Package 'SeqDetect'

January 20, 2025

Type Package

Title Sequence and Latent Process Detector

Version 1.0.7

Date 2020-03-02

Author Dalibor Krleža

Maintainer Dalibor Krleža <dalibor.krleza@fer.hr>

Description Sequence detector in this package contains a specific automaton model that can be used to learn and detect data and process sequences.
Automaton model in this package is capable of learning and tracing sequences. Automaton model can be found in Krleža, Vrdoljak, Brčić (2019) <doi:10.1109/ACCESS.2019.2955245>.
This research has been partly supported under Competitiveness and Cohesion Operational Programme from the European Regional and Development Fund, as part of the Integrated Anti-Fraud System project no. KK.01.2.1.01.0041. This research has also been partly supported by the European Regional Development Fund under the grant KK.01.1.1.01.0009.

License LGPL-3

Encoding UTF-8

Depends R (>= 3.4.0), Rcpp (>= 1.0.3), eventdataR

Imports igraph, dplyr, methods

Suggests xtable

LinkingTo Rcpp

RcppModules ETT

NeedsCompilation yes

VignetteBuilder xtable, dplyr

SystemRequirements C++14

Repository CRAN

Date/Publication 2020-03-02 14:30:05 UTC

Contents

`bpi_challenge_2019_test1` *BPI 2019 challenge test*

Description

A single sales process flow from the BPI 2019 challenge event log was taken to perform the Sequence Detector testing. The results are available in [1].

Usage

```
bpi_challenge_2019_test1()
```

Value

None

References

[1] D. Krleža, B. Vrdoljak, and M. Brčić, Latent Process Discovery using Evolving Tokenized Transducer, *IEEE Access*, vol. 7, pp. 169657 - 169676, Dec. 2019

`classify` *Pre-classifying method*

Description

An abstract method that needs to be implemented by classes that derive HSC_PC. It performs classification on the input event stream. See the SeqDetect vignette for details on how to implement a HSC_PC derived class.

Usage

```
classify(x, stream, ...)
```

Arguments

x	(HSC_PC) - A pre-classifier object
stream	(data.frame) - An input event stream
...	An additional list of parameters needed for the used pre-classifier.

Value

(data.frame) - An output, a consolidated stream. Each row in the output data.frame must have *.clazz* field, containing the row classification value.

`cleanKeys,HybridSequenceClassifier-method`
Sequence Detector Method: clean keys and tokens in machines (ETTs)

Description

Sequence Detector method for removing tokens.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
cleanKeys(machine_id=NULL)
```

Arguments

machine_id	(character) - An identifier of the machine (ETT) whose token needs to be removed. If NULL, all machines tokens are removed.

See Also

HybridSequenceClassifier

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                 alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
       "sequence_id","sequence_id","product",pc,pp)
hsc$process(input_streams)
hsc$cleanKeys()
```

---

clone,HybridSequenceClassifier-method

Sequence Detector Method: clone the Sequence Detector object

Description

Sequence Detector method for cloning. Clones the Sequence Detector object and all its ETTs.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
clone()
```

See Also

HybridSequenceClassifier

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                 alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
       "sequence_id","sequence_id","product",pc,pp)
hsc$process(input_streams)
```

```
tt <- data.frame(product=c("P672","P113","P983","P23872","P5","P672","P2982","P983","P672",
                           "P991","P983","P113","P2982","P344"),
                 sales=c(2,11,12,98,8,18,298,16,24,25,18,16,43,101),alert=NA)
test_streams <- list(stream=tt)
hsc2 <- hsc$clone()
hsc2$process(test_streams,learn=FALSE)
```

`compressMachines,HybridSequenceClassifier-method`
Sequence Detector Method: compress machines (ETTs)

Description

Sequence Detector method for compressing machines by isolating common isomorphic sub-structured into child ETTs. See the SeqDetect vignette for details and examples.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
compressMachines(ratio=0.5)
```

Arguments

`ratio`	(numeric) - A minimal isomorphic overlap between ETTs to be eligible for compression. Using this parameter too low (e.g. <0.5) might lead to overfragmentation of ETTs.

See Also

`HybridSequenceClassifier`

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                 alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
       "sequence_id","sequence_id","product",pc,pp)
hsc$process(input_streams)
hsc$compressMachines()
```

c_to_string *String list formatting function*

Description

A method that formats an input list made of strings into a single output string. The output string is formatted as *[e1,e2,...,en]*.

