

Package: heimdall (via r-universe)

October 16, 2026

Title Drift Adaptable Models

Version 1.4.0

Description In streaming data analysis, it is crucial to detect significant shifts in the data distribution or the accuracy of predictive models over time, a phenomenon known as concept drift. The package aims to identify when concept drift occurs and provide methodologies for adapting models in non-stationary environments. It offers a range of state-of-the-art techniques for detecting concept drift and maintaining model performance. Additionally, the package provides tools for adapting models in response to these changes, ensuring continuous and accurate predictions in dynamic contexts. Methods for concept drift detection are described in Tavares (2022) [<doi:10.1007/s12530-021-09415-z>](https://doi.org/10.1007/s12530-021-09415-z).

License MIT + file LICENSE

URL <https://cefet-rj-dal.github.io/heimdall/>,
<https://github.com/cefet-rj-dal/heimdall>

BugReports <https://github.com/cefet-rj-dal/heimdall/issues>

Encoding UTF-8

Depends R (>= 3.5.0)

Roxxygen list(markdown = TRUE)

Imports stats, utils, daltoolbox, pROC

Suggests daltoolboxdp, reticulate, testthat (>= 3.0.0), knitr,
rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libicu-dev

Repository <https://cefet-rj-dal.r-universe.dev>

Date/Publication 2026-10-16 11:12:54 UTC

RemoteUrl <https://github.com/cefet-rj-dal/heimdall>

RemoteRef HEAD

RemoteSha aad63c38824f3f78ee7be0e43a1f8208b234c39a

Contents

dfr_adwin	3
dfr_aedd	4
dfr_cusum	6
dfr_ddm	7
dfr_ecdd	9
dfr_eddm	11
dfr_hddm	12
dfr_inactive	14
dfr_kldist	15
dfr_kswin	17
dfr_lbdd	19
dfr_mcdd	21
dfr_multi_criteria	23
dfr_page_hinkley	24
dfr_passive	26
dist_based	27
drifter	27
dummy	28
error_based	28
fit.drifter	29
metric	30
mt_accuracy	30
mt_fscore	31
mt_precision	31
mt_recall	32
mt_rocauc	32
mv_dist_based	33
nrm_base	33
nrm_memory	34
reset_state	35
st_drift_examples	35
stealthy	36
update_state	38

Index	39
--------------	-----------

dfr_adwin

ADWIN method

Description

ADWIN (Adaptive Windowing) is a sequential change detector that maintains a variable-length window and tests whether the means of two subwindows differ significantly. In this package, the implementation is primarily used for **virtual concept drift** when it monitors a numeric feature stream, although the same mechanism can also detect **real concept drift** if applied to an error or loss stream. The theoretical basis follows Bifet and Gavalda (2007) [doi:10.1137/1.9781611972771.42](https://doi.org/10.1137/1.9781611972771.42).

Usage

```
dfr_adwin(target_feat = NULL, delta = 2e-05)
```

Arguments

target_feat	Feature to be monitored.
delta	The significance parameter for the ADWIN algorithm.

Details

The implementation is written in pure R and follows the exponential-histogram version of ADWIN distributed with scikit-multiflow. The window is compressed into buckets of exponentially growing size (at most five buckets per size), and the cut points are checked every 32 observations, once the window holds more than 10 observations. All the state lives in the object itself, so a `dfr_adwin` object can be serialized with `base::saveRDS()`.

Value

`dfr_adwin` object

References

Bifet, A., and Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, 443-448. [doi:10.1137/1.9781611972771.42](https://doi.org/10.1137/1.9781611972771.42)

Examples

```
library(daltoolbox)
library(heimdall)

data(st_drift_examples)
data <- st_drift_examples$uv_virtual_drift
data$event <- NULL
```

```

model <- dfr_adwin(target_feat = "serie")

detection <- NULL
output <- list(obj = model, drift = FALSE)
for (i in seq_along(data$serie)) {
  output <- update_state(output$obj, data$serie[i])
  if (output$drift) {
    type <- "drift"
    output$obj <- reset_state(output$obj)
  } else {
    type <- ""
  }
  detection <- rbind(detection, data.frame(idx = i, event = output$drift, type = type))
}

print(detection[detection$type == "drift", ])

```

dfr_aedd

Autoencoder-Based Drift Detection method

Description

AEDD is an unsupervised multivariate detector that compares reconstruction errors produced by an autoencoder on reference and recent windows. Because it monitors changes in the input distribution rather than classifier performance, this implementation is primarily aimed at **virtual concept drift**. The method follows Kaminskyi, Li, and Muller (2022) [doi:10.1109/ICDMW58026.2022.00109](https://doi.org/10.1109/ICDMW58026.2022.00109).

Usage

```

dfr_aedd(
  encoding_size,
  ae_class = NULL,
  ae_params = list(batch_size = 32, num_epochs = 1000, learning_rate = 0.001,
    error_function = "mae"),
  window_size = 100,
  window_type = "sliding",
  monitoring_step = floor(window_size/2),
  criteria = "mann_whitney",
  alpha = 0.01,
  threshold = 0.01,
  target_feat = NULL
)

```

Arguments

encoding_size	Encoding Size
ae_class	Autoencoder constructor. When NULL (the default), <code>daltoolboxdp::autoenc_ed</code> is used, which requires the suggested package <code>daltoolboxdp</code> and its Python backend.

ae_params	Named list of arguments forwarded to ae_class, plus error_function, which selects how the reconstruction error is summarised. One of "mae" (default) or "rmse".
window_size	Number of most recent rows kept by the detector. The oldest half is the reference window and the newest half the recent window.
window_type	Window policy. "sliding" (default) keeps at most window_size rows, so both halves move with the stream. "anchored" keeps every row, which pins the reference half to the beginning of the stream and makes memory grow with the stream length.
monitoring_step	Number of rows between two consecutive checks. The default checks twice per window.
criteria	The method used to check for a drift. One of "mann_whitney" (default), "kolmogorov_smirnov", "levene", "psi", "parametric_threshold", "nonparametric_threshold" or "diff_threshold".
alpha	Significance level of the hypothesis-test criteria ("mann_whitney", "kolmogorov_smirnov" and "levene").
threshold	Drift threshold of the "psi" and "diff_threshold" criteria, compared directly against the divergence or the median error difference. It is a raw threshold, not a probability.
target_feat	Names of the columns to be monitored. NULL (the default) monitors every numeric column.

