Package ‘datadriftR’

January 9, 2025

Title Concept Drift Detection Methods for Stream Data

Version 1.0.0

Description A system designed for detecting concept drift in streaming datasets. It offers a comprehensive suite of statistical methods to detect concept drift, including methods for monitoring changes in data distributions
over time. The package supports several tests, such as Drift Detection Method (DDM), Early Drift Detection Method (EDDM), Hoeffding Drift Detection Methods (HDDM_A, HDDM_W), Kolmogorov-Smirnov test-based Windowing (KSWIN)
and Page Hinkley (PH) tests. The methods implemented in this package are based on established research and have been demonstrated to be effective in real-time data analysis. For more details on the methods,
please check to the following sources. Kobylińska et al. (2023) <doi:10.48550/arXiv.2308.11446>, S. Kullback & R.A. Leibler (1951) <doi:10.1214/aoms/1177729694>, Gama et al. (2004) <doi:10.1007/978-3-540-28645-5_29>, Baena-Garcia et al. (2006) <https://www.researchgate.net/publication/245999704_Early_Drift_Detection_Method>, Frías-Blanco et al. (2014) <https://ieeexplore.ieee.org/document/6871418>, Raab et al. (2020) <doi:10.1016/j.neucom.2019.11.111>, Page (1954) <doi:10.1093/biomet/41.1-2.100>, Montiel et al. (2018) <https://jmlr.org/papers/volume19/18-251/18-251.pdf>.

Depends R (>= 3.5.2)

Imports R6, doremi, fda.usc

License GPL (>= 2)

Encoding UTF-8

URL https://github.com/ugurdar/datadriftR

BugReports https://github.com/ugurdar/datadriftR/issues

RoxygenNote 7.3.2

NeedsCompilation no

Author Ugur Dar [aut, cre],
Mustafa Cavus [ctb, ths]

Maintainer Ugur Dar <ugurdarr@gmail.com>

Repository CRAN

Date/Publication 2025-01-09 14:30:09 UTC

Contents

`DDM` *DDM (Drift Detection Method)*

Description

Implements the Drift Detection Method (DDM), used for detecting concept drift in data streams by analyzing the performance of online learners. The method monitors changes in the error rate of a learner, signaling potential concept drift.

Details

DDM is designed to be simple yet effective for detecting concept drift by monitoring the error rate of any online classifier. The method is particularly sensitive to increases in the error rate, which is typically a strong indicator of concept drift.

Public fields

`min_instances` Minimum number of instances required before drift detection begins.

`warning_level` Multiplier for the standard deviation to set the warning threshold.

`out_control_level` Multiplier for the standard deviation to set the out-of-control threshold.

`sample_count` Counter for the number of samples processed.

`miss_prob` Current estimated probability of misclassification.

`miss_std` Current estimated standard deviation of misclassification probability.

`miss_prob_sd_min` Minimum recorded value of misclassification probability plus its standard deviation.

`miss_prob_min` Minimum recorded misclassification probability.

`miss_sd_min` Minimum recorded standard deviation.

`estimation` Current estimation of misclassification probability.

`change_detected` Boolean indicating if a drift has been detected.

`warning_detected` Boolean indicating if a warning level has been reached.

`delay` Delay since the last relevant sample.

Methods

Public methods:

- `DDM$new()`
- `DDM$reset()`
- `DDM$add_element()`
- `DDM$detected_change()`
- `DDM$clone()`

Method `new()`**:** Initializes the DDM detector with specific parameters.

Usage:

```
DDM$new(min_num_instances = 30, warning_level = 2, out_control_level = 3)
```

Arguments:

`min_num_instances` Minimum number of samples required before starting drift detection.

`warning_level` Threshold multiplier for setting a warning level.

`out_control_level` Threshold multiplier for setting the out-of-control level.

Method `reset()`**:** Resets the internal state of the DDM detector.

