SAS® Event Stream Processing 4.3: Using SAS® Event Stream Processing Analytics

Overview

SAS Event Stream Processing Analytics is a separately orderable and licensed package that enables you to use advanced analytics and machine learning techniques in an event stream processing model.

You use these analytics and techniques through the following window types:

score	Scores events using online algorithms packaged with SAS Event Stream Processing or with offline scoring models.
train	Generates scoring models for a class of online algorithms based on streaming event data.
calculate	Transforms data events using a variety of packaged algorithms.
model reader	Uses training algorithms outside SAS Event Stream Processing. You can get algorithms through CAS actions and then score them through the analytic store (ASTORE) binary file conversion to a scoring model.
model supervisor	Manages models for scoring inside SAS Event Stream Processing.

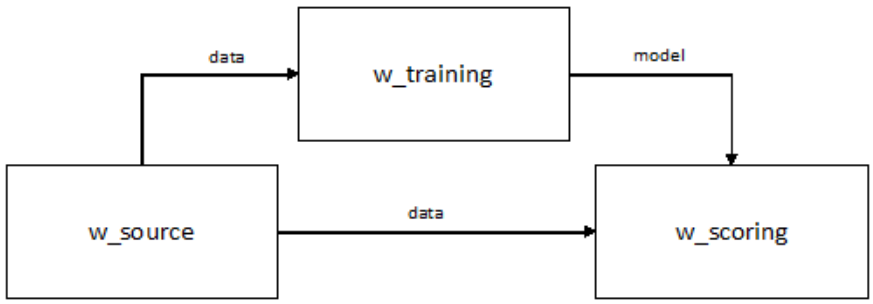
When you execute an algorithm, you must specify its name, parameter definitions, and input and output mapping properties. To obtain this information for a particular window type and algorithm combination:

- use the streaming analytics command-line utility.
- use HTTP requests to the ESP server through the RESTful API.

Incoming edges to any of these windows must specify a role that corresponds to the incoming event type.

role='data'	Sends data events between windows. A data event streams data to be processed by an analytical algorithm into the receiving window.
role='model'	Sends model events between windows. A model event, which has a fixed schema, streams model details into the receiving window.
role='request'	Sends request events between windows. A request event, which has a fixed schema, requests that a specific action be performed within the receiving window.

Consider the following arrangement of windows, which is common to training models:



The following code specifies roles for the edges between the windows:

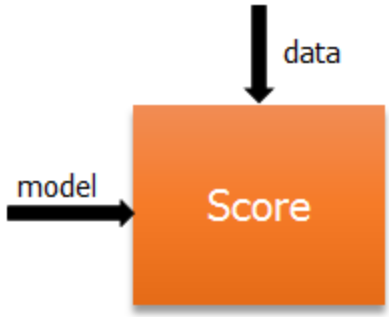
```
<edges>
    <edge source='w_source' target='w_training' role='data'/>
    <edge source='w_source' target='w_scoring' role='data'/>
    <edge source='w_training' target='w_scoring' role='model'/>
   </edges>
```

- The first edge specifies that a data event originating from the source window stream into the train window; you can set up the train window to process the data with a specified algorithm.
- The second edge specifies that a data event originating from the source window stream into the score window.
- The third edge specifies that a model event originating from the train window stream into the score window.

Streaming Analytics Window Types

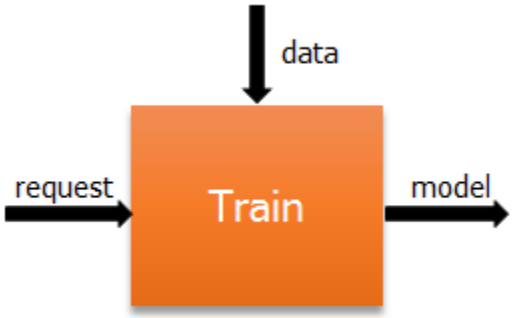
Score Window

The *score window* receives data events and model events and publishes events that contain scores on the incoming data. You can use the scores to generate predictions based on the trained model. (No role is assigned to the outgoing edges, so they do not appear in the diagram.)



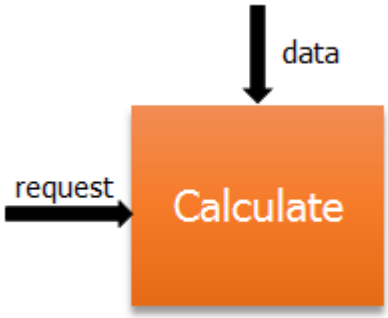
Train Window

The *train window* receives data events and publishes model events into another streaming analytics window. Often, the data is historical data from which to learn patterns. The data should contain both the outcome that you are trying to predict and related variables. A learning model uses that data to extract statistical patterns and build a model. The train window can also receive request events.



Calculate Window

The *calculate window* receives data events and publishes score data into output events. (No role is assigned to the outgoing edges, so they do not appear in the diagram.) The calculate window can also receive request events. Calculate windows combine training and scoring into a single window.



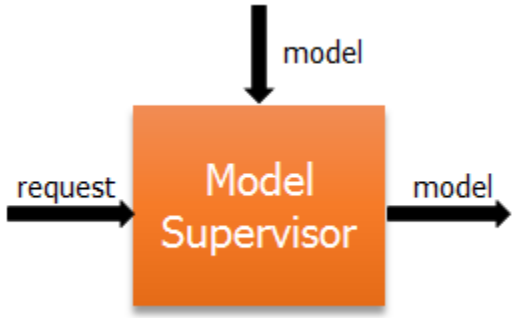
This window type is designed for data normalization and transformation methods, as well as for learning models that bundle training and scoring together.

Model Reader Window

The *model reader window* receives request events that include the location and type of an offline analytic store (ASTORE) model. This window type publishes a model event to another streaming analytics window.

Model Supervisor Window

The *model supervisor window* manages the flow of model events. Through input request events, you can control what model to deploy and when and where to deploy it. The model event is published to another streaming analytics window.



A model supervisor window can receive any number of model events. In a streaming analytics project, model events are typically sent by a train window or a model reader window. After receiving a model event, a model supervisor window processes and publishes events to other streaming analytics windows based on the model supervisor's deployment mode and on user requests.

The `<window-model-supervisor>` element has three properties:

- `name=`
- `deployment-policy=`

  `immediate`
  sends model events to any receiving window immediately after receiving them

  `on-demand`
  sends model events according to the requests specified at the command line

- `capacity=` specifies the number of current model events to keep. After the capacity is met, older model events are discarded.

Model Events

Model events stream model details into a score window or into a model supervisor window. You do not need to specify model event schema for train windows and model reader windows; it is implicit in the XML for the query. Internally, model events have a fixed schema, as follows:

```
<schema>
  <fields>
    <field name='model_id' type='int64' key='true'/>
    <field name='model_addr' type='int64'/>
    <field name='model_origin' type='string'/>
    <field name='model_token' type='int64'/>
    <field name='model_create_time' type='int64'/>
    <field name='model_perf' type='double'/>
  </fields>
</schema>
```

Field	Description
`model_id`	Specifies the ID assigned by a train window or by a model supervisor window.
`model_addr`	Specifies the address of the model descriptor in memory.
`model_origin`	Specifies the window name from which this model was first created.
`model_token`	Specifies a token to determine the receiver of this model event.
`model_create_time`	Specifies the timestamp when the model was first created.
`model_perf`	Specifies the performance metric of a model.