Usage

```
c_to_string(var)
```

Arguments

var	(list) - A string list

Value

(character) - An output string made of the input list elements, formatted as *[e1,e2,...,en]*.

deserializeFromList *Sequence Detector Method: deserialize Sequence Detector object from external list*

Description

Sequence Detector method for deserializing from a list.

Usage

```
deserializeFromList(l)
```

Arguments

l	(list) - A list containing a Sequence Detector details.

Value

(HybridSequenceClassifier) - Returns a deserialized Sequence Detector object.

See Also

HybridSequenceClassifier-class, serializeToList,HybridSequenceClassifier-method

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                 alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
       "sequence_id","sequence_id","product",pc,pp)
hsc$process(input_streams)

hsc_list <- hsc$serializeToList()
saveRDS(hsc_list,"test_list.RDS")


new_hsc_list <- readRDS("test_list.RDS")
file.remove("test_list.RDS")
hsc2 <- deserializeFromList(new_hsc_list)
```

---

getMachineIdentifiers,HybridSequenceClassifier-method

Sequence Detector Method: retrieve machine identifiers (ETTs)

Description

Sequence Detector method for retrieving list of machine identifiers.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
getMachineIdentifiers()
```

Value

(list) A list of strings, represeting machine identifiers.

See Also

HybridSequenceClassifier

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                 alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
```

```
       "sequence_id","sequence_id","product",pc,pp)
res <- hsc$process(input_streams)
message(hsc$getMachineIdentifiers())
```

HSC_PC *Abstract pre-classifier class*

Description

All pre-classifiers must inherit this class. A pre-classifier instance cannot be directly created by this abstract class.

See Also

HSC_PC_None,HSC_PC_Attribute,HSC_PC_Binning

HSC_PC_Attribute *Attribute pre-classifier*

Description

Extends the HSC_PC abstract class.

Usage

```
HSC_PC_Attribute(field)
```

Arguments

field	(character) - Field taken as the classification value from the input event stream.

Details

A pre-classifier takes classification from the predefined field in the input event stream and copies these values to the *.clazz* field. The rest of the input event stream remains unmodified.

Examples

```
event_stream <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                           sales=c(2,12,18,16,18,24,8),
                           alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
pc <- HSC_PC_Attribute("sales")
cons_stream <- classify(pc,event_stream)
```

HSC_PC_Binning *Binning pre-classifier*

Description

Extends the HSC_PC abstract class.

Usage

```
HSC_PC_Binning(min_value, max_value, bins, value_field)
```

Arguments

min_value	(numeric) - Minimal value.
max_value	(numeric) - Maximal value:
bins	(integer) - A number of bins that needs to be created.
value_field	(character) - The name of the value field in the input event stream.

Details

A pre-classifier takes performs binning on a value field of the input event stream.

Examples

```
event_stream <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                           sales=c(2,12,18,16,18,24,8),
                           alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
pc <- HSC_PC_Binning(0,100,40,"sales")
cons_stream <- classify(pc,event_stream)
# Minimal value = 0, Maximal value = 100, 40 bins, values taken from the field named *sales*
```

HSC_PC_None *Straight-through pre-classifier*

Description

Extends the HSC_PC abstract class.

Usage

```
HSC_PC_None()
```

Details

A pre-classifier class that does not contain any classifier. It passes an input event stream straight through without any modifications. The only thing is to check whether the input event stream contains *.clazz* field, which should carry classification and input symbols for Sequence Detector ETTs.

Examples