Details

The autoencoder expects inputs in the $[0, 1]$ range. Values outside that range are clipped to it, and the detector warns the first time this happens, so the data should be normalized beforehand (for example with `daltoolbox::minmax()`). `ae_class` must be a constructor accepting the arguments `input_size` and `encoding_size` plus the entries of `ae_params`, whose objects implement `fit()` and `transform()`.

The autoencoder is trained on the reference window the first time the window is full, and again after every drift or whenever new columns appear.

The "levene" criterion tests every reconstructed feature and combines the per-feature p-values with a Bonferroni correction, instead of looking at the first feature only.

Value

dfr_aedd object

References

Kaminskyi, D., Li, B., and Muller, E. (2022). Reconstruction-based unsupervised drift detection over multivariate streaming data. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. doi:[10.1109/ICDMW58026.2022.00109](https://doi.org/10.1109/ICDMW58026.2022.00109)

Examples

```
# Training an autoencoder is too slow for an inline example. A rendered
# walkthrough is available at
# https://github.com/cefet-rj-dal/heimdall/blob/main/examples/1_detection/dfr_aedd.md
```

dfr_cusum

Cumulative Sum for Concept Drift Detection (CUSUM) method

Description

CUSUM is a sequential analysis procedure that accumulates deviations in a monitored signal and raises an alarm when the cumulative evidence exceeds a threshold. In this package, the detector is implemented as an error-based monitor, so it is primarily intended for **real concept drift** affecting predictive performance. The concept-drift adaptation follows the sequential change-detection literature discussed by Muthukrishnan, Berg, and Wu (2007) [doi:10.1109/ICDMW.2007.89](https://doi.org/10.1109/ICDMW.2007.89).

Usage

```
dfr_cusum(lambda = 50, delta = 0.005, min_instances = 30)
```

Arguments

lambda	Detection threshold: a drift is reported when the cumulative sum exceeds it. Lower values react faster at the cost of more false alarms.
delta	Tolerance subtracted at every step, i.e. the increase of the error rate that is still considered acceptable.
min_instances	Minimum number of observations before a drift can be reported.

Details

The detector follows the CusumDM detector of MOA: it accumulates $\max(0, \text{sum} + x - \text{mean} - \text{delta})$, where mean is the running error rate, so it reacts to an increase of the error rate whatever its level. After a drift is reported the statistics restart from scratch. Missing values are treated as correct predictions (0).

Up to version 1.3.0, `dfr_cusum()` accumulated +1 for every error and -1 for every correct prediction, so it could only detect error rates above 50%.

Value

dfr_cusum object

References

Muthukrishnan, S., Berg, E., and Wu, Y. (2007). Sequential change detection on data streams. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. [doi:10.1109/ICDMW.2007.89](https://doi.org/10.1109/ICDMW.2007.89)

Examples

```
# Load Heimdall and the synthetic benchmark stream.
library(heimdall)

# Fix the seed for reproducibility.
seed <- 1
set.seed(seed)

# Load the stream and derive a simple binary error-like signal.
data(st_drift_examples)
data <- st_drift_examples$uv_virtual_drift
data$prediction <- st_drift_examples$uv_virtual_drift$serie > 4

# Plot the binary monitored stream used by CUSUM.
plot(x=seq_len(nrow(data)), y=data$prediction)

# Instantiate the CUSUM detector.
model <- dfr_cusum()

# Update the detector sequentially over the binary stream.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(data))) {
  output <- update_state(output$obj, data$prediction[i])
  if (output$drift) {
    type <- 'drift'
    output$obj <- reset_state(output$obj)
  } else {
    type <- ''
  }
  detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
}

# Print the drift alarms produced by the detector.
detection[detection$type == 'drift',]

# Map the drift alarms back onto the original numeric series.
plot(x=seq_len(nrow(data)), y=data$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}
```

dfr_ddm

Adapted Drift Detection Method (DDM) method

Description

DDM monitors the online error rate of a predictive model under the PAC-learning assumption that, in a stationary environment, the error should decrease or remain stable as more samples are observed. Because it operates on the classifier error stream, it is primarily a detector of **real concept drift**. The method follows Gama et al. (2004) [doi:10.1007/978-3-540-28645-5_29](https://doi.org/10.1007/978-3-540-28645-5_29).

Usage

```
dfr_ddm(min_instances = 30, warning_level = 2, out_control_level = 3)
```

Arguments

`min_instances` The minimum number of instances before detecting change

`warning_level` Necessary level for warning zone (2 standard deviation)

`out_control_level`
 Necessary level for a positive drift detection

Details

The detector only reacts to an increase of the error rate (a degradation of the model); improvements are not reported. Values must be 1 (error) or 0 (correct prediction), and missing values are treated as correct predictions (0).

Value

dfr_ddm object

References

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. P. (2004). Learning with drift detection. In *Advances in Artificial Intelligence - SBIA 2004*, 286-295. doi:[10.1007/978-3-540-28645-5_29](https://doi.org/10.1007/978-3-540-28645-5_29)

Examples

```
# Load Heimdall and the built-in synthetic data stream.
library(heimdall)

# Fix the seed to make the example reproducible.
seed <- 1
set.seed(seed)

# Load the univariate stream and derive the binary monitored signal.
data(st_drift_examples)
data <- st_drift_examples$uv_virtual_drift
data$prediction <- st_drift_examples$uv_virtual_drift$serie > 4

# Plot the binary stream monitored by DDM.
plot(x=seq_len(nrow(data)), y=data$prediction)

# Instantiate the DDM detector.
model <- dfr_ddm()

# Process the stream sequentially and store the drift alarms.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(data))) {
  output <- update_state(output$obj, data$prediction[i])
}
```

```

    if (output$drift){
      type <- 'drift'
      output$obj <- reset_state(output$obj)
    } else {
      type <- ''
    }
    detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
  }

# Print the positions where DDM detected drift.
detection[detection$type == 'drift',]

# Show the detected drifts over the original numeric signal.
plot(x=seq_len(nrow(data)), y=data$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}

```

dfr_ecdd

Adapted EWMA for Concept Drift Detection (ECDD) method

Description

ECDD applies an exponentially weighted moving average (EWMA) control chart to the online classification error stream. Since it monitors predictive errors directly, it is primarily designed to detect **real concept drift**. The method follows Ross et al. (2012), who adapted EWMA charts for concept-drift detection in streaming classifiers [doi:10.1016/j.patrec.2011.08.019](https://doi.org/10.1016/j.patrec.2011.08.019).