Request Events

Events that flow through request edges are called *request events*. You can inject request events into a source window with an adapter, or you can specify that they come from another window in the continuous query.

You specify the schema of a request event in the originating window. Request events have a fixed schema of an ID, a `req_key` field, and a `req_val` field.

```
<schema>
          <fields>
            <field name='req_id'  type='int64' key='true'/>
            <field name='req_key' type='string'/>
            <field name='req_val' type='string'/>
          </fields>
      </schema>
```

The first request event should always specify **`action`** as the `req_key`. The last request event should always specify an empty `req_key`, which indicates the end of a request. The parameters of the action are specified in the events between the first and last. In those middle events, the keys specify the names of parameters, and the values of `req_val` specify their values.

Request events can carry out the following actions:

Action	Window	Description
reconfig	calculate, train	Change the value of a property.
list	model supervisor	List all available models currently stored.
send	model supervisor	Send a model to a downstream window.
remove	model supervisor	Remove a model.
load	model reader	Load an offline ASTORE file into the event stream processing engine.

Consider the following series of events:

```
i,n,1,action,reconfig
i,n,2,arg1,val1
i,n,3,arg2,val2
i,n,4,,
```

All events are Insert (Normal):

1. The first event specifies **action** as the req_key and **reconfig** as the action to perform.
2. The second event specifies **arg1** as the req_key and **val1** as the req_val.
3. The third event specifies **arg2** as the req_key and **val2** as the req_val.
4. The fourth event has an empty req_key. This submits the reconfig request with arg1=val1 and arg2=val2.

Now consider the following model. It specifies **request** for the edge role between a source window named w_request and a calculate window named w_calculate:

```
<windows>
  ...
  <window-source name='w_request'>
    <schema>
      <fields>
        <field name='req_id'  type='int64' key='true'/>
        <field name='req_key' type='string'/>
        <field name='req_val' type='string'/>
      </fields>
    </schema>
  </window-source>

  <window-calculate name='w_calculate' algorithm='Summary'>
    <schema>
    ...
    </schema>
    <parameters>
      <properties>
        <property name="windowLength">4</property>
      </properties>
    </parameters>
    ...
  </window-calculate>
```

```
</windows>
...
<edges>
  ...
  <edge source='w_request' target='w_calculate' role='request'/>
</edges>
```

By injecting the following events into the `w_request` source window, you send a **`reconfig`** event to the `w_calculate` window. The request changes the value of `windowLength` from 4 (as defined in the properties of `w_calculate`) to 100.

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","100"
i,n,3,,
```

Streaming Analytics Command-Line Utility

SAS Event Stream Processing Analytics includes **`dfesp_analytics`**, a UNIX command-line utility that is located in **`$DFESP_HOME/bin`**. Use the utility to determine what algorithms are available to a window and what properties you must set to use the algorithms.

Command	Result
`$DFESP_HOME/bin/dfesp_analytics`	Returns a list of the utility's available options
`$DFESP_HOME/bin/dfesp_analytics –`*`window_type`*	Lists the algorithms available to the specified *window_type*: **`train`**, **`score`**, or **`calculate`**.
`$DFESP_HOME/bin/dfesp_analytics –`*`window_type`*` –algorithm `*`algorithm`*	For a specified *algorithm*, lists a window type's properties in text format. For example,`$DFESP_HOME/bin/dfesp_analytics -train –algorithm dbscan` returns the properties that are required to use the DBSCAN clustering algorithm in a train window, as well as the default values of those properties.
`$DFESP_HOME/bin/dfesp_analytics -`*`window`*` –algorithm `*`algorithm`*` –xml`	For a specified *algorithm*, lists a window type's properties in XML format.

Online Streaming Analytics Projects

Overview

Online streaming analytics projects use algorithms that are packaged with SAS Event Stream Processing Analytics and models that are trained in SAS Event Streaming Processing.

The following algorithms are packaged with SAS Event Stream Processing Analytics. They can be used in online projects.

Train and Score Windows
Streaming K-Means Clustering
Streaming DBSCAN Clustering

Calculate Window
Streaming Summary (Univariate Statistics)
Streaming Pearson Correlation
Segmented Correlation
Streaming Distribution Fitting
Short-Time Fourier Transform
Streaming Summary Statistics
Streaming Text Tokenization

Training and Scoring with K-means Clustering

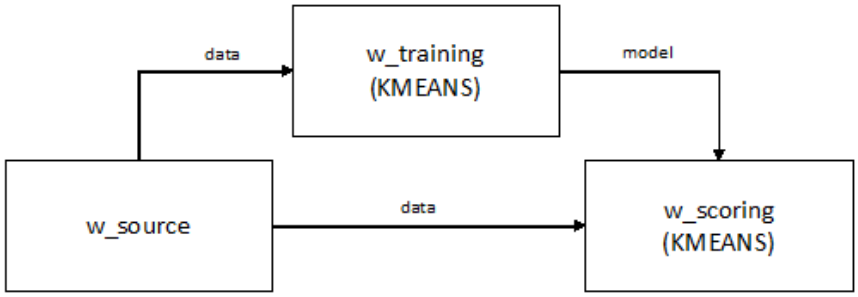
The classic k-means clustering algorithm performs two basic steps:

1. An assignment step in which data points are assigned to their nearest cluster centroid
2. An update step in which each cluster centroid is recomputed as the average of data points belonging to the cluster

The algorithm runs these two steps iteratively until a convergence criterion is met.

SAS Event Stream Processing Analytics includes k-means clustering as an algorithm for the train and score windows.

Consider the following example:



This continuous query includes the following:

- a source window that receives the data to be scored
- a train window that generates and periodically updates the k-means model
- a score window that performs the scoring

The source window `w_source` receives a data event that streams data from a file named `input.csv` through a file-and-socket connector. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; and a y coordinate of data named `y_c`.

```
<window-source name='w_source' insert-only='true'>
      <schema>
        <fields>
          <field name='id'  type='int64' key='true'/>
          <field name='x_c' type='double'/>
          <field name='y_c' type='double'/>
        </fields>
      </schema>
      <connectors>
        <connector class='fs' name='publisher'>
          <properties>
            <property name='type'>pub</property>
            <property name='fstype'>csv</property>
            <property name='fsname'>input.csv</property>
            <property name='transactional'>true</property>
            <property name='blocksize'>1</property>
          </properties>
        </connector>
      </connectors>
    </window-source>
```

The train window `w_training` looks at all observations and periodically generates a new clustering model using the k-means algorithm. Generated clustering model events are published to the score window `w_score`, where incoming events are clustered.