```
event_stream <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                           sales=c(2,12,18,16,18,24,8),
                           alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"),
                           .clazz=c(2,12,18,16,18,24,8))
pc <- HSC_PC_None()
cons_stream <- classify(pc,event_stream)
```

HSC_PP *Pre-processor top-level class*

Description

Class that needs to be derived to create new pre-processors. A pre-processor can be directly instantiated from the HSC_PP class.

Usage

```
HSC_PP(fields, timestamp_field, create_unique_key = FALSE,
  auto_id = FALSE)
```

Arguments

fields	(vector) - The complete list of fields in the input data streams that needs to be present in the output event stream
timestamp_field	(character) - The name of the sequencing field. Could be autogenerated by the pre-processor, or already present in the input data streams. Used for ordering of the output event stream.
create_unique_key	(logical) - If TRUE, the pre-processor adds field named *.key* to the output event stream comprising a unique key (1) for all data items.
auto_id	(logical) - If TRUE, the pre-processor generates autoincremented values and assigns then to the *timestamp_field*. Can be used when input data streams do not comprise any timing information.

Details

Example 1
`pp <- HSC_PP(c("product","time","sales"),"time")` - Creates a new HSC_PP pre-processor that uses *time* field for ordering of the output event stream.

Example 2
`pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)` - Creates a new HSC_PP pre-processor that has no time field. Instead, the pre-processor adds the *sequence_id* field and generates autoincremented values for it.

Example 3
`pp <- HSC_PP(c("sequence_val"),"sequence_id",create_unique_key=TRUE,auto_id=TRUE)` - Creates a new HSC_PP pre-processor that has no time and no key field. The pre-processor adds the *sequence_id* field and generates autoincremented values for it. Also, the *.key=1* column is added to all output events.

HybridSequenceClassifier-class

Sequence Detector

Description

The Sequence Detector class.

Details

Instantiates a Sequence Detector object. Constructor takes a number of parameters that define pre-processing and pre-classification stages, as well as the structure of the input consolidated data stream. These stages can be redefined again later using `setInputDefinitions,HybridSequenceClassifier-method` method. See the SeqDetect vignette for examples.

Fields

fields (vector, character) - A vector of all relevant consolidated data stream fields.

timestamp_start_field (character) - A name of the field having starting time point values.

timestamp_finish_field (character) - A name of the field having finishing time point values.

context_field (character) - A name of the context identifier field (key field). If NULL, then *.key* field is used for retrieving context identifier values.

preclassifier (HSC_PC) - A pre-classifier object. If NULL, the Sequence Detector creates new *HSC_PC_None* pre-classifier, which means that the input consolidated data stream must have *.clazz* field for retrieving classification values (input symbols in the underlying ETTs).

preprocessor (HSC_PP) - A pre-processing object. If NULL, the Sequence Detector creates new *HSC_PP* pre-processor having the same fields as define in the *fields* parameter, and ordering timestamp field as defined in *timestamp_start_field*.

decay_descriptors (list) - A list of decay descriptors. If NULL, token decay machanism is not used. Descriptor structure can be seen in vignettes.

pattern_field (character) - A name of the field having output symbol values, i.e., relational ETT classification output.

time_series_sequence_stats (logical) - If TRUE, ETTs are instructed to create sequence statistics. This is used whe having input time-series data streams. If FALSE, the sequence statistics are not created.

reuse_states (logical) - The parameter defined in [1]. ETTs are created so that each ETT have a state that represents each input symbol.

parallel_execution (logical) - Force parallel execution of ETTs in the Sequence Detector object. Useful when we expect higher number of ETTs in the same Sequence Detector.

Methods

`cleanKeys(machine_id=NULL)` Sequence Detector method for removing tokens and keys
`cleanKeys,HybridSequenceClassifier-method`

`clone()` Sequence Detector method for cloning
`clone,HybridSequenceClassifier-method`

`compressMachines(ratio=0.5)` Sequence Detector method for compressing the underlying set of ETTs
`compressMachines,HybridSequenceClassifier-method`

`getMachineIdentifiers()` Sequence Detector method for retrieving identifiers for the underlying set of ETTs
`getMachineIdentifiers,HybridSequenceClassifier-method`

`induceSubmachine(threshold, isolate=FALSE)` Sequence Detector method for performing statistical projections on the underlying set of ETTs
`induceSubmachine,HybridSequenceClassifier-method`

`mergeMachines()` Sequence Detector method for merging the underlying set of ETTs
`mergeMachines,HybridSequenceClassifier-method`

`plotMachines(machine_id=NULL)` Sequence Detector method for plotting the underlying set of ETTs
`plotMachines,HybridSequenceClassifier-method`

`printMachines(machine_id=NULL, state=NULL, print_cache=TRUE, print_keys=TRUE)` Sequence Detector method for printing the underlying set of ETTs to the R console
`printMachines,HybridSequenceClassifier-method`

`process(streams, learn=TRUE, give_explain=TRUE, threshold=NULL, debug=FALSE, out_filename=NULL, ...)` Sequence Detector method for processing an input streams slice
`process,HybridSequenceClassifier-method`

`serialize()` Sequence Detector method for serializing the underlying set of ETTs definitions
`serialize,HybridSequenceClassifier-method`

`serializeToList()` Sequence Detector method for serializing the underlying set of ETTs definitions to the list
`serializeToList,HybridSequenceClassifier-method`

`setOutputPattern(states=c(), transitions=c(), pattern, machine_id=NULL)` Sequence Detector method for setting the output alphabet to the underlying set of ETTs
`setOutputPattern,HybridSequenceClassifier-method`

`setPreprocessor(preprocessor)` Sequence Detector method for setting the pre-processor
`setPreprocessor,HybridSequenceClassifier-method`

`setPreclassifier(preclassifier)` Sequence Detector method for setting the pre-classifier
`setPreclassifier,HybridSequenceClassifier-method`

`setInputDefinitions(fields, timestamp_start_field, timestamp_finish_field, context_field=NULL, precla`
Sequence Detector method for redefining the input definitions
`setInputDefinitions,HybridSequenceClassifier-method`

References

[1] D. Krleža, B. Vrdoljak, and M. Brčić, Latent Process Discovery using Evolving Tokenized Transducer, *IEEE Access*, vol. 7, pp. 169657 - 169676, Dec. 2019

`induceSubmachine,HybridSequenceClassifier-method`
Sequence Detector Method: ETT projection

Description

Sequence Detector method for ETT projections. See the SeqDetect vignette for proper usage and cases. All projection changes are performed on the same Sequence Detector object.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
induceSubmachine(threshold,isolate=FALSE)
```