Usage

```
dfr_ecdd(lambda = 0.2, min_run_instances = 30, average_run_length = 400)
```

Arguments

lambda	EWMA smoothing parameter
min_run_instances	The minimum number of instances before detecting change
average_run_length	Desired Average Run Length (ARL), i.e. the expected number of observations between two false alarms. The published control-limit polynomials exist for 100, 400 and 1000; other values use the next one up, and values above 1000 are rejected. The default changed from 100 to 400 in the development version, which reduced false alarms about sevenfold.

Details

Missing values are treated as correct predictions (0).

Value

dfr_ecdd object

References

Ross, G. J., Adams, N. M., Tasoulis, D. K., and Hand, D. J. (2012). Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2), 191-198. doi:[10.1016/j.patrec.2011.08.019](https://doi.org/10.1016/j.patrec.2011.08.019)

Examples

```
# Load Heimdall and the synthetic example stream.
library(heimdall)

# Fix the seed for reproducibility.
seed <- 1
set.seed(seed)

# Load the stream and derive the binary monitored signal.
data(st_drift_examples)
data <- st_drift_examples$uv_virtual_drift
data$prediction <- st_drift_examples$uv_virtual_drift$serie > 4

# Plot the binary stream that ECDD will monitor.
plot(x=seq_len(nrow(data)), y=data$prediction)

# Instantiate the ECDD detector.
model <- dfr_ecdd()

# Update the detector sequentially and store the detected alarms.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(data))) {
  output <- update_state(output$obj, data$prediction[i])
  if (output$drift) {
    type <- 'drift'
    output$obj <- reset_state(output$obj)
  } else {
    type <- ''
  }
  detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
}

# Print the drift points signaled by ECDD.
detection[detection$type == 'drift',]

# Overlay the alarms on the original numeric signal.
plot(x=seq_len(nrow(data)), y=data$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}
```

dfr_eddm

*Adapted Early Drift Detection Method (EDDM) method***Description**

EDDM extends DDM by monitoring the distance between classification errors instead of only the error rate, which makes it more sensitive to gradual degradation. Because it operates on the model error stream, it is primarily intended for **real concept drift**. The method follows Baena-Garcia et al. (2006), who proposed EDDM for improved detection of gradual drift.

Usage

```
dfr_eddm(
  min_instances = 30,
  min_num_errors = 30,
  warning_level = 0.95,
  out_control_level = 0.9
)
```

Arguments

`min_instances` The minimum number of instances before detecting change
`min_num_errors` The minimum number of errors before detecting change
`warning_level` Necessary level for warning zone
`out_control_level` Necessary level for a positive drift detection

Details

EDDM monitors the distance between consecutive errors, which makes it sensitive to gradual drifts but also prone to false alarms when the error rate is high. `warning_level` must not be lower than `out_control_level`. Values must be 1 (error) or 0 (correct prediction), and missing values are treated as correct predictions (0).

Value

dfr_eddm object

References

Baena-Garcia, M., del Campo-Avila, J., Fidalgo, R., Bifet, A., Gavalda, R., and Morales-Bueno, R. (2006). Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*.

Examples

```
# Load Heimdall and the built-in synthetic stream.
library(heimdall)

# Fix the seed so the results are reproducible.
seed <- 1
set.seed(seed)

# Load the stream and derive the binary monitored signal.
data(st_drift_examples)
data <- st_drift_examples$uv_virtual_drift
data$prediction <- st_drift_examples$uv_virtual_drift$serie > 4

# Visualize the binary stream that EDDM will monitor.
plot(x=seq_len(nrow(data)), y=data$prediction)

# Instantiate the EDDM detector.
model <- dfr_eddm()

# Process the stream sequentially and store each detected alarm.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(data))) {
  output <- update_state(output$obj, data$prediction[i])
  if (output$drift) {
    type <- 'drift'
    output$obj <- reset_state(output$obj)
  } else {
    type <- ''
  }
  detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
}

# Print the drift alarms returned by EDDM.
detection[detection$type == 'drift',]

# Show those alarms on top of the original numeric series.
plot(x=seq_len(nrow(data)), y=data$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}
```

dfr_hddm

Adapted Hoeffding Drift Detection Method (HDDM) method

Description

HDDM_A is a sequential detector based on Hoeffding's inequality that tests whether the mean of the monitored error stream has increased beyond statistically expected fluctuations. Because this implementation is error-based, it is primarily targeted at **real concept drift**. The theoretical basis follows Frias-Blanco et al. (2015) [doi:10.1109/TKDE.2014.2345382](https://doi.org/10.1109/TKDE.2014.2345382).

Usage

```
dfr_hddm(  
  drift_confidence = 0.001,  
  warning_confidence = 0.005,  
  two_side_option = TRUE  
)
```

Arguments

drift_confidence	Confidence to the drift
warning_confidence	Confidence to the warning
two_side_option	Option to monitor error increments and decrements (two-sided) or only increments (one-sided)

Details

The detector reports increases of the error rate. With `two_side_option = TRUE`, a significant decrease restarts the statistics but is not reported as a drift. Values must be 1 (error) or 0 (correct prediction), and missing values are treated as correct predictions (0).