```
<window-train name='w_training' algorithm='KMEANS'>
      <parameters>
        <properties>
          <property name="nClusters">2</property>
          <property name="initSeed">1</property>
          <property name="dampingFactor">0.8</property>
          <property name="fadeOutFactor">0.05</property>
          <property name="disturbFactor">0.01</property>

          <property name="nInit">50</property>
          <property name="velocity">5</property>
          <property name="commitInterval">25</property>
        </properties>
      </parameters>
      <input-map>
        <properties>
          <property name="inputs"><![CDATA[x_c,y_c]]></property>
        </properties>
      </input-map>
    </window-train>
```

The following properties govern the k-means algorithm in the train window:

Property	Value Type	Property Type	Default Value	Description
Parameters				
`nClusters`	int32	Optional	`2`	Specifies the number of clusters $K(K > 0)$ to report.
`initSeed`	int32	Optional	`12345`	Specifies the random seed used during initialization when each point is assigned to a random cluster.
`dampingFactor`	double	Optional	`0.8`	Specifies the damping factor $\alpha\,(0 < \alpha < 1)$ for old data points. If the current time is T, data points arriving at time T would have weight 1, and data points arriving at time $T - t$ would have weight α^t.
`fadeOutFactor`	double	Optional	`0.05`	Specifies the factor $\theta\,(0 < \theta < 1)$ for determining whether an existing cluster is fading out. If a cluster weight is smaller than the maximal cluster weight among other clusters multiplied by θ, then this cluster is considered to be fading out.
`disturbFactor`	double	Optional	`0.01`	Specifies the factor $\delta\,(\delta > 0)$ for the disturbance when splitting a cluster. When an old cluster fades out, the cluster with the maximal weight is split into two, and both new clusters share half of its weight. If the old centroid is $\overrightarrow{C}$, the two new centroids are $\left(1+\delta\right)\cdot\overrightarrow{C}$ and $\left(1-\delta\right)\cdot\overrightarrow{C}$, respectively.
`nInit`	int64	Optional	`50`	Specifies the number of data events used during initialization.
`velocity`	int64	Optional	`1`	Specifies the number of events arriving at a single timestamp.
`commitInterval`	int64	Optional	`25`	Specifies the number of timestamps to elapse before committing a model to downstream scoring.
Input Mapping				
`inputs`	varlist	Required	No default value	Specifies the list of variable names used in clustering. Variable names are defined in the input schema of the source window, and they are separated by a comma in the list.

The score window `w_scoring` assigns a cluster number to each input event. The cluster number indicates which cluster the observation falls into according to the k-means clustering algorithm.

```
<window-score name='w_scoring'>
      <schema>
        <fields>
          <field name='id'  type='int64' key='true'/>
          <field name='x_c' type='double'/>
          <field name='y_c' type='double'/>
          <field name='seg' type='int32'/>
          <field name='min_dist' type='double'/>
          <field name='model_id' type='int64'/>
        </fields>
      </schema>
      <models>
```

```
        <online algorithm='KMEANS'>
            <input-map>
                <properties>
                      <property name="inputs"><![CDATA[id, x_c, y_c, seg, min_dist, model_id]]></property>
                </properties>
            </input-map>
        </online>
      </models>
    </window-score>
```

The following properties are unique to score windows for streaming k-means clustering:

Property	Value Type	Property Type	Default Value	Description
Input Mapping				
`inputs`	varlist	Optional	" "	Specifies the list of variable names used in clustering. Variable names are defined in the input schema of the source window, and they are separated by a comma in the list.
Output Mapping				
`labelOut`	variable	Optional	" "	Specifies the output variable name in the output schema that stores the cluster label.
`minDistanceOut`	variable	Optional	" "	Specifies the output variable name in the output schema that stores the distance to the nearest cluster. If not specified, the minimal distance column is not shown.
`modelIdOut`	variable	Optional	" "	Specifies the output variable name in the output schema that stores the ID of the model from which the score is computed. If not specified, the model ID column is not shown.

The edges are defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

```
    <edges>
    <edge source='w_source' target='w_training' role='data'/>
    <edge source='w_source' target='w_scoring' role='data'/>
    <edge source='w_training' target='w_scoring' role='model'/>
  </edges>
```

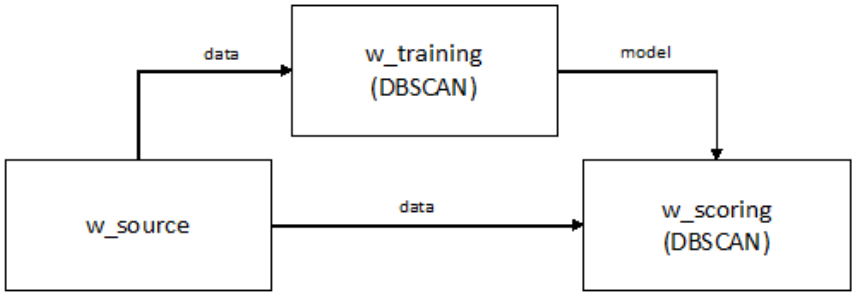
You can view the default values of the k-means algorithm parameter properties for the score and train windows with the streaming analytics command-line utility.

Training and Scoring with DBSCAN Clustering

DBSCAN is a density-based clustering approach. Given a set of data points, the algorithm tries to find connected high-density regions as clusters. To do that, it searches for a core point where the number of neighbors in its ε range is greater than or equal to μ. If such a core point exists, the algorithm visits its neighbors. If a neighbor point is also a core point, then the point is further extended. Otherwise, no more core points can be reached, and the algorithm starts with an unvisited core point and repeats the previous process until all points are visited. In the end, all points (core and non-core) that are reachable from a given core point form a cluster.

SAS Event Stream Processing Analytics includes DBSCAN clustering as an algorithm for the train and score windows.

Consider the following example:



This continuous query includes the following:

- a source window that receives data events that stream the data to be scored
- a train window that generates and periodically updates the DBSCAN model
- a score window that performs the scoring

The source window `w_source` receives a data event that streams data from a file named `input.csv` through a file-and-socket connector. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c;`, and a y coordinate of data named `y_c`.

```
<window-source name='w_source'>
      <schema>
        <fields>
          <field name='id'  type='int64' key='true'/>
          <field name='x_c' type='double'/>
          <field name='y_c' type='double'/>
        </fields>
      </schema>
      <connectors>
        <connector class='fs' name='publisher'>
          <properties>
            <property name='type'>pub</property>
            <property name='fstype'>csv</property>
            <property name='fsname'>input.csv</property>
            <property name='transactional'>true</property>
            <property name='blocksize'>1</property>
          </properties>
        </connector>
      </connectors>
    </window-source>
```

The train window `w_training` processes all observations and periodically generates a new clustering model using the DBSCAN algorithm.

```
<window-train name='w_training' algorithm='DBSCAN'>
      <parameters>
        <properties>
          <property name="epsilon">2.0</property>
          <property name="mu">3</property>
          <property name="beta">0.5</property>
          <property name="lambda">0.05</property>
          <property name="recluster">1</property>
```

```
        <property name="reclusterFactor">2.75</property>
        <property name="nInit">50</property>
        <property name="velocity">5</property>
        <property name="commitInterval">25</property>
      </properties>
    </parameters>
    <input-map>
      <properties>
        <property name="inputs"><![CDATA[x_c,y_c]]></property>
      </properties>
    </input-map>
  </window-train>
```

The following properties govern the DBSCAN algorithm in the train window:

Property	Value Type	Property Type	Default Value	Description
Parameters				
`epsilon`	double	Optional	`3.0`	Specifies the range of the neighborhood being considered. $(\varepsilon > 0)$
`mu`	int64	Optional	`4`	Specifies the weight of core micro clusters. $(\mu > 1)$
`beta`	double	Optional	`0.3`	Specifies the factor for μ to determine a micro cluster is p-mc or o-mc. $(0 < \beta \leq 1)$ and $(\beta \cdot \mu > 1)$
`lambda`	double	Optional	`0.02`	Specifies the decaying factor for the data weight. Assuming that the current time is T, data points arriving at time T have weight 1, and data points arriving at time $T - t$ have weight $2^{-\lambda t}$ $(\lambda > 0)$.
`recluster`	Boolean	Optional	`1`	Specifies whether reclustering (with offline weighted DBSCAN) is performed: valid values are 1 for true and 0 for false.
`reclusterFactor`	double	Optional	`2.0`	Specifies the factor (c) for ε used in reclustering. $(c \geq 2)$.
`nInit`	int64	Optional	`50`	Specifies the number of data events used during initialization.
`velocity`	int64	Optional	`1`	Specifies the number of events arriving at a single timestamp.
`commitInterval`	int64	Optional	`25`	Specifies the number of timestamps to elapse before committing a model to downstream scoring.
Input Mapping				
`input`	varlist	Required	No default value	Specifies the list of variable names used in clustering. Variable names are defined in the input schema, and they are separated by a comma in the list.

Generated clustering models are published to the score window `w_scoring`. This window assigns a cluster number to each input event. The cluster number indicates which cluster the observation falls into according to the DBSCAN algorithm.