Arguments

`threshold`	(integer) - A threshold for the ETT projection. All transitions that have invocation statistic above the threshold are moved to a submachine.
`isolate`	(logical) - After the regular sequences are moved the the submachine, the original parent can be removed, leaving only the most regular sequences. If TRUE, the parent ETT is removed and only the most regular sequences are left.

Value

Returns:
TRUE - projection was performed successfully
FALSE - no projection was performed.

See Also

`HybridSequenceClassifier`

Examples

```
st <- data.frame(product=c("P1","P2"),sales=c(5,76),alert=c(NA,NA))
for(i in 1:400) {
  st <- rbind(st,data.frame(product=c("P1","P2"),sales=c(10,58),alert=c(NA,NA)))
  st <- rbind(st,data.frame(product=c("P1","P2"),sales=c(20,31),alert=c(NA,NA)))
}
st <- rbind(st,data.frame(product=c("P1","P2"),sales=c(30,11),
                          alert=c("Sequence 1","Sequence 2")))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales","alert"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Attribute("sales")
hsc <- HybridSequenceClassifier(c("sequence_id","product","sales","alert"),"sequence_id",
                                "sequence_id",context_field="product",preclassifier=pc,
                               preprocessor=pp,reuse_states=TRUE,pattern_field="alert")
hsc$process(input_streams,learn=TRUE)
hsc$cleanKeys()
hsc$induceSubmachine(200,isolate=TRUE)
hsc$printMachines()
```