Value

dfr_hddm object

References

Frias-Blanco, I., del Campo-Avila, J., Ramos-Jimenez, G., Morales-Bueno, R., Ortiz-Diaz, A., and Caballero-Mota, Y. (2015). Online and nonparametric drift detection methods based on Hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3), 810-823. doi:[10.1109/TKDE.2014.2345382](https://doi.org/10.1109/TKDE.2014.2345382)

Examples

```
# Load Heimdall and the synthetic stream example.  
library(heimdall)  
  
# Fix the seed for reproducibility.  
seed <- 1  
set.seed(seed)  
  
# Load the stream and derive the binary monitored signal.  
data(st_drift_examples)  
data <- st_drift_examples$uv_virtual_drift  
data$prediction <- st_drift_examples$uv_virtual_drift$serie > 4  
  
# Plot the binary stream that HDDM_A will monitor.  
plot(x=seq_len(nrow(data)), y=data$prediction)
```

```

# Instantiate the HDDM_A detector.
model <- dfr_hddm()

# Process the stream sequentially and record the detector alarms.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(data))){
  output <- update_state(output$obj, data$prediction[i])
  if (output$drift){
    type <- 'drift'
    output$obj <- reset_state(output$obj)
  } else {
    type <- ''
  }
  detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
}

# Print the detected drift points.
detection[detection$type == 'drift',]

# Overlay the alarms on the original numeric stream.
plot(x=seq_len(nrow(data)), y=data$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}

```

dfr_inactive

Inactive dummy detector

Description

Implements a dummy detector that never reports a drift. Useful as a lower baseline when evaluating adaptation strategies.

Usage

```
dfr_inactive()
```

Value

Drifter object

Examples

```

model <- dfr_inactive()
output <- update_state(model, 1)
output$drift

```

dfr_kldist

*KL Distance method***Description**

This detector compares a reference and a recent window through the symmetrised Kullback-Leibler divergence of their binned empirical distributions, $\sum((p - q) * \log(p / q))$, which is the quantity also known as the Jeffreys divergence or, in the credit-risk literature, the Population Stability Index. It is multivariate: every monitored column contributes to the score. Because it monitors the distribution of the observed features rather than predictive error, it is primarily used for **virtual concept drift**. The statistical foundation is the divergence introduced by Kullback and Leibler (1951).

Usage

```
dfr_kldist(
  target_feat = NULL,
  window_size = 100,
  threshold = 0.12,
  window_type = "sliding",
  breaks = 20,
  monitoring_step = 1,
  data = NULL,
  p_th = NULL
)
```

Arguments

target_feat	Names of the columns to be monitored. NULL (the default) monitors every numeric column.
window_size	Size of the window
threshold	Drift threshold applied to the divergence. The default (0.12) was calibrated by simulation for the default window: on stationary normal data the divergence stayed below 0.1, while a shift of two standard deviations exceeded 0.13 in 90% of the runs. Shifts smaller than about one standard deviation are not separable from noise with 50-row halves; use a larger window to detect them.
window_type	Window policy. "sliding" (default) keeps at most window_size rows, so both halves move with the stream. "anchored" keeps every row, which pins the reference half to the beginning of the stream and makes memory grow with the stream length.
breaks	Number of bins used to discretise each feature before the divergence is computed.
monitoring_step	Number of rows between two consecutive evaluations. The default (1) evaluates at every row; larger values reduce the computational cost on long streams.
data	Already collected data to avoid cold start.
p_th	Deprecated name of threshold.

Details

Bins are built over the reference and recent halves together, so the two histograms are always comparable. Empty bins are smoothed before the logarithm is taken, and the score is averaged over bins and over features.

The detector consumes one row of a data frame at a time (a single value is also accepted). Rows whose monitored values are all missing are skipped instead of being imputed.

Value

dfr_kldist object

References

Kullback, S., and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1), 79-86. doi:[10.1214/aoms/1177729694](https://doi.org/10.1214/aoms/1177729694)

Examples

```
# Load Heimdall and the example data stream.
library(heimdall)

# Fix the seed to keep the example reproducible.
seed <- 1
set.seed(seed)

# Load the univariate stream monitored in this walkthrough.
data(st_drift_examples)
serie <- st_drift_examples$uv_virtual_drift

# Plot the monitored numeric signal before detection.
plot(x=seq_len(nrow(serie)), y=serie$serie)

# Instantiate the KL-divergence-based detector.
model <- dfr_kldist(window_size=100, threshold=0.10)

# Update the detector sequentially and store drift alarms.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(serie))) {
  output <- update_state(output$obj, serie[i, 'serie', drop=FALSE])
  if (output$drift) {
    type <- 'drift'
    output$obj <- reset_state(output$obj)
  } else {
    type <- ''
  }
  detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
}

# Print the points where KLDIST detected drift.
detection[detection$type == 'drift',]
```

```
# Overlay the detected drifts on the original numeric stream.
plot(x=seq_len(nrow(serie)), y=serie$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}
```

dfr_kswin

KSWIN method

Description

KSWIN applies a Kolmogorov-Smirnov test between a recent window and a reference sample drawn from older observations. In this package, the method is primarily used for **virtual concept drift**, because it monitors distributional changes in a numeric feature stream. The method follows Raab et al. (2020) [doi:10.1016/j.neucom.2019.11.111](https://doi.org/10.1016/j.neucom.2019.11.111).

Usage

```
dfr_kswin(
  target_feat = NULL,
  window_size = 300,
  stat_size = 100,
  alpha = 1e-04,
  window_type = "sliding",
  monitoring_step = 1,
  exact = NULL,
  data = NULL,
  seed = NULL
)
```

Arguments

target_feat	Feature to be monitored.
window_size	Size of the sliding window. It must be greater than stat_size, and at least 2 * stat_size is recommended so that the reference sample is as large as the recent one.
stat_size	Size of the statistic window
alpha	Probability for the test statistic of the Kolmogorov-Smirnov-Test The alpha parameter is very sensitive, therefore should be set below 0.01.
window_type	Window policy. "sliding" (default) keeps at most window_size observations, so both the reference and the statistic halves move with the stream. "anchored" keeps every observation, which pins the reference sample to the beginning of the stream and makes memory grow with the stream length.
monitoring_step	Number of observations between two consecutive tests. The default (1) tests at every observation; larger values reduce the computational cost on long streams.

exact	Passed to <code>stats::ks.test()</code> . The default (NULL) lets R choose between the exact and the asymptotic test, which is much faster for windows of this size. Up to version 1.3.0 the default was TRUE.
data	Already collected data to avoid cold start.
seed	Seed of the random number stream used to subsample the reference window. When NULL (the default) it is drawn from the global random number generator once, when the detector is created.