```
<window-score name='w_scoring'>
      <schema>
        <fields>
          <field name='id'  type='int64' key='true'/>
          <field name='x_c' type='double'/>
          <field name='y_c' type='double'/>
          <field name='seg' type='int32'/>
          <field name='min_dist' type='double'/>
          <field name='model_id' type='int64'/>
        </fields>
      </schema>
      <models>
        <online algorithm='DBSCAN'>
            <input-map>
                <properties>
                      <property name="inputs"><![CDATA[id, x_c, y_c, seg, min_dist, model_id]]></property>
                </properties>
            </input-map>
        </online>
      </models>
    </window-score>
```

The following properties are unique to score windows for streaming DBSCAN clustering:

Property	Value Type	Property Type	Default Value	Description
Input Mapping				
`inputs`	varlist	Optional	" "	Specifies the list of variable names used in clustering. Variable names are defined in the input schema of the source window, and they are separated by a comma in the list.
Output Mapping				
`labelOut`	variable	Optional	" "	Specifies the output variable name in the output schema that stores the cluster label.
`minDistanceOut`	variable	Optional	" "	Specifies the output variable name in the output schema that stores the distance to the nearest cluster. If not specified, the minimal distance column is not shown.
`modelIdOut`	variable	Optional	" "	Specifies the output variable name in the output schema that stores the ID of the model from which the score is computed. If not specified, the model ID column is not shown.

The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

```
    <edges>
    <edge source='w_source' target='w_training' role='data'/>
    <edge source='w_source' target='w_scoring' role='data'/>
    <edge source='w_training' target='w_scoring' role='model'/>
  </edges>
```

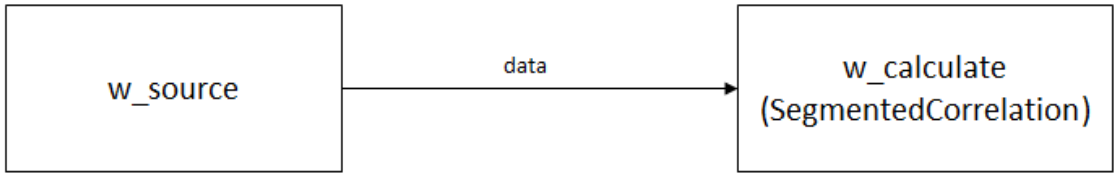
You can view the default values of the DBSCAN algorithm parameter properties for the score and train windows with the streaming analytics command-line utility.

Calculating Segmented Correlation

Segmented correlation is similar to autocorrelation. It specifies the correlation between the elements of a series and others from the same series that are separated from them by a specified interval. You can use segmented correlation to find repeating patterns, such as the occurrence of a signal obscured by noise.

The calculate window can calculate the segmented correlation of variable values streaming over time.

Consider the following example:



This continuous query includes the following:

- a source window that receives the data to be analyzed
- a calculate window that calculates the segmented correlation between of a variable from an incoming data stream and publishes the results

The source window `w_source` receives a data event that consists of the contents of a file named `input.csv`. It receives the event through a file-and-socket connector. The input stream is placed into four fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; a y coordinate of data named `y_c`; and an indicator variable named `indicator` that signals when a segment of the series begins and ends.

```
<window-source name='w_source' insert-only='true' autogen-key='false'>
     <schema>
      <fields>
       <field name='id'  type='int64' key='true'/>
       <field name='x_c' type='double'/>
       <field name='y_c' type='double'/>
       <field name='indicator' type='int64'/>
      </fields>
     </schema>
     <connectors>
      <connector class='fs' name='publisher'>
       <properties>
        <property name='type'>pub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>input/input.csv</property>
        <property name='transactional'>true</property>
        <property name='blocksize'>1</property>
       </properties>
      </connector>
     </connectors>
    </window-source>
```

The calculate window `w_calculate` receives data events from `w_source`. It publishes the calculated autocorrelation of the specified `x` variable according to the segmented correlation algorithm properties that are specified at the `window-calculate` level.

```
<window-calculate name="w_calculate" algorithm="SegmentedCorrelation">
      <schema>
          <fields>
          <field name='id' type='int64' key='true'/>
          <field name='y_c' type='double'/>
          <field name='x_c' type='double'/>
          <field name='corOut' type='double'/>
        </fields>
      </schema>
      <parameters>
        <properties>
          <property name="sampleSize">3</property>
          <property name="minSize">0</property>
          <property name="maxSize">100</property>
        </properties>
      </parameters>
      <input-map>
        <properties>
          <property name="x">x_c</property>
          <property name="indicator">indicator</property>
        </properties>
      </input-map>
      <output-map>
        <properties>
          <property name="corOut">corOut</property>
        </properties>
      </output-map>
     <connectors>
      <connector class='fs' name='sub'>
       <properties>
        <property name='type'>sub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>TEST_OUTPUT/output/result.out</property>
        <property name='snapshot'>true</property>
       </properties>
      </connector>
     </connectors>
    </window-calculate>
```

The segmented correlation algorithm is governed by the following properties:

Property	Value Type	Property Type	Default Value	Description
Parameters				

Property	Value Type	Property Type	Default Value	Description
`sampleSize`	int64	Optional	1,000	Specifies the number of samples to run the correlation. If `indicator` is not specified, then the input data stream is divided into segments of size `sampleSize`, and the correlation is performed on adjacent segments. If `indicator` is specified, then the input data stream is first divided into segments on events when `indicator=1`. If `sampleSize` is smaller than 1, then correlation is performed on all adjacent segments. Otherwise, segments are further divided into subsegments of size `sampleSize` (the size of the last subsegment can be smaller than `sampleSize`). Then correlation is computed on the first set of subsegments in adjacent segments, the second set of subsegments, and so on.
`minSize`	int64	Optional	0	Specifies the lower bound of the number of samples to run the correlation.
`maxSize`	int64	Optional	INT64_MAX	Specifies the upper bound of the number of samples to run the correlation. **Note:** The default value is `INT64_MAX=9223372036854775807`.
Input Mapping				
`x`	variable	Required	NA	Specifies the input variable for the segmented correlation.
`indicator`	variable	Optional	`""` (empty string)	Specifies the input variable for the segment indicator. The variable value should be 1 when an old segment ends and a new segment begins and 0 otherwise.
Output Mapping				
`corOut`	variable	Optional	`""` (empty string)	Specifies the output variable name for the segmented correlation.

The calculated segmented correlation output variables are published through a file-and-socket adapter to a CSV file named **`result.out`**.

The edges are defined at the end of the project.

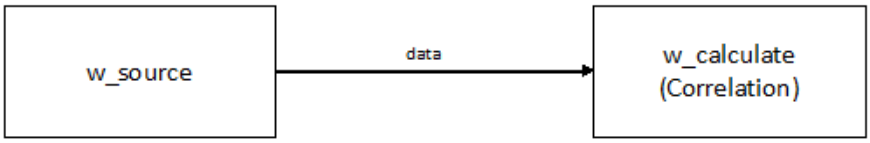
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the default values of the segmented correlation algorithm parameter properties for the calculate window with the streaming analytics command-line utility.

Calculating Streaming Pearson Correlation

The most common measure of how sets of data correlate with one another is the *Pearson correlation coefficient*. It shows the linear relationship between two sets of data.

Consider the following example:



This continuous query includes the following:

- a source window that receives the data to be analyzed
- a calculate window that calculates the correlation between two variables from an incoming data stream and publishes the results in real time

The source window `w_source` receives a data event that consists of the contents of a file named `input.csv`. It receives the event through a file-and-socket connector. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; and a y coordinate of data named `y_c`.