---

mergeMachines,HybridSequenceClassifier-method

Sequence Detector Method: merge machines (ETTs)

Description

Sequence Detector method for merging machines. See the SeqDetect vignette for details and examples.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
mergeMachines()
```

See Also

HybridSequenceClassifier

Examples

```
ldf1 <- data.frame(product=c("P1","P1","P1","P1"),sequence_id=c(1,3,5,7),
                   sales=c(5,76,123,1),alert=c(NA,NA,NA,"Alert P1"))
ldf2 <- data.frame(product=c("P2","P2","P2","P2"),sequence_id=c(2,4,6,8),
                   sales=c(21,76,123,42),alert=c(NA,NA,NA,"Alert P2"))
input_streams <- list(stream1=ldf1,stream2=ldf2)
pp <- HSC_PP(c("product","sales","alert","sequence_id"),"sequence_id")
pc <- HSC_PC_Attribute("sales")
hsc <- HybridSequenceClassifier(c("sequence_id","product","sales","alert"),
                                "sequence_id","sequence_id",context_field="product",
                                 preclassifier=pc,preprocessor=pp,reuse_states=TRUE,
```

```
                                        pattern_field="alert")
hsc$process(input_streams,learn=TRUE)
hsc$cleanKeys()
hsc$mergeMachines()
hsc$printMachines()
```

---

plotMachines,HybridSequenceClassifier-method

Sequence Detector Method: plot machines (ETTs)

Description

Sequence Detector method for plotting of machines in the Sequence Detector object. Plotting is following the output symbols of the states and transitions. For machines that don't have a small output alphabet could not be plotted fully and correctly.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
plotMachines(machine_id=NULL)
```

Arguments

machine_id	(character) - A machine identifier that needs to be plotted. If NULL, all machines are plotted.

See Also

HybridSequenceClassifier

Examples

```
ldf1 <- data.frame(product=c("P1","P1","P1","P1"),sequence_id=c(1,3,5,7),
                   sales=c(5,76,123,1),alert=c(NA,NA,NA,"Alert P1"))
ldf2 <- data.frame(product=c("P2","P2","P2","P2"),sequence_id=c(2,4,6,8),
                   sales=c(21,76,123,42),alert=c(NA,NA,NA,"Alert P2"))
input_streams <- list(stream1=ldf1,stream2=ldf2)
pp <- HSC_PP(c("product","sales","alert","sequence_id"),"sequence_id")
pc <- HSC_PC_Attribute("sales")
hsc <- HybridSequenceClassifier(c("sequence_id","product","sales","alert"),
                                  "sequence_id","sequence_id",context_field="product",
                                   preclassifier=pc,preprocessor=pp,reuse_states=TRUE,
                                       pattern_field="alert")
hsc$process(input_streams,learn=TRUE)
hsc$cleanKeys()
hsc$mergeMachines()
hsc$plotMachines()
```

`preprocess` *Pre-processing method*

Description

A method that all pre-processor classes need to implement. It is the code that aggregates and consolidates input data streams into one output event stream.

Usage

```
preprocess(x, streams, ...)
```

Arguments

`x`	The pre-processor object.
`streams`	A named list that comprises input data streams. Each input data stream is a data frame comprising fields declared while creating the HSC_PP object.
`...`	An additional list of parameters that can be used by the pre-proccessor.

Details

Input streams can be created as

```
streams -> list(stream1=x1,stream2=x2,....)
```

where *x1* is a data frame and *stream1* is the name of the stream. All examnples can be seen in the SeqDetect vignette.