Details

The reference window is randomly subsampled to `stat_size` observations. The detector keeps its own random number stream, so it does not change the global random state while it runs; set seed (or call `base::set.seed()` before creating the detector) for reproducible results.

A drift requires both a significant p-value and a KS statistic above $\sqrt{-\log(\alpha) / \text{stat_size}}$, as defined by Raab et al. Missing observations are skipped instead of being imputed.

Value

dfr_kswin object

References

Raab, C., Heusinger, M., and Schleif, F.-M. (2020). Reactive soft prototype computing for concept drift streams. *Neurocomputing*, 416, 340-351. doi:[10.1016/j.neucom.2019.11.111](https://doi.org/10.1016/j.neucom.2019.11.111)

Examples

```
# Load Heimdall and the synthetic univariate stream.
library(heimdall)

# Fix the seed for reproducibility.
seed <- 1
set.seed(seed)

# Load the numeric stream used by the detector.
data(st_drift_examples)
serie <- st_drift_examples$uv_virtual_drift

# Plot the monitored variable before detection.
plot(x=seq_len(nrow(serie)), y=serie$serie)

# Instantiate KSWIN with explicit window parameters.
model <- dfr_kswin(target_feat='serie', window_size=100, stat_size=50)

# Update the detector sequentially and store the alarm positions.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(serie))) {
  output <- update_state(output$obj, serie$serie[i])
  if (output$drift) {
```

```

    type <- 'drift'
    output$obj <- reset_state(output$obj)
  } else {
    type <- ''
  }
  detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
}

# Print the points where KSWIN raised drift alarms.
detection[detection$type == 'drift',]

# Overlay the detected drifts on the original stream.
plot(x=seq_len(nrow(serie)), y=serie$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}

```

dfr_lbdd

Levene Based Drift Detection Method method

Description

LBDD is a window-based detector that compares the variability of a reference and a recent sample using Levene's test. Because it monitors changes in the distribution of an observed feature rather than model performance, it is primarily aimed at **virtual concept drift**. In this package, the detector follows the statistical-testing approach discussed by Giusti et al. (2021) for drift analysis, using Levene's variance test as its core mechanism.

Usage

```

dfr_lbdd(
  target_feat = NULL,
  alpha = 0.001,
  window_size = 300,
  window_type = "sliding",
  monitoring_step = 1,
  data = NULL
)

```

Arguments

target_feat	Feature to be monitored
alpha	Probability threshold for the test statistic
window_size	Size of the window
window_type	Window policy. "sliding" (default) keeps at most window_size observations, so both halves move with the stream. "anchored" keeps every observation, which pins the reference half to the beginning of the stream and makes memory grow with the stream length.

monitoring_step	Number of observations between two consecutive tests. The default (1) tests at every observation; larger values reduce the computational cost on long streams.
data	Already collected data to avoid cold start.

Details

LBDD only reacts to changes of spread (variance); a shift of the mean with the same spread is not detected, so combine it with `dfr_mcdd()` when both kinds of change matter. The test runs at every observation, so alpha should be small to keep false alarms rare.

Levene's test is computed with the group spread centred on the median (the Brown-Forsythe variant).

Missing observations are skipped instead of being imputed. When a drift is reported the window is trimmed to its most recent half; `reset_state()` empties it.

Value

dfr_lbdd object

References

Giusti, L., Carvalho, L., Gomes, A. T., Coutinho, R., Soares, J., and Ogasawara, E. (2021). Analysing flight delay under concept drift. *Evolving Systems*. doi:[10.1007/s12530-021-09415-z](https://doi.org/10.1007/s12530-021-09415-z)

Examples

```
# Load Heimdall and the synthetic univariate stream.
library(heimdall)

# Fix the seed for reproducibility.
seed <- 1
set.seed(seed)

# Load the numeric stream monitored by the detector.
data(st_drift_examples)
serie <- st_drift_examples$uv_virtual_drift

# Plot the series before starting drift detection.
plot(x=seq_len(nrow(serie)), y=serie$serie)

# Instantiate the Levene-based detector.
model <- dfr_lbdd(target_feat='serie', window_size=100)

# Update the detector over the stream and record each drift alarm.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(serie))) {
  output <- update_state(output$obj, serie$serie[i])
  if (output$drift) {
    type <- 'drift'
    output$obj <- reset_state(output$obj)
  }
}
```

```

    } else {
      type <- ''
    }
    detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
  }

  # Print the positions where LBDD detected drift.
  detection[detection$type == 'drift',]

  # Overlay those drift points on the original numeric stream.
  plot(x=seq_len(nrow(serie)), y=serie$serie)
  for (drift_index in detection[detection$type == 'drift', 'idx']) {
    abline(v=drift_index, col='red', lty=2)
  }

```

dfr_mcdd

Mean Comparison Distance method

Description

MCDD is a window-based detector that compares the location of a reference and a recent sample by means of a hypothesis test on their central tendency. Because it monitors the distribution of an observed feature rather than predictive errors, it is primarily intended for **virtual concept drift**. In this package, the detector follows the statistical-testing perspective adopted by Giusti et al. (2021) for drift analysis.

Usage

```

dfr_mcdd(
  target_feat = NULL,
  alpha = 1e-04,
  normality_alpha = 0.05,
  window_size = 300,
  window_type = "sliding",
  monitoring_step = 1,
  data = NULL
)

```

Arguments

target_feat	Feature to be monitored
alpha	Probability threshold for the comparison test
normality_alpha	Significance level of the Shapiro-Wilk normality test used to decide between the parametric and the nonparametric comparison. It is deliberately separate from alpha, since the two thresholds answer different questions.
window_size	Size of the window

window_type	Window policy. "sliding" (default) keeps at most window_size observations, so both halves move with the stream. "anchored" keeps every observation, which pins the reference half to the beginning of the stream and makes memory grow with the stream length.
monitoring_step	Number of observations between two consecutive tests. The default (1) tests at every observation; larger values reduce the computational cost on long streams.
data	Already collected data to avoid cold start.

Details

MCDD only reacts to changes of location (mean or median); a change of spread with the same centre is not detected, so combine it with `dfr_lbdd()` when both kinds of change matter.