Usage:

```
DDM$reset()
```

Method `add_element()`**:** Adds a new prediction error value to the model, updates the calculation of the misclassification probability and its standard deviation, and checks for warnings or drifts based on updated statistics.

Usage:

```
DDM$add_element(prediction)
```

Arguments:

`prediction` The new data point (prediction error) to be added to the model.

Method `detected_change()`**:** Returns a boolean indicating whether a drift has been detected based on the monitored statistics.

Usage:

```
DDM$detected_change()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DDM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

João Gama, Pedro Medas, Gladys Castillo, Pedro Pereira Rodrigues: Learning with Drift Detection. SBIA 2004: 286-295

Implementation: https://github.com/scikit-multiflow/scikit-multiflow/blob/a7e316d1cc79988a6df40da35312e00f6c4eabb2/s

Examples

```
set.seed(123)  # Setting a seed for reproducibility
data_part1 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.7, 0.3))

# Introduce a change in data distribution
data_part2 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.3, 0.7))

# Combine the two parts
data_stream <- c(data_part1, data_part2)
ddm <- DDM$new()
# Iterate through the data stream
for (i in seq_along(data_stream)) {
  ddm$add_element(data_stream[i])
  if (ddm$change_detected) {
    message(paste("Drift detected!", i))
  } else if (ddm$warning_detected) {
    # message(paste("Warning detected at position:", i))
  }
}
```

EDDM *EDDM (Early Drift Detection Method)*

Description

This class implements the Early Drift Detection Method (EDDM), designed to detect concept drifts in online learning scenarios by monitoring the distances between consecutive errors. EDDM is particularly useful for detecting gradual drifts earlier than abrupt changes.

Details

EDDM is a statistical process control method that is more sensitive to changes that happen more slowly and can provide early warnings of deterioration before the error rate increases significantly.

Public fields

`eddm_warning` Warning threshold setting.

`eddm_outcontrol` Out-of-control threshold setting.

`m_num_errors` Current number of errors encountered.

`m_min_num_errors` Minimum number of errors to initialize drift detection.

`m_n` Total instances processed.

`m_d` Distance to the last error from the current instance.

`m_lastd` Distance to the previous error from the last error.

`m_mean` Mean of the distances between errors.

`m_std_temp` Temporary standard deviation accumulator for the distances.

`m_m2s_max` Maximum mean plus two standard deviations observed.

`delay` Delay count since the last detected change.

`estimation` Current estimated mean distance between errors.

`warning_detected` Boolean indicating if a warning has been detected.

`change_detected` Boolean indicating if a change has been detected.

Methods

Public methods:

- `EDDM$new()`
- `EDDM$reset()`
- `EDDM$add_element()`
- `EDDM$clone()`

Method `new()`**:** Initializes the EDDM detector with specific parameters.

Usage:

```
EDDM$new(min_num_instances = 30, eddm_warning = 0.95, eddm_outcontrol = 0.9)
```

Arguments:

`min_num_instances` Minimum number of errors before drift detection starts.

`eddm_warning` Threshold for warning level.

`eddm_outcontrol` Threshold for out-of-control level.

Method `reset()`**:** Resets the internal state of the EDDM detector.

Usage:

```
EDDM$reset()
```

Method `add_element()`**:** Adds a new observation and updates the drift detection status.

Usage:

```
EDDM$add_element(prediction)
```

Arguments:

`prediction` Numeric value representing a new error (usually 0 or 1).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
EDDM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Early Drift Detection Method. Manuel Baena-Garcia, Jose Del Campo-Avila, Raúl Fidalgo, Albert Bifet, Ricard Gavalda, Rafael Morales-Bueno. In Fourth International Workshop on Knowledge Discovery from Data Streams, 2006.

Implementation: https://github.com/scikit-multiflow/scikit-multiflow/blob/a7e316d1cc79988a6df40da35312e00f6c4eabb2/s

Examples