Value

Returns a list that comprises:

- *obj* - A returning pre-processor object. Passed in the subsequent invocation as *x*.
- *res* - An output event stream. A resulting data frame representing the output event stream that is ordered according to the timestamp / sequence field and comprises all declared fields.

printMachines,HybridSequenceClassifier-method
Sequence Detector Method: printout machines (ETTs)

Description

Sequence Detector method for printing out the machines (ETTs) in the Sequence Detector object. See The SeqDetect vignette for proper usage and cases.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
printMachines(machine_id=NULL,state=NULL,print_cache=TRUE,print_keys=TRUE)
```

Arguments

machine_id	(character) - If defined, printout only machine that has the supplied identifier. If NULL, printout all machines.
state	(character) - If defined, printout only states that have the supplied identifier. If NULL, printout all states.
print_cache	(logical) - Switch for printout of the cache. If FALSE, the cache printout is omitted. The cache can be quite big for each machine and state, and could potentially blur the printout.
print_keys	(logical) - Switch for printout of the current token set. If FALSE, the token set printout is omitted. The number of tokens can be considerable, and could potentially blur the printout.

See Also

HybridSequenceClassifier

Examples

```
ldf1 <- data.frame(product=c("P1","P1","P1","P1"),sequence_id=c(1,3,5,7),
                   sales=c(5,76,123,1),alert=c(NA,NA,NA,"Alert P1"))
ldf2 <- data.frame(product=c("P2","P2","P2","P2"),sequence_id=c(2,4,6,8),
                   sales=c(21,76,123,42),alert=c(NA,NA,NA,"Alert P2"))
input_streams <- list(stream1=ldf1,stream2=ldf2)
pp <- HSC_PP(c("product","sales","alert","sequence_id"),"sequence_id")
pc <- HSC_PC_Attribute("sales")
hsc <- HybridSequenceClassifier(c("sequence_id","product","sales","alert"),
                                  "sequence_id","sequence_id",context_field="product",
                                   preclassifier=pc,preprocessor=pp,reuse_states=TRUE,
                                       pattern_field="alert")
hsc$process(input_streams,learn=TRUE)
hsc$cleanKeys()
hsc$mergeMachines()
hsc$printMachines()
```

process,HybridSequenceClassifier-method
Sequence Detector Method: processing input data streams

Description

Sequence Detector method for processing of input data streams. See the SeqDetect vignette for proper usage and cases.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
process(streams,learn=TRUE,give_explain=TRUE,threshold=NULL,debug=FALSE,
out_filename=NULL, ...)
```

Arguments

streams	(list, data.frame) - A named list that comprises input data streams. Each list element is a data frame that represents one input data stream.
learn	(logical) - Are ETTs in the Sequence Detector extendable? If TRUE, the Sequence Detector learns new sequences from the supplied input data streams.
give_explain	(logical) - Determines elements that will be returned by the method. If TRUE, output explanation and sequence statistical analysis will be returned as well.
threshold	(integer) - Needed threshold for the pushing mechanism. Pushing will work only for transitions that are above the supplied threshold. If NULL, all transitions are taken in consideration.
debug	(logical) - A switch for debug printout.
out_filename	(character) - A filename where the consolidated data stream should be written. The written file is in the CSV format. If NULL, file writing is skipped.
...	- An additional list of parameters passed into pre-processor and pre-classifier.

Value

A list that comprises the following elements:

- stream - The consolidated stream.

If *give_explain* is TRUE then an additional element is:

- explanation - Actual and potential output symbols for each data item of the consolidated data stream.

If *give_explain* is TRUE and *time_series_sequence_stats* is TRUE then an additional element is:

- sequences - The complete sequence statistics for the input time-series data.

See Also

HybridSequenceClassifier

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                 alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
       "sequence_id","sequence_id","product",pc,pp)
res <- hsc$process(input_streams)
message(res)
```

---

sales_dataset_test *Sales time-series test*

Description

Sales dataset taken from [2], which comprises 811 product one year sales quantities. We applied this dataset to test the Sequence Detector. The results are available in [1]. The results of the test are various statistics on detected sequences. The testing set of products is re-tested by simultaneously rising the projection threshold, until no more sequences could be detected or *max_th* parameter is reached.

Usage