Normality of both halves of the window is checked with `stats::shapiro.test()`; when it holds, a t-test is applied, otherwise a Wilcoxon rank-sum test is used. Shapiro-Wilk is only defined for samples of at most 5000 observations, so larger windows always fall back to the nonparametric test.

Missing observations are skipped instead of being imputed.

Value

dfr_mcdd object

References

Giusti, L., Carvalho, L., Gomes, A. T., Coutinho, R., Soares, J., and Ogasawara, E. (2021). Analysing flight delay under concept drift. *Evolving Systems*. doi:[10.1007/s12530-021-09415-z](https://doi.org/10.1007/s12530-021-09415-z)

Examples

```
# Load Heimdall and the synthetic stream.
library(heimdall)

# Fix the seed to keep the walkthrough reproducible.
seed <- 1
set.seed(seed)

# Load the univariate series monitored in this example.
data(st_drift_examples)
serie <- st_drift_examples$uv_virtual_drift

# Plot the monitored signal before running the detector.
plot(x=seq_len(nrow(serie)), y=serie$serie)

# Instantiate the mean-comparison detector.
model <- dfr_mcdd(target_feat='serie', window_size=100)

# Update the detector sequentially and collect the alarm positions.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(serie))) {
```

```

    output <- update_state(output$obj, serie$serie[i])
    if (output$drift){
      type <- 'drift'
      output$obj <- reset_state(output$obj)
    } else {
      type <- ''
    }
    detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
  }

# Print the detected drift points.
detection[detection$type == 'drift',]

# Overlay the detected drifts on the original series.
plot(x=seq_len(nrow(serie)), y=serie$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}

```

dfr_multi_criteria *Multi Criteria Drifter sub-class*

Description

Combines the output of several drift detectors into a single decision.

Usage

```
dfr_multi_criteria(drifter_list, combination = "or", fuzzy_window = 10)
```

Arguments

drifter_list	Named list of drifters to combine. Every element must inherit from either <code>dist_based</code> or <code>mv_dist_based</code> .
combination	How the drifters will be combined. One of 'or' (default), 'and' or 'fuzzy'.
fuzzy_window	Number of observations during which a detection is still taken into account by the 'and' and 'fuzzy' combinations.

Details

- 'or' reports a drift as soon as any detector does;
- 'and' reports a drift when every detector has reported one within the last `fuzzy_window` observations;
- 'fuzzy' gives each detection a weight that decays linearly from 1 to 0 over `fuzzy_window` observations, and reports a drift when the sum of the weights exceeds half the number of detectors.

After a combined drift is reported, the past detections are forgotten, so a single change is reported once. Detectors with a `target_feat` receive that column of each row; the other univariate detectors receive the row mean.

`obj$detections` counts the drifts reported by each detector and `obj$last_drifts` holds their decisions on the last observation.

Value

Drifter object

Examples

```
library(daltoolbox)
library(heimdall)

set.seed(1)
data <- data.frame(serie = c(stats::rnorm(100), stats::rnorm(100, mean = 10)))

model <- dfr_multi_criteria(
  drifter_list = list(
    ph = dfr_page_hinkley(),
    ph_sensitive = dfr_page_hinkley(threshold = 10)
  ),
  combination = "or"
)

model <- fit(model, data)
model$drifted
model$detections
```

dfr_page_hinkley	<i>Adapted Page Hinkley method</i>
------------------	------------------------------------

Description

The Page-Hinkley test is a sequential change-point detector that monitors the cumulative deviation of a signal from its running mean and signals a change when that deviation departs from its running minimum by more than a threshold. In this package, the implementation is primarily used for **virtual concept drift** when it monitors a numeric feature stream, although the same statistic can also be applied to error streams to detect **real concept drift**. The method is based on Page (1954).

Usage

```
dfr_page_hinkley(
  target_feat = NULL,
  min_instances = 30,
  delta = 0.005,
  threshold = 50,
```

```

    alpha = 1 - 1e-04,
    direction = "increase"
  )

```

Arguments

target_feat	Feature to be monitored.
min_instances	The minimum number of instances before detecting change
delta	The delta factor for the Page Hinkley test
threshold	The change detection threshold (lambda)
alpha	The forgetting factor, used to weight the observed value and the mean
direction	Which changes of the mean are detected: "increase" (the default, as in the reference implementation), "decrease" or "both".

Details

Missing observations are skipped instead of being imputed.

Value

dfr_page_hinkley object

References

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2), 100-115. doi:[10.2307/2333009](https://doi.org/10.2307/2333009)

Examples

```

# Load Heimdall and the synthetic stream example.
library(heimdall)

# Fix the seed for reproducibility.
seed <- 1
set.seed(seed)

# Load the univariate numeric stream monitored in this example.
data(st_drift_examples)
serie <- st_drift_examples$uv_virtual_drift

# Plot the monitored signal before detection.
plot(x=seq_len(nrow(serie)), y=serie$serie)

# Instantiate the Page-Hinkley detector.
model <- dfr_page_hinkley(target_feat='serie')

# Update the detector sequentially and record every drift alarm.
detection <- NULL
output <- list(obj=model, drift=FALSE)
for (i in seq_len(nrow(serie))) {

```

```

output <- update_state(output$obj, serie$serie[i])
if (output$drift){
  type <- 'drift'
  output$obj <- reset_state(output$obj)
} else {
  type <- ''
}
detection <- rbind(detection, data.frame(idx=i, event=output$drift, type=type))
}

# Print the detected drift points.
detection[detection$type == 'drift',]

# Overlay those alarms on the original numeric stream.
plot(x=seq_len(nrow(serie)), y=serie$serie)
for (drift_index in detection[detection$type == 'drift', 'idx']) {
  abline(v=drift_index, col='red', lty=2)
}

print(paste('Successfull run at', Sys.time()))

```

dfr_passive

Passive dummy detector

Description

Implements a dummy detector that always reports a drift. Useful as an upper baseline, since it forces the model to be retrained at every batch.

Usage

```
dfr_passive()
```

Value

Drifter object

Examples

```

model <- dfr_passive()
output <- update_state(model, 1)
output$drift

```

dist_based	<i>Distribution Based Drifter sub-class</i>
------------	---

Description

Implements univariate Distribution Based drift detectors. These detectors monitor a single numeric feature; missing observations are skipped instead of being imputed.