```
<window-source name='w_source' insert-only='true' autogen-key='false'>
     <schema>
      <fields>
       <field name='id'  type='int64' key='true'/>
       <field name='x_c' type='double'/>
       <field name='y_c' type='double'/>
      </fields>
     </schema>
     <connectors>
      <connector class='fs' name='publisher'>
       <properties>
        <property name='type'>pub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>input/input.csv</property>
        <property name='transactional'>true</property>
        <property name='blocksize'>1</property>
       </properties>
      </connector>
     </connectors>
    </window-source>
```

The calculate window `w_calculate` receives data events including the values of two variables. It publishes their calculated correlation according to the correlation algorithm properties that are specified.

```
<window-calculate name="w_calculate" algorithm="Correlation">
      <schema>
          <fields>
          <field name='id' type='int64' key='true'/>
          <field name='y_c' type='double'/>
          <field name='x_c' type='double'/>
          <field name='corOut' type='double'/>
        </fields>
      </schema>
        <parameters>
          <properties>
           <property name="windowLength">5</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
```

```
          <property name="x">x_c</property>
          <property name="y">x_c</property>
         </properties>
        </input-map>
        <output-map>
         <properties>
          <property name="corOut">corOut</property>
         </properties>
        </output-map>
      <connectors>
       <connector class='fs' name='sub'>
        <properties>
         <property name='type'>sub</property>
         <property name='fstype'>csv</property>
         <property name='fsname'>TEST_OUTPUT/output/result.out</property>
         <property name='snapshot'>true</property>
        </properties>
       </connector>
      </connectors>
     </window-calculate>
```

The following properties govern the correlation algorithm in the calculate window:

Property	Value Type	Property Type	Default Value	Description
Parameters				
`windowLength`	int64	Optional	`3`	Specifies the length of the sliding window. The default value is 3.
Input Mapping				
`x`	variable	Required	No default value	Specifies the input variable X by its name in the source schema.
`y`	variable	Required	No default value	Specifies the input variable Y by its name in the source schema.
Output Mapping				
`corOut`	variable	Required	`""` (empty string)	Specifies the name of the output variable for the correlation between X and Y input variables.

The calculated correlation output variables are published through a file-and-socket adapter to a CSV file named **`result.out`**.

The edges are defined at the end of the project.

```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the default values of the streaming correlation algorithm parameter properties for the calculate window with the streaming analytics command-line utility.

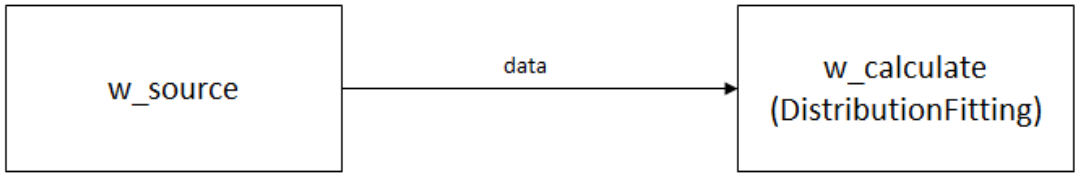
Streaming Distribution Fitting

The calculate window can fit a probability distribution to streaming data with the distribution fitting algorithm.

The distribution fitting algorithm fits the Weibull distribution to a variable in the incoming data stream, which is described by the following probability density function:

$$f\left(x;\beta,\alpha,\mu\right)=\frac{\beta}{\alpha}\left(\frac{x-\mu}{\alpha}\right)^{\beta-1}\exp\left(-\left(\frac{x-\mu}{\alpha}\right)^{\beta}\right),\ x\geq\mu$$

Consider the following example:



This continuous query includes the following:

- a source window that receives the data to be analyzed
- a calculate window that fits the Weibull distribution to a variable from an incoming data stream and publishes the variable's functional parameters as results

The source window `w_source` receives a data event that consists of the contents of a file named `input.csv`. It receives the event through a file-and-socket connector. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; and a y coordinate of data named `y_c`.

```
<window-source name='w_source' insert-only='true' autogen-key='false'>
     <schema>
      <fields>
       <field name='id'  type='int64' key='true'/>
       <field name='x_c' type='double'/>
       <field name='y_c' type='double'/>
      </fields>
     </schema>
     <connectors>
      <connector class='fs' name='publisher'>
       <properties>
        <property name='type'>pub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>input/input.csv</property>
        <property name='transactional'>true</property>
        <property name='blocksize'>1</property>
       </properties>
      </connector>
     </connectors>
    </window-source>
```

The calculate window `w_calculate` receives data events. It publishes the β, α, and μ parameters of the Weibull probability density function for the specified `x` variable.

```
<window-calculate name="w_calculate" algorithm="DistributionFitting">
      <schema>
          <fields>
          <field name='id' type='int64' key='true'/>
          <field name='y_c' type='double'/>
          <field name='x_c' type='double'/>
          <field name='betaOut' type='double'/>
```

```
        <field name='alphaOut' type='double'/>
        <field name='muOut' type='double'/>
        <field name='convergeOut' type='int64'/>
      </fields>
    </schema>
      <parameters>
        <properties>
         <property name="windowLength">5</property>
         <property name="overlap">2</property>
         <property name="maxIter">50</property>
         <property name="distribution">Weibull</property>
        </properties>
      </parameters>
      <input-map>
        <properties>
         <property name="x">x_c</property>
        </properties>
      </input-map>
      <output-map>
        <properties>
         <property name="betaOut">betaOut</property>
         <property name="alphaOut">alphaOut</property>
         <property name="muOut">muOut</property>
        </properties>
      </output-map>
   <connectors>
    <connector class='fs' name='sub'>
     <properties>
      <property name='type'>sub</property>
      <property name='fstype'>csv</property>
      <property name='fsname'>TEST_OUTPUT/output/result.out</property>
      <property name='snapshot'>true</property>
      <property name='convergeOut'>convergeOut</property>
     </properties>
    </connector>
   </connectors>
  </window-calculate>
```

The distribution fitting algorithm is governed by the following properties:

Property	Value Type	Property Type	Default Value	Description
Parameters				
`windowLength`	int64	Optional	`3`	Specifies the length of the sliding window.
`overlap`	int64	Optional	`1`	Specifies the overlap between consecutive windows. Must be strictly less than `windowLength`.
`maxIter`	int32	Optional	`100`	Specifies the maximum number of iterations.
`distribution`	varchar	Optional	`Weibull`	Specifies the type of probability distribution to fit.
Input Mapping				
`x`	variable	Required	No default value	Specifies the input variable for distribution fitting.