```
set.seed(123)  # Setting a seed for reproducibility
data_part1 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.7, 0.3))

# Introduce a change in data distribution
data_part2 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.3, 0.7))

# Combine the two parts
data_stream <- c(data_part1, data_part2)
eddm <- EDDM$new()
for (i in 1:length(data_stream)) {
  eddm$add_element(data_stream[i])
  if (eddm$change_detected) {
    message(paste("Drift detected!",i))
  } else if (eddm$warning_detected) {
    message(paste("Warning detected!",i))
  }
}
```

HDDM_A *HDDM_A: Drift Detection Method based on Adaptive Windows*

Description

This class implements the HDDM_A drift detection method that uses adaptive windows to detect changes in the mean of a data stream. It is designed to monitor online streams of data and can detect increases or decreases in the process mean in a non-parametric and online manner.

Details

HDDM_A adapts to changes in the data stream by adjusting its internal windows to track the minimum and maximum values of the process mean. It triggers alerts when a significant drift from these benchmarks is detected.

Public fields

`drift_confidence` Confidence level for detecting a drift.

`warning_confidence` Confidence level for warning detection.

`two_side_option` Boolean flag for one-sided or two-sided mean monitoring.

`total_n` Total number of samples seen.

`total_c` Total cumulative sum of the samples.

`n_max` Maximum window end for sample count.

`c_max` Maximum window end for cumulative sum.

`n_min` Minimum window start for sample count.

`c_min` Minimum window start for cumulative sum.

`n_estimation` Number of samples since the last detected change.

`c_estimation` Cumulative sum since the last detected change.

`change_detected` Boolean indicating if a change was detected.

`warning_detected` Boolean indicating if a warning has been detected.

`estimation` Current estimated mean of the stream.

`delay` Current delay since the last update.

Methods

Public methods:

- `HDDM_A$new()`
- `HDDM_A$add_element()`
- `HDDM_A$mean_incr()`
- `HDDM_A$mean_decr()`
- `HDDM_A$reset()`
- `HDDM_A$update_estimations()`
- `HDDM_A$clone()`

Method `new()`**:** Initializes the HDDM_A detector with specific settings.

Usage:

```
HDDM_A$new(
  drift_confidence = 0.001,
  warning_confidence = 0.005,
  two_side_option = TRUE
)
```

Arguments:

`drift_confidence` Confidence level for drift detection.

`warning_confidence` Confidence level for issuing warnings.

`two_side_option` Whether to monitor both increases and decreases.

Method `add_element()`**:** Adds an element to the data stream and updates the detection status.

Usage:

```
HDDM_A$add_element(prediction)
```

Arguments:

`prediction` Numeric, the new data value to add.

Method `mean_incr()`**:** Calculates if there is an increase in the mean.

Usage:

```
HDDM_A$mean_incr(c_min, n_min, total_c, total_n, confidence)
```

Arguments:

`c_min` Minimum cumulative sum.

`n_min` Minimum count of samples.

`total_c` Total cumulative sum.

`total_n` Total number of samples.

`confidence` Confidence threshold for detection.

Method `mean_decr()`**:** Calculates if there is a decrease in the mean.

Usage:

`HDDM_A$mean_decr(c_max, n_max, total_c, total_n)`

Arguments:

`c_max` Maximum cumulative sum.

`n_max` Maximum count of samples.

`total_c` Total cumulative sum.

`total_n` Total number of samples.

Method `reset()`**:** Resets all internal counters and accumulators to their initial state.

Usage:

`HDDM_A$reset()`

Method `update_estimations()`**:** Updates estimations of the mean after detecting changes.

Usage:

`HDDM_A$update_estimations()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`HDDM_A$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

References

Frías-Blanco I, del Campo-Ávila J, Ramos-Jimenez G, et al. Online and non-parametric drift detection methods based on Hoeffding's bounds. IEEE Transactions on Knowledge and Data Engineering, 2014, 27(3): 810-823.

Albert Bifet, Geoff Holmes, Richard Kirkby, Bernhard Pfahringer. MOA: Massive Online Analysis; Journal of Machine Learning Research 11: 1601-1604, 2010.

Implementation: https://github.com/scikit-multiflow/scikit-multiflow/blob/a7e316d1cc79988a6df40da35312e00f6c4eabb2/s

Examples

