

# DeviationSequenceLearn operator

This operator learns deviation of (each point of a) sequence. The sequence needs a counter so that the operator can distinguish multiple sequences and the values within a sequence. The operator uses the online-learn algorithm of the [DeviationLearn operator](#).

## Example

The operator gets the following input:

| Counter | Value |
|---------|-------|
| 0       | 8     |
| 1       | 5     |
| 2       | 20    |
| 0       | 4     |
| 1       | 6     |
| 2       | 22    |

This were two sequences, because a new sequence starts if the counter of the next tuple is smaller than the previous counter. Every sequence had three tuples. The output on port 1 would be the following:

| group | mean | standardDeviation |
|-------|------|-------------------|
| 0     | 8.0  | 0.0               |
| 1     | 5.0  | 0.0               |
| 2     | 20.0 | 0.0               |
| 0     | 6.0  | 2.828             |
| 1     | 5.5  | 0.707             |
| 2     | 21.0 | 1.414             |

## Parameters

- **attribute** Name of the attribute which should be analysed
- **sequencesToLearn** The number of (correct) sequences to learn from. The first x sequences will define the perfect sequence the others are compared to. If set to 0, the operator will not stop to learn (learn infinity sequences). Default is 0.
- **GROUP\_BY** To group the tuples into the single parts of the sequence.
- **fastGrouping** Use hash code instead of tuple compare to create group. Potentially unsafe!

## Example

The example PQL code shows, how to use the operator. The GROUP\_BY parameter is very important because it is used to distinguish the single values within one sequence.

```

#PARSER PQL
#RUNQUERY
/// Values above 50 will be 'true' (which means that the current sequence starts / runs) and smaller values to
'false' (means: sequence ended)
stateInfo = MAP({
    expressions = ['temp', ['temp > 50', 'state']]
},
    System.manual
)

/// The elements within one sequence will be counted (starts from 1 with each new sequence)
sequence = MAP({
    expressions = ['temp', 'counter(state)']
},
    stateInfo
)

/// The tuple which marks the end of the sequence (and itself is not part of the sequence) has the
counter_state_ 0 and will be filtered out
onlySequence = SELECT({PREDICATE = 'counter_state_ > 0'}, sequence)

/// Learn how a "normal" sequence is. The first 15 sequences will be learned and used as the definition of
"normal"
sequenceLearn = DEVIATIONSEQUENCELEARN({
    group_by = ['counter_state_'],
    attribute = 'temp',
    sequencesToLearn = 15
},
    onlySequence
)

/// Check, if the current tuple of this sequence differs from the normal tuples of the sequence at the
specific point of the sequence
sequenceAnalysis = DEVIATIONSEQUENCEANOMALYDETECTION({
    interval = 4.0,
    standardDeviationLearnAttribute = 'standardDeviation',
    group_by = ['group'],
    meanLearnAttribute = 'mean',
    valueDataAttribute = 'temp'
},
    0:sequenceLearn,
    1:sequenceLearn
)

```