Property	Value Type	Property Type	Default Value	Description
Output Mapping				
`betaOut`	variable	Optional	" " (empty string)	Specifies the output variable name for parameter β.
`alphaOut`	variable	Optional	" " (empty string)	Specifies the output variable name for parameter α.
`muOut`	variable	Optional	" " (empty string)	Specifies the output variable name for parameter μ.
`convergeOut`	variable	Optional	" " (empty string)	Specifies the output variable name that indicates whether convergence is attained. Set the value to 1 when computation is converged and 0 otherwise.

The calculated parameter output variables are published through a file-and-socket adapter to a CSV file named **`result.out`**.

The edges are defined at the end of the project.

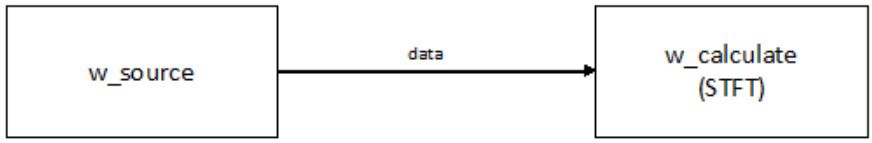
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the default values of the streaming distribution fitting algorithm parameter properties for the calculate window with the streaming analytics command-line utility.

Calculating Short-Time Fourier Transforms

A *Fourier transform* decomposes a function of time into its underlying frequencies. The amplitude, offset, and rotation speed of every underlying cycle is returned by the function.

Consider the following example:



This continuous query includes the following:

- a source window that receives the data to be analyzed
- a calculate window that calculates short–time Fourier transforms (STFTs) on incoming data events and publishes the results

The source window `w_source` receives input data, a file named `input.csv`, through a file-and-socket connector. The input stream is placed into two fields for each observation: an ID that acts as the data stream's key, named `id`, and a y coordinate of data named `y`.

```
<window-source name='w_source' insert-only='true'>
     <schema>
      <fields>
       <field name='ID'  type='int64' key='true'/>
       <field name='y' type='double'/>
```

```
      </fields>
     </schema>
     <connectors>
      <connector class='fs' name='publisher'>
       <properties>
        <property name='type'>pub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>input/input.csv</property>
        <property name='transactional'>true</property>
        <property name='blocksize'>1</property>
       </properties>
      </connector>
     </connectors>
    </window-source>
```

The calculate window `w_calculate` receives data events and publishes calculated transforms according to the STFT algorithm properties that are specified.

```
<window-calculate name='w_calculate' algorithm='STFT'>
      <schema>
          <fields>
          <field name='ID' type='int64' key='true'/>
          <field name='time' type='int64'/>
          <field name='bin' type='int64'/>
          <field name='power' type='double'/>
          <field name='phase' type='double'/>
        </fields>
      </schema>
        <parameters>
          <properties>
           <property name='windowLength'>4096</property>
           <property name='windowType'>15</property>
           <property name='windowParam'>-1.5</property>
           <property name='fftLength'>4096</property>
           <property name='binsInSchema'>2048</property>
           <property name='overlap'>3072</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
           <property name='input'>y</property>
           <property name='timeId'>ID</property>
          </properties>
        </input-map>
        <output-map>
          <properties>
           <property name='keyOut'>ID</property>
           <property name='timeIdOut'>time</property>
           <property name='binOut'>bin</property>
           <property name='powerOut'>power</property>
           <property name='phaseOut'>phase</property>
          </properties>
        </output-map>
      <connectors>
       <connector class='fs' name='sub'>
        <properties>
         <property name='type'>sub</property>
```

```
      <property name='fstype'>csv</property>
      <property name='fsname'>TEST_OUTPUT/output/result.out</property>
      <property name='snapshot'>true</property>
     </properties>
    </connector>
   </connectors>
  </window-calculate>
```

The following properties govern the STFT algorithm in the calculate window:

Property	Value Type	Property Type	Default Value	Description
Parameters				
windowLength	int64	Optional	128	Specifies the length of the sliding window.
windowType	int64	Optional	15	Specify one of the following window types:: 1=Bartlett, 2=Bohman, 3=Chebyshev, 4=Gaussian, 5=Kaiser, 6=Parzen, 7=Rectangular, 10=Tukey, 11=Bartlett-Hann, 12=Blackman-Harris, 13=Blackman, 14=Hamming, 15=Hanning, and 16=Flat Top.
windowParam	double	Optional	-1.0	Specifies the parameters for windowType. If not required for the window type selected, this value is ignored.
fftLength	int64	Optional	128	Specifies the length to which windowed data should be expanded. Zeros are appended to the data before the Fast Fourier Transform (FFT) is performed.The specified value must be positive and at least as large as windowLength. A power of two is suggested to maximize computational efficiency.
binsInSchema	int64	Optional	64	Specifies the number of frequency bins to output. Must be less than or equal to fftLength. For real signals, bins greater than (fftLength/2) are not physically meaningful.
overlap	int64	Optional	127	Specifies the overlap between consecutive windows. Must be strictly less than windowLength.
Input Mapping				
input	variable	Required	No default value	Specifies the input variable by its name in the source schema. The calculate window analyzes this variable.
timeId	variable	Required	No default value	Specifies the time ID variable name in the input stream. This variable should be of type int64.
Output Mapping				
keyOut	variable	Required	No default value	Specifies the key variable name (unique for each output event) in the output stream. This variable should be of type int64.
timeIdOut	variable	Required	No default value	Specifies the time ID variable name in the output stream. This variable should be of type int64. There is more than one output event for a given time ID.

Property	Value Type	Property Type	Default Value	Description
`binOut`	variable	Required	No default value	Specifies the frequency bin variable name in the output stream. This variable should be of type `int64`.
`powerOut`	variable	Optional	`""` (empty string)	Specifies the power variable name in the output stream.
`phaseOut`	variable	Optional	`""` (empty string)	Specifies the phase variable name in the output stream.

The calculated STFT output data is organized by event fields that are specified in the schema of the calculate window. The output variables are published through a file-and-socket adapter to a CSV file named **`result.out`**.

The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

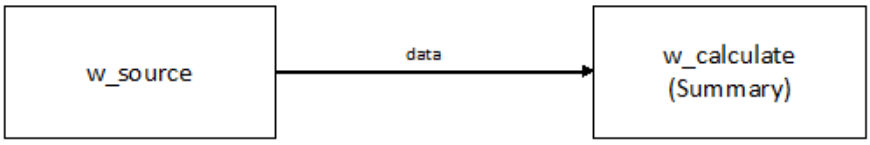
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the parameters and the input and output mapping properties required to set up a streaming STFT project with the streaming analytics command-lineuUtility.

Calculating Streaming Summary Statistics for One Variable

SAS Event Stream Processing Analytics includes univariate summary statistics as an algorithm for the calculate window.