```
set.seed(123)  # Setting a seed for reproducibility
data_part1 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.7, 0.3))

# Introduce a change in data distribution
data_part2 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.3, 0.7))

# Combine the two parts
data_stream <- c(data_part1, data_part2)
```

```
# Initialize the hddm_a object
hddm_a_instance <- HDDM_A$new()

# Iterate through the data stream
for(i in seq_along(data_stream)) {
  hddm_a_instance$add_element(data_stream[i])
  if(hddm_a_instance$warning_detected) {
    message(paste("Warning detected at index:", i))
  }
  if(hddm_a_instance$change_detected) {
    message(paste("Concept drift detected at index:", i))
  }
}
```

`HDDM_W` *KSWIN (Kolmogorov-Smirnov WINdowing) for Change Detection*

Description

Implements the Kolmogorov-Smirnov test for detecting distribution changes within a window of streaming data. KSWIN is a non-parametric method for change detection that compares two samples to determine if they come from the same distribution.

Details

KSWIN is effective for detecting changes in the underlying distribution of data streams. It is particularly useful in scenarios where data properties may evolve over time, allowing for early detection of changes that might affect subsequent data processing.

Public fields

`drift_confidence` Confidence level for detecting a drift (default: 0.001).

`warning_confidence` Confidence level for warning detection (default: 0.005).

`lambda_option` Decay rate for the EWMA statistic, smaller values give less weight to recent data (default: 0.050).

`two_side_option` Boolean flag for one-sided or two-sided error monitoring (default: TRUE).

`total` Container for the EWMA estimator and its bounded conditional sum.

`sample1_decr_monitor` First sample monitor for detecting decrements.

`sample1_incr_monitor` First sample monitor for detecting increments.

`sample2_decr_monitor` Second sample monitor for detecting decrements.

`sample2_incr_monitor` Second sample monitor for detecting increments.

`incr_cutpoint` Cutpoint for deciding increments.

`decr_cutpoint` Cutpoint for deciding decrements.

`width` Current width of the window.

`delay` Delay count since last reset.

`change_detected` Boolean indicating if a change was detected.

`warning_detected` Boolean indicating if currently in a warning zone.

`estimation` The current estimation of the stream's mean.

Methods

Public methods:

- `HDDM_W$new()`
- `HDDM_W$add_element()`
- `HDDM_W$SampleInfo()`
- `HDDM_W$reset()`
- `HDDM_W$detect_mean_increment()`
- `HDDM_W$monitor_mean_incr()`
- `HDDM_W$monitor_mean_decr()`
- `HDDM_W$update_incr_statistics()`
- `HDDM_W$update_decr_statistics()`
- `HDDM_W$clone()`

Method `new()`**:** Initializes the HDDM_W detector with specific parameters.

Usage:

```
HDDM_W$new(
  drift_confidence = 0.001,
  warning_confidence = 0.005,
  lambda_option = 0.05,
  two_side_option = TRUE
)
```

Arguments:

`drift_confidence` Confidence level for drift detection.

`warning_confidence` Confidence level for issuing warnings.

`lambda_option` Decay rate for the EWMA statistic.

`two_side_option` Whether to monitor both increases and decreases.

Method `add_element()`**:** Adds a new element to the data stream and updates the detection status.

Usage:

```
HDDM_W$add_element(prediction)
```

Arguments:

`prediction` The new data value to add.

Method `SampleInfo()`**:** Provides current information about the monitoring samples, typically used for debugging or monitoring.

Usage:

```
HDDM_W$SampleInfo()
```

Method `reset()`**:** Resets the internal state to initial conditions.

Usage:

```
HDDM_W$reset()
```

Method `detect_mean_increment()`**:** Detects an increment in the mean between two samples based on the provided confidence level.

Usage:

```
HDDM_W$detect_mean_increment(sample1, sample2, confidence)
```

Arguments:

`sample1` First sample information, containing EWMA estimator and bounded conditional sum.

`sample2` Second sample information, containing EWMA estimator and bounded conditional sum.

`confidence` The confidence level used for calculating the bound.

Returns: Boolean indicating if an increment in mean was detected.

Method `monitor_mean_incr()`**:** Monitors the data stream for an increase in the mean based on the set confidence level.

Usage:

```
HDDM_W$monitor_mean_incr(confidence)
```

Arguments:

`confidence` The confidence level used to detect changes in the mean.

Returns: Boolean indicating if an increase in the mean was detected.

Method `monitor_mean_decr()`**:** Monitors the data stream for a decrease in the mean based on the set confidence level.

Usage:

```
HDDM_W$monitor_mean_decr(confidence)
```

Arguments:

`confidence` The confidence level used to detect changes in the mean.

Returns: Boolean indicating if a decrease in the mean was detected.

Method `update_incr_statistics()`**:** Updates increment statistics for drift monitoring based on new values and confidence. This method adjusts the cutpoint for increments and updates the monitoring samples.

Usage:

```
HDDM_W$update_incr_statistics(value, confidence)
```

Arguments:

`value` The new value to update statistics.

`confidence` The confidence level for the update.

Method `update_decr_statistics()`**:** Updates decrement statistics for drift monitoring based on new values and confidence. This method adjusts the cutpoint for decrements and updates the monitoring samples.

Usage:

```
HDDM_W$update_decr_statistics(value, confidence)
```

Arguments:

`value` The new value to update statistics.

`confidence` The confidence level for the update.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
HDDM_W$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Frías-Blanco I, del Campo-Ávila J, Ramos-Jimenez G, et al. Online and non-parametric drift detection methods based on Hoeffding's bounds. IEEE Transactions on Knowledge and Data Engineering, 2014, 27(3): 810-823.

Albert Bifet, Geoff Holmes, Richard Kirkby, Bernhard Pfahringer. MOA: Massive Online Analysis; Journal of Machine Learning Research 11: 1601-1604, 2010. Implementation: https://github.com/scikit-multiflow/scikit-multiflow/blob/a7e316d1cc79988a6df40da35312e00f6c4eabb2/src/skmultiflow/drift_detection/hddm_w.py

Examples

```
set.seed(123)  # Setting a seed for reproducibility
data_part1 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.7, 0.3))

# Introduce a change in data distribution
data_part2 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.3, 0.7))

# Combine the two parts
data_stream <- c(data_part1, data_part2)

# Initialize the HDDM_W object
hddm_w_instance <- HDDM_W$new()

# Iterate through the data stream
for(i in seq_along(data_stream)) {
  hddm_w_instance$add_element(data_stream[i])
  if(hddm_w_instance$warning_detected) {
    message(paste("Warning detected at index:", i))
  }
  if(hddm_w_instance$change_detected) {
    message(paste("Concept drift detected at index:", i))
  }
}
```

`KLDivergence` *Kullback-Leibler Divergence (KLD) for Change Detection*

Description

Implements the Kullback-Leibler Divergence (KLD) calculation between two probability distributions using histograms. The class can detect drift by comparing the divergence to a predefined threshold.

Details

The Kullback-Leibler Divergence (KLD) is a measure of how one probability distribution diverges from a second, expected probability distribution. This class uses histograms to approximate the distributions and calculates the KLD to detect changes over time. If the divergence exceeds a predefined threshold, it signals a detected drift.

Public fields

`epsilon` Value to add to small probabilities to avoid log(0) issues.

`base` The base of the logarithm used in KLD calculation.

`bins` Number of bins used for the histogram.

`drift_level` The threshold for detecting drift.

`drift_detected` Boolean indicating if drift has been detected.

`p` Initial distribution.

`kl_result` The result of the KLD calculation.

Methods

Public methods:

- `KLDivergence$new()`
- `KLDivergence$reset()`
- `KLDivergence$set_initial_distribution()`
- `KLDivergence$add_distribution()`
- `KLDivergence$calculate_kld()`
- `KLDivergence$get_kl_result()`
- `KLDivergence$is_drift_detected()`
- `KLDivergence$clone()`

Method `new()`**:** Initializes the KLDivergence class.

Usage:

```
KLDivergence$new(epsilon = 1e-10, base = exp(1), bins = 10, drift_level = 0.2)
```

Arguments:

`epsilon` Value to add to small probabilities to avoid log(0) issues.

`base` The base of the logarithm used in KLD calculation.

`bins` Number of bins used for the histogram.

`drift_level` The threshold for detecting drift.

Method `reset()`**:** Resets the internal state of the detector.

Usage:

```
KLDivergence$reset()
```

Method `set_initial_distribution()`**:** Sets the initial distribution.

Usage:

```
KLDivergence$set_initial_distribution(initial_p)
```

Arguments:

`initial_p` The initial distribution.

Method `add_distribution()`**:** Adds a new distribution and calculates the KLD.

Usage:

```
KLDivergence$add_distribution(q)
```

Arguments:

`q` The new distribution.

Method `calculate_kld()`**:** Calculates the KLD between two distributions.

Usage:

```
KLDivergence$calculate_kld(p, q)
```

Arguments:

`p` The initial distribution.

`q` The new distribution.

Returns: The KLD value.

Method `get_kl_result()`**:** Returns the current KLD result.

Usage:

```
KLDivergence$get_kl_result()
```

Returns: The current KLD value.

Method `is_drift_detected()`**:** Checks if drift has been detected.

Usage:

```
KLDivergence$is_drift_detected()
```

Returns: TRUE if drift is detected, otherwise FALSE.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
KLDivergence$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Kullback, S., and Leibler, R.A. (1951). On Information and Sufficiency. Annals of Mathematical Statistics, 22(1), 79-86.

Examples

```
set.seed(123)  # Setting a seed for reproducibility
initial_data <- c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)
kld <- KLDivergence$new(bins = 10, drift_level = 0.2)
kld$set_initial_distribution(initial_data)

new_data <- c(0.2, 0.2, 0.3, 0.4, 0.4, 0.5, 0.6, 0.7, 0.7, 0.8)
kld$add_distribution(new_data)

kl_result <- kld$get_kl_result()
message(paste("KL Divergence:", kl_result))

if (kld$is_drift_detected()) {
  message("Drift detected.")
}
```

`KSWIN` *KSWIN (Kolmogorov-Smirnov WINdowing) for Change Detection*

Description

Implements the Kolmogorov-Smirnov test for detecting distribution changes within a window of streaming data. KSWIN is a non-parametric method for change detection that compares two samples to determine if they come from the same distribution.

Details

KSWIN is effective for detecting changes in the underlying distribution of data streams. It is particularly useful in scenarios where data properties may evolve over time, allowing for early detection of changes that might affect subsequent data processing.

Public fields

`alpha` Significance level for the KS test.

`window_size` Total size of the data window used for testing.

`stat_size` Number of data points sampled from the window for the KS test.

`window` Current data window used for change detection.

`change_detected` Boolean flag indicating whether a change has been detected.

`p_value` P-value of the most recent KS test.

Methods

Public methods:

- `KSWIN$new()`
- `KSWIN$reset()`
- `KSWIN$add_element()`
- `KSWIN$detected_change()`
- `KSWIN$clone()`

Method `new()`**:** Initializes the KSWIN detector with specific settings.

Usage:

```
KSWIN$new(alpha = 0.005, window_size = 100, stat_size = 30, data = NULL)
```

Arguments:

`alpha` The significance level for the KS test.

`window_size` The size of the data window for change detection.

`stat_size` The number of samples in the statistical test window.

`data` Initial data to populate the window, if provided.

Method `reset()`**:** Resets the internal state of the detector to its initial conditions.

Usage:

```
KSWIN$reset()
```

Method `add_element()`**:** Adds a new element to the data window and updates the detection status based on the KS test.

Usage:

```
KSWIN$add_element(x)
```

Arguments:

`x` The new data value to add to the window.

Method `detected_change()`**:** Checks if a change has been detected based on the most recent KS test.

Usage:

```
KSWIN$detected_change()
```

Returns: Boolean indicating whether a change was detected.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
KSWIN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Christoph Raab, Moritz Heusinger, Frank-Michael Schleif, Reactive Soft Prototype Computing for Concept Drift Streams, Neurocomputing, 2020.

Implementation: https://github.com/scikit-multiflow/scikit-multiflow/blob/a7e316d1cc79988a6df40da35312e00f6c4eabb2/s

Examples

```
set.seed(123)  # Setting a seed for reproducibility
data_part1 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.7, 0.3))

# Introduce a change in data distribution
data_part2 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.3, 0.7))

# Combine the two parts
data_stream <- c(data_part1, data_part2)
```

`PageHinkley` *Page-Hinkley Test for Change Detection*

Description

Implements the Page-Hinkley test, a sequential analysis technique used to detect changes in the average value of a continuous signal or process. It is effective in detecting small but persistent changes over time, making it suitable for real-time monitoring applications.

Details

The Page-Hinkley test is a type of cumulative sum (CUSUM) test that accumulates differences between data points and a reference value (running mean). It triggers a change detection signal when the cumulative sum exceeds a predefined threshold. This test is especially useful for early detection of subtle shifts in the behavior of the monitored process.

Public fields

`min_instances` Minimum number of instances required to start detection.

`delta` Minimal change considered significant for detection.

`threshold` Decision threshold for signaling a change.

`alpha` Forgetting factor for the cumulative sum calculation.

`x_mean` Running mean of the observed values.

`sample_count` Counter for the number of samples seen.

`sum` Cumulative sum used in the change detection.

`change_detected` Boolean indicating if a drift has been detected.

Methods

Public methods:

- `PageHinkley$new()`
- `PageHinkley$reset()`
- `PageHinkley$add_element()`
- `PageHinkley$detected_change()`

- PageHinkley$clone()

Method `new()`: Initializes the Page-Hinkley test with specific parameters.

Usage:

```
PageHinkley$new(
  min_instances = 30,
  delta = 0.005,
  threshold = 50,
  alpha = 1 - 1e-04
)
```

Arguments:

`min_instances` Minimum number of samples before detection starts.

`delta` Change magnitude to trigger detection.

`threshold` Cumulative sum threshold for change detection.

`alpha` Weight for older data in cumulative sum.

Method `reset()`: Resets all the internal states of the detector to initial values.

Usage:

```
PageHinkley$reset()
```

Method `add_element()`: Adds a new element to the data stream and updates the detection status based on the Page-Hinkley test.

Usage:

```
PageHinkley$add_element(x)
```

Arguments:

`x` New data value to add and evaluate.

Method `detected_change()`: Checks if a change has been detected based on the last update.

Usage:

```
PageHinkley$detected_change()
```

Returns: Boolean indicating whether a change was detected.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PageHinkley$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

E. S. Page. 1954. Continuous Inspection Schemes. Biometrika 41, 1/2 (1954), 100–115.

Montiel, Jacob, et al. "Scikit-Multiflow: A Multi-output Streaming Framework." Journal of Machine Learning Research, 2018. This framework provides tools for multi-output and stream data mining and was an inspiration for some of the implementations in this class.

Implementation: https://github.com/scikit-multiflow/scikit-multiflow/blob/a7e316d1cc79988a6df40da35312e00f6c4eabb2/s

Examples