Usage

```
dist_based(target_feat)
```

Arguments

target_feat Feature to be monitored.

Value

Drifter object

Examples

```
obj <- dist_based(target_feat = "serie")
obj$target_feat
```

drifter	<i>Drifter</i>
---------	----------------

Description

Ancestor class for drift detection.

Every detector in the package follows the same contract:

- `update_state(obj, value)` returns a list with the updated object (`obj`) and a logical flag (`drift`) telling whether a drift was detected **** in that call****;
- `obj$drifted` is sticky: it stays TRUE after the first detection until `reset_state()` is called;
- `reset_state(obj)` returns the detector to its initial state;
- `obj$drifter_output` holds the per-observation diagnostics produced by the last call to `fit()`, one row per observation; rows are NA when the detector reported nothing for that observation (for example while its window is filling, or for a skipped missing value).

Usage

```
drifter()
```

Value

Drifter object

Examples

```
model <- drifter()
model$drifted
```

dummy

Dummy Drifter sub-class

Description

Ancestor class for the baseline detectors that ignore the monitored signal, `dfr_inactive()` and `dfr_passive()`.

Usage

```
dummy()
```

Value

Drifter object

Examples

```
obj <- dummy()
class(obj)
```

error_based

Error Based Drifter sub-class

Description

Implements Error Based drift detectors. These detectors monitor the residuals of a predictive model, where 1 (or TRUE) means a wrong prediction and 0 (or FALSE) a correct one. Missing values are treated as correct predictions.

Usage

```
error_based()
```

Value

Drifter object

Examples

```
obj <- error_based()
class(obj)
```

`fit.drifter`*Process Batch*

Description

Process Batch

Usage

```
## S3 method for class 'drifter'
fit(obj, data, prediction = NULL, ...)
```

Arguments

<code>obj</code>	Drifter object
<code>data</code>	data batch in data frame format
<code>prediction</code>	prediction batch as vector format. Optional and unused by the default method; it is kept for compatibility with detectors that need the model output.
<code>...</code>	optional arguments

Value

updated Drifter object

Examples

```
library(daltoolbox)
model <- dfr_inactive()
model <- fit(model, data.frame(serie = 1:10))
model$drifted
```

metric	<i>Metric</i>
--------	---------------

Description

Ancestor class for metric calculation

Usage

```
metric()
```

Value

Metric object

Examples

```
obj <- metric()
class(obj)
```

mt_accuracy	<i>Accuracy Calculator</i>
-------------	----------------------------

Description

Class for accuracy calculation. Binary labels (logical, 0/1 or factors with those labels) are compared by value; other labels are compared as text.

Usage

```
mt_accuracy()
```

Value

Metric object

Examples

```
library(daltoolbox)
obj <- mt_accuracy()
evaluate(obj, c(TRUE, FALSE, TRUE), c(TRUE, TRUE, TRUE))
```

`mt_fscore`*FScore Calculator*

Description

Class for FScore calculation

Usage

```
mt_fscore(f = 1)
```

Arguments

`f` The beta parameter of the F-beta score. `f = 1` (the default) gives the usual F1 score, values below 1 weight precision more heavily and values above 1 weight recall more heavily.

Value

Metric object

Examples

```
library(daltoolbox)
obj <- mt_fscore(f = 1)
evaluate(obj, c(TRUE, FALSE, TRUE), c(TRUE, TRUE, TRUE))
```

`mt_precision`*Precision Calculator*

Description

Class for precision calculation

Usage

```
mt_precision()
```

Value

Metric object

Examples

```
library(daltoolbox)
obj <- mt_precision()
evaluate(obj, c(TRUE, FALSE, TRUE), c(TRUE, TRUE, TRUE))
```

`mt_recall`*Recall Calculator*

Description

Class for recall calculation

Usage

```
mt_recall()
```

Value

Metric object

Examples

```
library(daltoolbox)
obj <- mt_recall()
evaluate(obj, c(TRUE, FALSE, TRUE), c(TRUE, TRUE, TRUE))
```

`mt_rocauc`*ROC AUC Calculator*

Description

Class for ROC AUC calculation

Usage

```
mt_rocauc()
```

Value

Metric object

Examples

```
library(daltoolbox)
obj <- mt_rocauc()
evaluate(obj, c(0.9, 0.2, 0.8, 0.1), factor(c(TRUE, TRUE, FALSE, FALSE)))
```

mv_dist_based

*Multivariate Distribution Based Drifter sub-class***Description**

Implements Multivariate Distribution Based drift detectors. These detectors consume one row of a data frame at a time and monitor its numeric columns.

Usage

```
mv_dist_based(target_feat = NULL)
```

Arguments

`target_feat` Names of the columns to be monitored. NULL (the default) monitors every numeric column.

Value

Drifter object

Examples

```
obj <- mv_dist_based(target_feat = c("serie1", "serie2"))
obj$target_feat
```

nrm_base

*Normalizer base class***Description**

Ancestor class for normalization techniques.

Usage

```
nrm_base(norm_class)
```

Arguments

`norm_class` Normalizer class

Details

This constructor was named `norm()` up to version 1.2.x. It was renamed to `nrm_base()` because `norm()` masks `base::norm()` once the package is attached.

Value

Norm object

Examples

```
library(daltoolbox)
obj <- nrm_base(norm_class = minmax())
class(obj)
```

nrm_memory

Memory Normalizer

Description

Normalizer that keeps its own memory of the data seen so far, so that the normalization parameters are estimated over the whole history rather than over the most recent batch alone.

Usage

```
nrm_memory(norm_class = daltoolbox::minmax(), max_size = NULL)
```

Arguments

norm_class	Normalizer class
max_size	Maximum number of rows kept in the memory; the oldest rows are dropped first. NULL (the default) keeps the whole history, so set it on long streams to bound memory.