```
sales_dataset_test(learning_set = 1:20, testing_set = 21:40,
  th_increment = 1, max_th = NULL)
```

Arguments

learning_set	(vector) - A set of products to learn ETTs in the Sequence Detector.
testing_set	(vector) - A set of products to test previously learned sales numbers.
th_increment	(integer) - A threshold increment between two tests.
max_th	(integer) - Maximal thershold for testing. When reached, no further tests and no further threshold increment is done. If NULL, re-testing is done while there are some sequences detected.

Value

A list that comprises sequence statistics for all tests and thresholds.

References

[1] D. Krleža, B. Vrdoljak, and M. Brčić, Latent Process Discovery using Evolving Tokenized Transducer, *IEEE Access*, vol. 7, pp. 169657 - 169676, Dec. 2019
[2] S. C. Tan and J. P. San Lau, Time series clustering: A superior alternative for market basket analysis, in Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013), Singapore, 2014, pp. 241–248.

`sepsis_dataset_test` *Sepsis dataset test*

Description

sepsis dataset is taken from the package *eventdataR* and used to test the Sequence Detector. The results are available in [1].

Usage

```
sepsis_dataset_test(induce_biomarker_decision_tree = TRUE,
  threshold = 75, debug = FALSE, hsc = NULL)
```

Arguments

`induce_biomarker_decision_tree`	(logical) - If FALSE, "Biomarker assessment" is one activity ignoring biomarker values. If TRUE, based on the biomarker values, several distinct "Biomarker assessment" activities are inferred.
`threshold`	(numeric) - Projection threshold.
`debug`	(logical) - Switch for debug printout.
`hsc`	(HybridSequenceClassifier) - An existing Sequence Detector that should be used instead of creating a new one.

Value

None

References

[1] D. Krleža, B. Vrdoljak, and M. Brčić, Latent Process Discovery using Evolving Tokenized Transducer, *IEEE Access*, vol. 7, pp. 169657 - 169676, Dec. 2019

serialize,HybridSequenceClassifier-method

Sequence Detector Method: serialize the Sequence Detector object

Description

Sequence Detector method for serializing. User needs to serialize the Sequence Detector object before saving. If not performed, Sequence Detector C++ part of the object is not saved properly, and cannot be restored later.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
serialize()
```

See Also

HybridSequenceClassifier

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                 alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
       "sequence_id","sequence_id","product",pc,pp)
res <- hsc$process(input_streams)
hsc$serialize()
#saveRDS(hsc,"test.RDS")
# Previous line is commented due to the CRAN checking policies
```

serializeToList,HybridSequenceClassifier-method

Sequence Detector Method: serialize and externalize Sequence Detector object

Description

Sequence Detector method for serializing to a list. The list can be saved, loaded and deserialized into a Sequence Detector object again using deserializeFromList function.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
serializeToList()
```

Value

Returns a list that comprises all Sequence Detector details.

See Also

HybridSequenceClassifier, deserializeFromList

Examples

```
st <- data.frame(product=c("P45","P134","P45","P134","P134","P45","P134"),
                 sales=c(2,12,18,16,18,24,8),
                alert=c(NA,NA,NA,NA,NA,"Alert P45","Alert P134"))
input_streams <- list(stream=st)
pp <- HSC_PP(c("product","sales"),"sequence_id",auto_id=TRUE)
pc <- HSC_PC_Binning(0,100,40,"sales")
hsc <- HybridSequenceClassifier(c("product","sales","sequence_id"),
       "sequence_id","sequence_id","product",pc,pp)
res <- hsc$process(input_streams)
hsc_list <- hsc$serializeToList()
#saveRDS(hsc_list,"test_list.RDS")
# Previous line is commented due to the CRAN checking policies
```