```
set.seed(123)  # Setting a seed for reproducibility
data_part1 <- sample(c(0, 1), size = 100, replace = TRUE, prob = c(0.7, 0.3))

# Introduce a change in data distribution
data_part2 <- sample(c(0, 5), size = 100, replace = TRUE, prob = c(0.3, 0.7))

# Combine the two parts
data_stream <- c(data_part1, data_part2)
ph <- PageHinkley$new()
for (i in seq_along(data_stream)) {
  ph$add_element(data_stream[i])
  if (ph$detected_change()) {
   cat(sprintf("Change has been detected in data: %s - at index: %d\n", data_stream[i], i))
  }
}
```

ProfileDifference *Profile Difference Calculation for Change Detection*

Description

Implements the calculation of profile differences using various methods such as PDI, L2, and L2 derivative. The class provides methods for setting profiles and calculating the differences.

Details

The class supports multiple methods for calculating profile differences, including the Profile Disparity Index (PDI) using gold or simple derivative methods, and L2 norm and L2 derivative calculations. It allows for customization of various parameters such as embedding dimensions, derivative orders, and thresholds.

Public fields

`method` The method used for profile difference calculation.

`deriv` The method used for derivative calculation.

`gold_spline` Boolean indicating if cubic spline should be used in gold method.

`gold_embedding` Embedding dimension for gold method.

`nderiv` Order of the derivative for simple method.

`gold_spline_threshold` Threshold for cubic spline in gold method.

`epsilon` Small value to avoid numerical issues.

`profile1` The first profile.

`profile2` The second profile.

Methods

Public methods:

- ProfileDifference$new()
- ProfileDifference$reset()
- ProfileDifference$set_profiles()
- ProfileDifference$calculate_difference()
- ProfileDifference$clone()

Method `new()`**:** Initializes the ProfileDifference class.

Usage:

```
ProfileDifference$new(
  method = "pdi",
  deriv = "gold",
  gold_spline = TRUE,
  gold_embedding = 4,
  nderiv = 4,
  gold_spline_threshold = 0.01,
  epsilon = NULL
)
```

Arguments:

`method` The method used for profile difference calculation.

`deriv` The method used for derivative calculation.

`gold_spline` Boolean indicating if cubic spline should be used in gold method.

`gold_embedding` Embedding dimension for gold method.

`nderiv` Order of the derivative for simple method.

`gold_spline_threshold` Threshold for cubic spline in gold method.

`epsilon` Small value to avoid numerical issues.

Method `reset()`**:** Resets the internal state of the detector.

Usage:

```
ProfileDifference$reset()
```

Method `set_profiles()`**:** Sets the profiles for comparison.

Usage:

```
ProfileDifference$set_profiles(profile1, profile2)
```

Arguments:

`profile1` The first profile.

`profile2` The second profile.

Method `calculate_difference()`**:** Calculates the difference between the profiles.

Usage:

```
ProfileDifference$calculate_difference()
```

Returns: A list containing the method details and the calculated distance.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ProfileDifference$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Kobylińska, K., Krzyziński, M., Machowicz, R., Adamek, M., & Biecek, P. (2023). Exploration of the Rashomon Set Assists Trustworthy Explanations for Medical Data. arXiv e-prints, arXiv-2308.

Examples

```
set.seed(123)  # Setting a seed for reproducibility
profile1 <- list(x = 1:100, y = sin(1:100))
profile2 <- list(x = 1:100, y = sin(1:100) + rnorm(100, 0, 0.1))
pd <- ProfileDifference$new(method = "pdi", deriv = "gold")
pd$set_profiles(profile1, profile2)
result <- pd$calculate_difference()
message(result)
```

Index