewma, 35
IpSdEwma, 38
IpTsSdEwma, 41

machine_temperature_system_failure, 44

NormalizeScore, 44
nyc_taxi, 46

occupancy_6005, 46
occupancy_t4013, 47
OcpPewma, *10*, *36*, 47, *52*
OcpSdEwma, *11*, *39*, 49, *55*
OcpTsSdEwma, *13*, 50
OipPewma, *36*, 52
OipSdEwma, *38*, 54
OipTsSdEwma, *41*, 57

PlotDetections, 60

rds_cpu_utilization_cc0c53, 62
rds_cpu_utilization_e47b3b, 62
ReduceAnomalies, 63
rogue_agent_key_hold, 64
rogue_agent_key_updown, 64

speed_6005, 65
speed_7578, 65
speed_t4013, 66

TravelTime_387, 66
TravelTime_451, 67
Twitter_volume_AAPL, 67
Twitter_volume_AMZN, 68
Twitter_volume_CRM, 68
Twitter_volume_CVS, 69
Twitter_volume_FB, 69
Twitter_volume_GOOG, 70
Twitter_volume_IBM, 70
Twitter_volume_KO, 71
Twitter_volume_PFE, 71
Twitter_volume_UPS, 72