Consider the following example:



This continuous query includes the following:

- a source window that receives the data to be analyzed
- a calculate window that calculates summary statistics on incoming data events and publishes the results

The source window `w_source` receives input data, a file named `input.csv`, through a file-and-socket connector. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; and a y coordinate of data named `y_c`.

```
<window-source name='w_source' insert-only='true' autogen-key='false'>
     <schema>
      <fields>
       <field name='id'  type='int64' key='true'/>
       <field name='x_c' type='double'/>
       <field name='y_c' type='double'/>
      </fields>
     </schema>
```

```
    <connectors>
     <connector class='fs' name='publisher'>
      <properties>
       <property name='type'>pub</property>
       <property name='fstype'>csv</property>
       <property name='fsname'>input/input.csv</property>
       <property name='transactional'>true</property>
       <property name='blocksize'>1</property>
      </properties>
     </connector>
    </connectors>
   </window-source>
```

The calculate window `w_calculate` receives data events and publishes calculated summary statistics according to the summary algorithm properties that are specified.

```
<window-calculate name='w_calculate' algorithm='Summary'>
      <schema>
          <fields>
          <field name='id' type='int64' key='true'/>
          <field name='y_c' type='double'/>
          <field name='x_c' type='double'/>
          <field name='nOut' type='double'/>
          <field name='nmissOut' type='double'/>
          <field name='minOut' type='double'/>
          <field name='maxOut' type='double'/>
          <field name='sumOut' type='double'/>
          <field name='meanOut' type='double'/>
          <field name='stdOut' type='double'/>
          <field name='varOut' type='double'/>
          <field name='cssOut' type='double'/>
          <field name='ussOut' type='double'/>
          <field name='stderrOut' type='double'/>
          <field name='cvOut' type='double'/>
        </fields>
      </schema>
        <parameters>
          <properties>
           <property name='windowLength'>5</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
           <property name='input'>x_c</property>
          </properties>
        </input-map>
        <output-map>
          <properties>
           <property name='nOut'>nOut</property>
           <property name='nmissOut'>nmissOut</property>
           <property name='minOut'>minOut</property>
           <property name='maxOut'>maxOut</property>
           <property name='sumOut'>sumOut</property>
           <property name='meanOut'>meanOut</property>
           <property name='stdOut'>stdOut</property>
           <property name='varOut'>varOut</property>
           <property name='cssOut'>cssOut</property>
```

```
        <property name='ussOut'>ussOut</property>
        <property name='stderrOut'>stderrOut</property>
        <property name='cvOut'>cvOut</property>
       </properties>
     </output-map>
   <connectors>
    <connector class='fs' name='sub'>
     <properties>
      <property name='type'>sub</property>
      <property name='fstype'>csv</property>
      <property name='fsname'>TEST_OUTPUT/output/result.out</property>
      <property name='snapshot'>true</property>
     </properties>
    </connector>
   </connectors>
  </window-calculate>
```

The following properties govern the summary algorithm in the calculate window:

Property	Value Type	Property Type	Default Value	Description
Parameters				
windowLength	int64	Optional	3	Specifies the length of the sliding window. The default value is 3.
Input Mapping				
input	variable	Required	No default value	Specifies the input variable by its name in the source schema. The univariate summary statistics are calculated for this variable.
Output Mapping				
nOut	variable	Optional	" " (empty string)	Specifies the output variable name for the number of observations analyzed for the incoming data events (N).
nmissOut	variable	Optional	" " (empty string)	Specifies the output variable name for the number of missing values in the incoming data events (NMISS).
minOut	variable	Optional	" " (empty string)	Specifies the output variable name for the minimum observed value (MIN).
maxOut	variable	Optional	" " (empty string)	Specifies the output variable name for the maximum value (MAX).
sumOut	variable	Optional	" " (empty string)	Specifies the output variable name for the linear sum (SUM).
meanOut	variable	Optional	" " (empty string)	Specifies the output variable name for the mean (MEAN).
stdOut	variable	Optional	" " (empty string)	Specifies the output variable name for the standard deviation (STD).
varOut	variable	Optional	" " (empty string)	Specifies the output variable name for the sample variance (VAR).

Property	Value Type	Property Type	Default Value	Description
`cssOut`	variable	Optional	" " (empty string)	Specifies the output variable name for the corrected sum of squares (CSS).
`ussOut`	variable	Optional	" " (empty string)	Specifies the output variable name for the uncorrected sum of squares (USS).
`stderrOut`	variable	Optional	" " (empty string)	Specifies the output variable name for the standard error (STDERR).
`cvOut`	variable	Optional	" " (empty string)	Specifies the output variable name for the coefficient of variation (CV).

The calculated summary statistics are organized into event fields that are specified in the schema of the calculate window. The events are published as results through a file-and-socket adapter to a CSV file named **`result.out`**.

The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

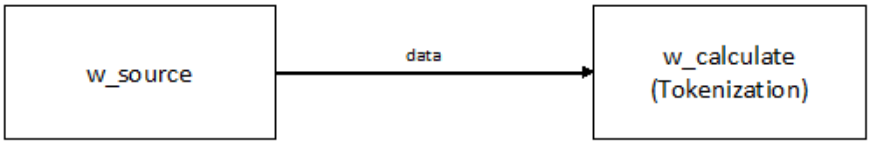
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the default values of the summary statistics algorithm parameter properties for the calculate window with the streaming analytics command-line utility.

Streaming Text Tokenization

The calculate window supports text tokenization through a tokenization algorithm.

Consider the following example:



This continuous query includes the following:

- a source window that receives the text data to be analyzed
- a calculate window that tokenizes text in incoming data events and publishes the results

The source window `w_source` receives input data, a file named `input.csv`, through a file-and-socket connector. The input stream is placed into two fields for each observation: a document ID that acts as the data stream's key, named `docId`, and a string of incoming text, named `doc`.

```
<window-source name='w_source' insert-only='true'>
     <schema>
      <fields>
       <field name='docId'  type='int64' key='true'/>
       <field name='doc' type='string'/>
      </fields>
     </schema>
```

```
    <connectors>
     <connector class='fs' name='publisher'>
      <properties>
       <property name='type'>pub</property>
       <property name='fstype'>csv</property>
       <property name='fsname'>input/input.csv</property>
       <property name='transactional'>true</property>
       <property name='blocksize'>1</property>
      </properties>
     </connector>
    </connectors>
   </window-source>
```

The calculate window `w_calculate` receives data events and publishes word tokens created with the tokenization algorithm.

```
<window-calculate name='w_calculate' algorithm='Tokenization'>
      <schema>
        <fields>
          <field name='docId' type='int64' key='true'/>
          <field name='tokenId' type='int64' key='true'/>
          <field name='word' type='string'/>
          <field name='startPos' type='int32'/>
          <field name='endPos' type='int32'/>
        </fields>
      </schema>
         <parameters>
          <properties>
            <property name="doc">doc</property>
            <property name="docId">docId</property>
            <property name="docIdOut">docId</property>
            <property name="tokenIdOut">tokenId</property>
            <property name="wordOut">word</property>
            <property name="startPosOut">startPos</property>
            <property name="endPosOut">endPos</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
           <property name='docId'>docId</property>
           <property name='doc'>doc</property>
          </properties>
        </input-map>
        <output-map>
          <properties>
           <property name='docIdOut'>docId</property>
           <property name='tokenIdOut'>tokenId</property>
           <property name='wordOut'>word</property>
           <property name='startPosOut'>startPos</property>
           <property name='endPosOut'>endPos</property>
          </properties>
        </output-map>
     <connectors>
      <connector class='fs' name='sub'>
       <properties>
        <property name='type'>sub</property>
        <property name='fstype'>csv</property>
```

```
      <property name='fsname'>TEST_OUTPUT/output/result.out</property>
      <property name='snapshot'>true</property>
     </properties>
    </connector>
   </connectors>
  </window-calculate>
```

The following properties govern the tokenization algorithm in the calculate window:

Property	Value Type	Property Type	Default Value	Description
Input Mapping				
docId	variable	Required	No default value	Specifies the input variable for the input doc from the source window.
doc	variable	Required	No default value	Specifies the input variable for the unique doc ID.
Output Mapping				
docIdOut	variable	Required	No default value	Specifies the output variable for the unique doc ID.
tokenIdOut	variable	Required	No default value	Specifies the output variable for the unique ID of the token.
wordOut	variable	Required	No default value	Specifies the output variable for the word content in the token.
startPosOut	variable	Optional	" " (empty string)	Specifies the output variable for the starting position of the token word.
endPosOut	variable	Optional	" " (empty string)	Specifies the output variable for the ending position of the token word.