Value

Norm object

Examples

```
library(daltoolbox)

obj <- nrm_memory(norm_class = minmax())
obj <- fit(obj, data.frame(x = c(1, 2, 3)))
transform(obj, data.frame(x = c(1, 2, 3)))
```

reset_state	<i>Reset State</i>
-------------	--------------------

Description

Reset Drifter State

Usage

```
reset_state(obj)
```

Arguments

obj Drifter object

Value

updated Drifter object

Examples

```
model <- dfr_ddm()
model <- reset_state(model)
model$drifted
```

st_drift_examples	<i>Synthetic time series for concept drift detection</i>
-------------------	--

Description

A list of univariate and multivariate time series for drift detection. Each element is a data frame:

- `uv_virtual_drift`: univariate series (`serie`) with a change in the feature distribution at four known points, flagged in event;
- `uv_vct_real_drift`: univariate series with a virtual change followed by a real drift;
- `uv_vdp_real_drift`: univariate series with a real drift and no change in the feature distribution;
- `mv_real_drift`: bivariate series with a real drift;
- `mv_vct_real_drift`: bivariate series with a virtual change followed by a real drift;
- `mv_vdp_real_drift`: bivariate series with a real drift and no change in the feature distribution.

Except for `uv_virtual_drift`, every data frame holds the series (`serie`, or `serie1` and `serie2`), a logical target, the index `i` and a drift column flagging the change point.

Usage

```
data(st_drift_examples)
```

Format

A list of six data frames with 500 rows each.

Source

heimdall package

References

heimdall package

Examples

```
data(st_drift_examples)
dataset <- st_drift_examples$mv_real_drift
which(dataset$drift == 1)
```

stealthy

Stealthy

Description

Ancestor class for drift adaptive models. It wraps a predictive model and a drift detector, retraining the model whenever a drift is reported.

Usage

```
stealthy(
  model,
  drift_method,
  norm_class = daltoolbox::zscore(),
  warmup_size = 100,
  th = 0.5,
  target_uni_drifter = FALSE,
  incremental_memory = FALSE,
  active_warmup = FALSE,
  class_balance = "inactive",
  obsolete_model = "null",
  reporting = FALSE,
  history_size = NULL,
  verbose = FALSE
)
```

Arguments

<code>model</code>	The algorithm object to be used for predictions
<code>drift_method</code>	The algorithm object to detect drifts
<code>norm_class</code>	Class used to perform normalization
<code>warmup_size</code>	Number of rows used to warmup the drifter. No drift will be detected during this phase
<code>th</code>	The threshold to be used with classification algorithms
<code>target_uni_drifter</code>	Passes the prediction target to the drifts as the target feat when the drifter is univariate and <code>dist_based</code> . The target must be binary (logical, 0/1 or a factor with those labels).
<code>incremental_memory</code>	If true, the model will retrain with all available data whenever the fit is called. If false, it only retrains when a drift is detected.
<code>active_warmup</code>	If true, the models will be retrained for each batch while the number of rows for the <code>warmup_size</code> is not met.
<code>class_balance</code>	Class balancing strategy. "buffer" sets a buffer with the last <code>warmup_size/2</code> rows for each class. "inactive" does nothing.
<code>obsolete_model</code>	Strategy used by <code>predict()</code> while no model is fitted. One of "null" (default, returns all FALSE), "majority" (majority class classifier) or "last_model" (keeps using the previous model).
<code>reporting</code>	If TRUE, every batch given to <code>fit()</code> (x and y) is accumulated in <code>drifter_input</code> .
<code>history_size</code>	Maximum number of rows kept in <code>drifter_output</code> and <code>drifter_input</code> ; the oldest rows are dropped first. NULL (the default) keeps the whole stream, so set it on long streams to bound memory.
<code>verbose</code>	if TRUE shows drift messages

Details

The drifter decides what it monitors through its own `target_feat`, given in terms of the one-hot encoded and normalized columns: a univariate drifter with a `target_feat` receives that column, a univariate drifter without one receives the row mean of every column, and multivariate, multi-criteria and dummy drifters receive the selected columns (every column when their `target_feat` is NULL).

The batch is evaluated before the model is updated: error-based detectors see prequential residuals, computed with the model as it was at the beginning of the call, and the detector is fed the incoming batch exactly once per call.

`obj$drifter_output` holds the diagnostics reported by the drifter (for example the window width of ADWIN or the p-values of KSWIN; see the `drifter_output` of each detector), with one row per row of x across every call to `fit()`. Rows are NA while the drifter is not monitoring (during the warmup, or after a drift until the model is retrained) and for observations it skipped, such as error-based rows without a prediction.

Value

Stealthy object

Examples

```
# See ?dfr_ddm for a drift detector that can be plugged here, and the
# online prediction walkthroughs at
# https://github.com/cefet-rj-dal/heimdall/tree/main/examples/2_online_prediction
```

update_state	<i>Update State</i>
--------------	---------------------

Description

Update Drifter State with a new observation.

Usage

```
update_state(obj, value, ...)
```

Arguments

obj	Drifter object
value	a value that represents a processed batch
...	optional arguments

Value

a list with two elements: obj, the updated Drifter object, and drift, a logical flag indicating whether a drift was detected in this call

Examples

```
model <- dfr_ddm()
output <- update_state(model, 0)
output$drift
```

Index

* datasets

st_drift_examples, 35

base::norm(), 33

base::saveRDS(), 3

base::set.seed(), 18

dfr_adwin, 3

dfr_aedd, 4

dfr_cusum, 6

dfr_ddm, 7

dfr_ecdd, 9

dfr_eddm, 11

dfr_hddm, 12

dfr_inactive, 14

dfr_kldist, 15

dfr_kswin, 17

dfr_lbdd, 19

dfr_lbdd(), 22

dfr_mcdd, 21

dfr_mcdd(), 20

dfr_multi_criteria, 23

dfr_page_hinkley, 24

dfr_passive, 26

dist_based, 27

drifter, 27

dummy, 28

error_based, 28

fit.drifter, 29

metric, 30

mt_accuracy, 30

mt_fscore, 31

mt_precision, 31

mt_recall, 32

mt_rocauc, 32

mv_dist_based, 33

nrm_base, 33

nrm_memory, 34

reset_state, 35

st_drift_examples, 35

stats::ks.test(), 18

stats::shapiro.test(), 22

stealthy, 36

update_state, 38