---

setInputDefinitions,HybridSequenceClassifier-method

Sequence Detector Method: redefine all input definitions

Description

A method for redefining the Sequence Detector input parameters. This method is useful when we want to reuse an existing Sequence Detector for a different set of input data streams. Based on the ETT definition, after the pre-processing and pre-classification stages we need to have a consolidated data frame that comprises context identifier, sequence fields (timestamps or incremental value) and classification values (an input symbol). Not everything can be redefined and needs to be left as defined at the time of Sequence Detector instantiation, such as decay descriptors.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
setInputDefinitions(fields, timestamp_start_field, timestamp_finish_field,
context_field=NULL, preclassifier=NULL, preprocessor=NULL, pattern_field=NULL)
```

Arguments

`fields`	(vector, character) - A vector of all relevant consolidated data stream fields.
`timestamp_start_field`	(character) - A name of the field having starting time point values.
`timestamp_finish_field`	(character) - A name of the field having finishing time point values.
`context_field`	(character) - A name of the context identifier field (key field). If NULL, then *.key* field is used for retrieving context identifier values.
`preclassifier`	(HSC_PC) - A pre-classifier object. If NULL, the Sequence Detector creates new *HSC_PC_None* pre-classifier, which means that the input consolidated data stream must have *.clazz* field for retrieving classification values (input symbols in the underlying ETTs).
`preprocessor`	(HSC_PP) - A pre-processing object. If NULL, the Sequence Detector creates new *HSC_PP* pre-processor having the same fields as define in the *fields* parameter, and ordering timestamp field as defined in *timestamp_start_field*.
`pattern_field`	(character) - A name of the field having output symbol values, i.e., relational ETT classification output.

`setOutputPattern,HybridSequenceClassifier-method`

Sequence Detector Method: assign output symbols

Description

Sequence Detector method for assigning output symbols to states and transitions. See the SeqDetect vignette for proper usage and cases.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
setOutputPattern(states=c(),transitions=c(),pattern,machine_id=NULL)
```

Arguments

`states`	(vector,character) - A character vector that comprises state identifiers. The supplied symbol (output alphabet, pattern parameter) is assigned to these states.
`transitions`	(vector,character) - A character vector that comprises transition identifiers. The supplied symbol (output alphabet, pattern parameter) is assigned to these transitions.
`pattern`	(character) - An output symbol, an element of the output alphabet, that needs to be assigned to supplied states and transitions.
`machine_id`	(character) - If defined, the output symbol assignment applies only to the machine having this identifier. If NULL, the output symbol assignment applies to all machines (ETTs) in this Sequence Detector object.

See Also

HybridSequenceClassifier

`setPreclassifier,HybridSequenceClassifier-method`
Sequence Detector Method: re-set the pre-classifier object

Description

Sequence Detector method for re-setting the pre-classifier object. This might be desirable when we want to use already existing Sequence Detector for new input data streams, having different structure.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
setPreclassifier(preclassifier)
```

Arguments

`preclassifier` (HSC_PC) - New pre-classifier object.

See Also

HybridSequenceClassifier

`setPreprocessor,HybridSequenceClassifier-method`
Sequence Detector Method: re-set the pre-processor object

Description

Sequence Detector method for re-setting the pre-processor object. This might be desirable when we want to use already existing Sequence Detector for new input data streams, having different structure.

Usage

```
## S4 method for signature 'HybridSequenceClassifier'
setPreprocessor(preprocessor)
```

Arguments

`preprocessor` (HSC_PP) - New pre-processor object.

See Also

HybridSequenceClassifier

`synthetic_test_agenda` *Synthetic process test*

Description

A synthetic process that was introduced in the process mining agenda [1]. The original event log introduced in [1] did not comprise any timestamps, and a process discovery algorithm was intended to infer this based on the event position in the log. ETT and new process discovery algorithms require events to have at least some sort of timing, and this was added for this test. It is worth noticing that the given event log has some parallel activities, which should be detected by the process discovery algorithm. The final results of this test are described in [2].

Usage

```
synthetic_test_agenda(label_aspect=1)
```

Arguments

`label_aspect`	(numeric) - A vector of all relevant consolidated data stream fields.

References

[1] W. M. P. van der Aalst and A. J. M. M. Weijters, Process mining: a research agenda, *Computers in Industry*, vol. 53, no. 2, pp. 231–244, Apr. 2004
[2] D. Krleža, B. Vrdoljak, and M. Brčić, Latent Process Discovery using Evolving Tokenized Transducer, *IEEE Access*, vol. 7, pp. 169657 - 169676, Dec. 2019

Index