The calculated tokens are organized by the event fields that are specified in the schema of the calculate window. The tokens are published through a file-and-socket adapter to a CSV file named **result.out**.

The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the default values of the streaming text tokenization algorithm parameter properties for the calculate window with the streaming analytics command-line utility.

Offline Streaming Analytics Projects

Overview

Offline streaming analytics projects enable you to use algorithms and models that are not provided with SAS Event Stream Processing. These algorithms and models are brought into SAS Event Stream Processing as analytic store (ASTORE) files.

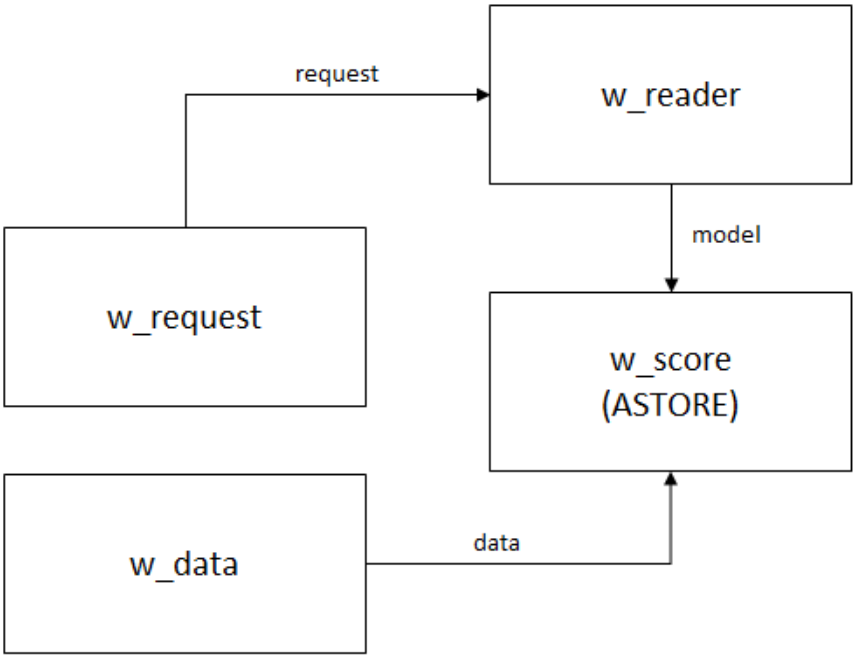
An *ASTORE file* is a binary file that contains a model's state after it completes the training phase of data analysis. For more information about the advanced analytics available to SAS Event Streaming Processing through ASTORE files, see *SAS Visual Data Mining and Machine Learning: Data Mining and Machine Learning Procedures*.

The following algorithms are supported:

Algorithm	Description
Support Vector Data Description	Describes a data set. A good description covers all target data but includes no superfluous space. The boundary of a data set can be used to detect novel data or outliers. The Support Vector Data Description obtains a spherically shaped boundary around a data set.
Random Forest	Builds decision trees at training time. It writes the class of trees that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees.
Gradient Boosting Tree	Produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion and generalizes it by optimizing an arbitrary differentiable loss function.
Support Vector Machine	Provides a discriminative classifier formally defined by a separating hyperplane. Given labeled training data (supervised learning), the algorithm writes an optimal hyperplane that categorizes new examples.
Factorization Machine	Provides a general predictor similar to support vector machines. It can estimate reliable parameters under very high sparsity.

Example: ASTORE with Random Forest

Consider the following example:



Two source windows are specified: one to read the data to be scored (`w_data`), as follows:

```
<window-source name='w_data'>
          <schema>
            <fields>
              <field name='id' type='int64' key='true'/>
              <field name='SepalLength' type='double'/>
              <field name='SepalWidth'  type='double'/>
              <field name='PetalLength' type='double'/>
              <field name='PetalWidth'  type='double'/>
              <field name='Species'     type='string'/>
            </fields>
          </schema>
          <connectors>
            <connector class='fs' name='publisher'>
              <properties>
                <property name='type'>pub</property>
                <property name='fstype'>csv</property>
                <property name='fsname'>input/iris_esp.csv</property>
                <property name='transactional'>true</property>
                <property name='blocksize'>1</property>
                <property name='rate'>30</property>
              </properties>
            </connector>
          </connectors>
        </window-source>
```

One to read requests (`w_request`):

```
<window-source name='w_request' insert-only='true' index='pi_EMPTY'>
          <schema>
            <fields>
              <field name='req_id'  type='int64' key='true'/>
```

```
      <field name='req_key' type='string'/>
      <field name='req_val' type='string'/>
    </fields>
  </schema>
  <connectors>
    <connector class='fs' name='publisher'>
      <properties>
        <property name='type'>pub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>input/reader_request.csv</property>
        <property name='transactional'>true</property>
        <property name='blocksize'>1</property>
      </properties>
    </connector>
  </connectors>
</window-source>
```

A model reader window (`w_reader`) receives requests from `w_request`, creates a model using the request information, and publishes the model event to the score window (`w_score`) for scoring.

```
<window-model-reader name='w_reader'/>
```

The score window `w_score` scores the incoming streaming events according to the model events that it receives from `w_reader` and the analytic store information from an ASTORE file. The type of offline model is specified as `astore`, and the name of the ASTORE file is referenced (`forest_iris_sasast`.

```
<window-score name='w_score'>
  <schema>
    <fields>
      <field name='id'           type='int64' key='true'/>
      <field name='SepalLength'  type='double'/>
      <field name='SepalWidth'   type='double'/>
      <field name='PetalLength'  type='double'/>
      <field name='PetalWidth'   type='double'/>
      <field name='Species'      type='string'/>
      <field name='P_SpeciesVersicolor' type='double'/>
      <field name='P_SpeciesVirginica'  type='double'/>
      <field name='P_SpeciesSetosa'     type='double'/>
      <field name='I_Species'           type='string'/>
    </fields>
  </schema>
  <models>
    <offline type='astore' reference='forest_iris_sasast'>
      <output-map>
        <properties>
          <property name='test'>testabc</property>
        </properties>
      </output-map>
    </offline>
  </models>
  <connectors>
    <connector class='fs' name='sub'>
      <properties>
        <property name='type'>sub</property>
        <property name='fstype'>csv</property>
        <property name='fsname'>TEST_OUTPUT/output/result.out</property>
        <property name='snapshot'>true</property>
      </properties>
```

```
            </connector>
          </connectors>
        </window-score>
```

The scored events are organized by event fields that are specified in the schema of the window. The output variables are published through a file-and-socket adapter to a CSV file named **result.out**.

The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<edges>
        <edge source='w_data'    target='w_score'  role='data'/>
        <edge source='w_reader'  target='w_score'  role='model'/>
        <edge source='w_request' target='w_reader' role='request'/>
      </edges>
```