SAS® Event Stream Processing 5.1: Using SAS® Event Stream Processing Analytics

Overview

SAS Event Stream Processing Analytics 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:

Window Type	Description
Score	Scores events with online algorithms that are packaged with SAS Event Stream Processing or with algorithms and models brought in as analytic store (ASTORE) files.
Train	Generates scoring models based on streaming event data.
Calculate	Transforms data events using a variety of algorithms.
Model Reader	Reads models brought into SAS Event Stream Processing as analytic store (ASTORE) files.
Model Supervisor	Manages models received from Model Reader windows.

Note: SAS Event Stream Processing Analytics is not supported on Microsoft Windows.

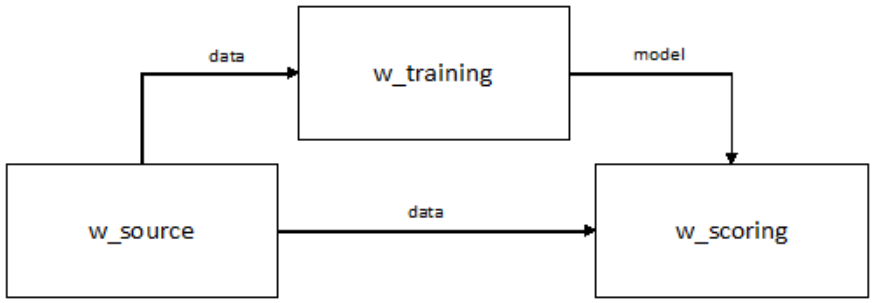
When you execute an algorithm, you must specify its name, parameter definitions, and input and output mapping properties. To obtain this information for a particular window type and algorithm combination:

- use the **`dfesp_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	Description
data	Sends data events between windows. A data event streams data to be processed by an analytical algorithm into the receiving window.
model	Sends model events between windows. A model event, which has a fixed schema, streams model details into the receiving window.
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 many training models:

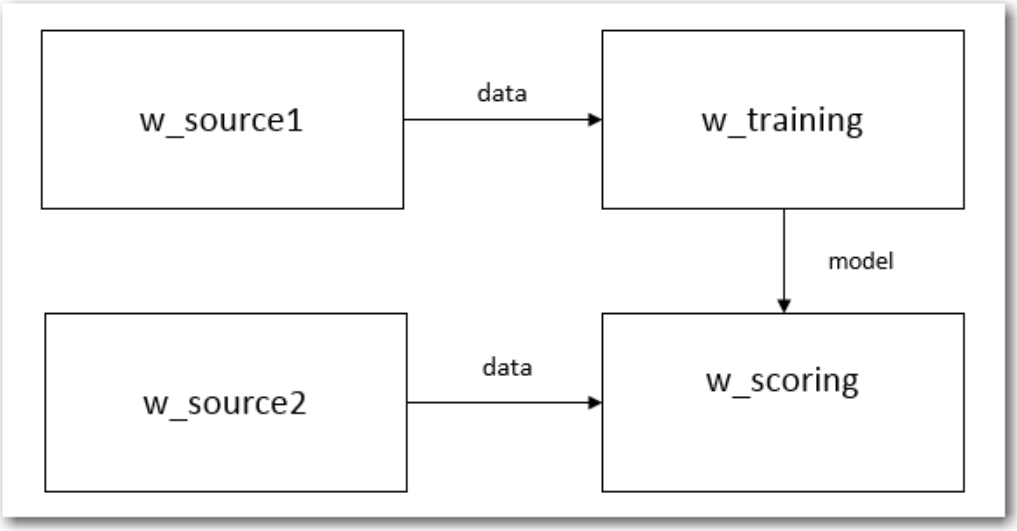


The following code specifies roles for the edges between the windows:

```
<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.
- 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.

Now consider an alternative arrangement of windows.



Here, one Source window streams data for training and a different Source window streams data for scoring. The following code specifies roles for the edges between the windows:

```
<edges>
    <edge source='w_source1' target='w_training' role='data'/>
    <edge source='w_source2' target='w_scoring' role='data'/>
    <edge source='w_training' target='w_scoring' role='model'/>
</edges>
```

Determining Algorithm Availability and Properties

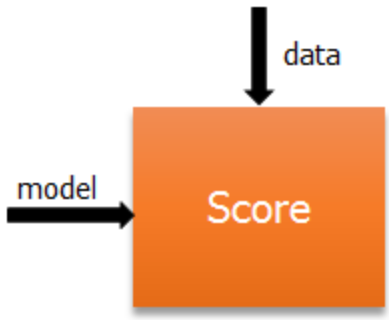
To determine what algorithms are available to a window and what properties you must set to use the algorithms, use **`dfesp_analytics`**, a UNIX command-line utility that is located in **`$DFESP_HOME/bin`**.

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.

Window Types

Score Window

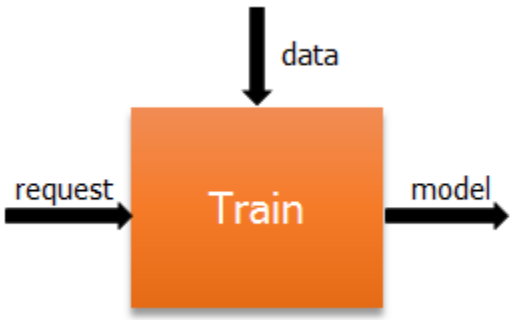
A *Score window* accepts model events to make predictions for incoming data events. It generates scored data. You can use this score data to generate predictions based on the trained model. (No role is assigned to the outgoing edges, so they do not appear in the diagram.)



Train Window

A *Train window* receives data events and publishes model events into a Score window. It uses the incoming data to develop and adjust a learning model in real time. Often, the data is historical data from which to learn patterns. Incoming data should contain both the outcome that you are trying to predict and related variables.

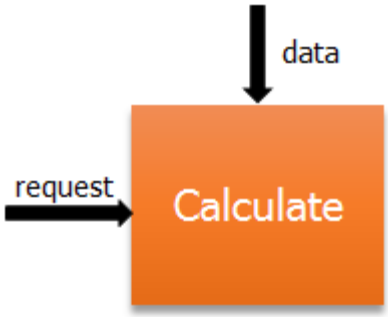
The Train window can also receive request events . These events can adjust the learning algorithm while events continue to stream.



After the Train window has adjusted an algorithm, it writes the adjusted model to a Score window or a Model Supervisor window through a model event.

Calculate Window

The *Calculate window* creates real time, running statistics that are based on established analytical techniques. It receives data events and publishes newly transformed 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.



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

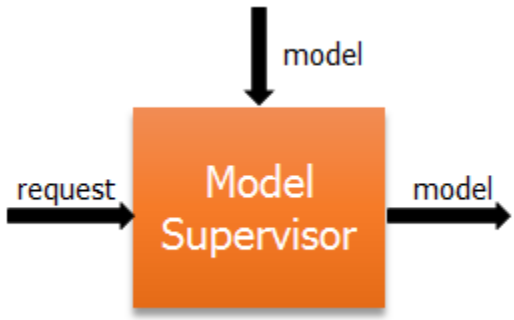
Model Reader Window

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 that contains the ASTORE model to a Score window or a Model Supervisor window.

Model reader models are referred to as “offline,” because the ASTORE models are developed and updated offline.

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 a Score 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 window’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.

Event Types

Data Events

Data events stream input data to be processed by a receiving window. Data that feeds into a Score window is processed with machine learning algorithms specified by the incoming model event. The Score window produces data events consisting of scored data. Data that feeds into a Train window is applied to train the model specified by the incoming model event to produce a trained model. The data that feeds into a Calculate window serves as input into analytical techniques to produce real–time running statistics.

Model Events

Model events stream model metadata into a Score window or into a Model Supervisor window. You do not explicitly specify model event schema; they are implicit for the query.

Internally, model events have a fixed schema, as follows:

Field	Description
`model_id`	Specifies a unique ID assigned by a Train window, Model Reader window, or a Model Supervisor window.
`model_addr`	Specifies the address of the model descriptor in memory.
`model_origin`	Specifies the window name from which the model was first created.
`model_token`	Specifies a token to determine the receiver of the 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

Overview

Events that are transferred through request edges are called *request events*. You can use request events to initiate an action (for example, reconfigure a model). 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. Request events are generally slower than data events.

Specify the schema of a request event in the originating window. Request events have a fixed schema that consists of a `req_id` field, a `req_key` field, and a `req_val` field.

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

Request events can perform the following actions:

Action	Window	Description
`list`	Model Supervisor	Lists all available models currently stored.
`send`	Model Supervisor	Sends a model to a downstream window.
`remove`	Model Supervisor	Removes a model.
`load`	Source Model Reader	Loads an offline ASTORE file into the event stream processing engine.
`reconfig`	Calculate, Train	Changes the value of a property.

The first request event should always specify **`action`** as the `req_key`. It should set the desired action (**`list`**, **`send`**, and so on) of the request as the `req_val`.

The last request event should always specify an empty `req_key`, which indicates the end of the request.

A request consists of a series of request events. The parameters of a request's action are specified in the request events between the first and last. In the middle request events, the values of `req_key` specify the names of parameters, and the values of `req_val` specify their values.

List Requests

Here is an example of a **`list`** request:

```
i,n,1,"action","list"
i,n,2,,
```

All events are Insert (Normal). After receiving the request, the Model Supervisor window sends multiple model events. Each model event represents an available model. The `model_addr` field of those model events are masked (that is, set to -1). Downstream windows (for example, Score windows) ignore these events.

After returning all available models, the model supervisor sends an event with `model_id` set to -1 to indicate the end of results. Thus, to get a list of available models, subscribe to the Model Supervisor window and capture the `model_id` fields.

Send Requests

The structure of a **`send`** request is as follows:

```
i,n,1,"action","send"
i,n,2,"modelId","USER_SPECIFIED_MODEL_ID"
```

```
i,n,3,"target","USER_SPECIFIED_TARGET"
i,n,4,,
```

All events are Insert (Normal). The *USER_SPECIFIED_MODEL_ID* is one previously obtained through a **`list`** request. The *USER_SPECIFIED_TARGET* can be the name of any window downstream of the Model Supervisor window.

Remove Requests

The structure of a **`remove`** request is as follows:

```
i,n,1,"action","remove"
i,n,2,"modelId","USER_SPECIFIED_MODEL_ID"
i,n,3,,
```

All events are Insert (Normal). The *USER_SPECIFIED_MODEL_ID* is one previously obtained through a **`list`** request. When you request to remove a model that has already been deployed to one window, the model is still valid and can continue to be used. However, the model can no longer be deployed to a new window. The model is removed permanently after no window uses it.

Load Requests

Send a **`load`** request to instruct a Model Reader window to load an ASTORE file into an engine. All events are Insert (Normal).

```
i,n,1,"action","load"
i,n,2,"type","astore"
i,n,3,"reference","YOUR_ASTORE_FILE"
i,n,4,
```

Reconfig Requests

Here is an example of a **`reconfig`** request:

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

Callbacks that handle requests (for example, **`reconfig`** in the Calculate and Train windows and **`read`** in the Model Reader window) are not invoked by each request event. They are invoked by each request.

Example

The following model 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,,
```

Online Projects

Overview

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

Note: When you persist a model that trains online projects, note that the Train window immediately starts training a new model upon model restore. No user intervention is required. For more information, see "Persist and Restore Operations" in SAS Event Stream Processing: Creating and Using Windows .

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

Table A.1 *Algorithms Packaged with SAS Event Stream Processing Analytics*

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

Train and Score Windows
Streaming Linear Regression
Streaming Logistic Regression
Streaming Support Vector Machines

Calculate Window
Streaming Summary (Univariate Statistics)
Streaming Pearson's Correlation
Segmented Correlation
Streaming Distribution Fitting
Short-Time Fourier Transform
Compute Fit Statistics
Compute ROC Information
Calculate a Streaming Histogram
Streaming Text Tokenization
Streaming Text Vectorization
Image Processing Algorithm
Moving Relative Range

Calculate windows analyze values in input events according to the specified algorithm and publish analysis results to output variables in output events. In most cases, for each event that streams into a Calculate window, there is a single corresponding output event that contains those results. There are three exceptions:

Algorithm	Number of Output Events per Input Event
Text Tokenizer	One for each word token produced (`wordOut`).
Short Time Fourier Transformation	The specified number of frequency bins (`binsInSchema`).
Compute ROC Information	The result of dividing 1 by the bin width (`cutStep`). The default value of `cutStep` is 0.01. Thus, by default you have 100 output events per input event.

Calculate windows that use algorithms with sliding windows do not produce output events until the total number of input events is equal to the length of the sliding window.

Training and Scoring with K-means Clustering

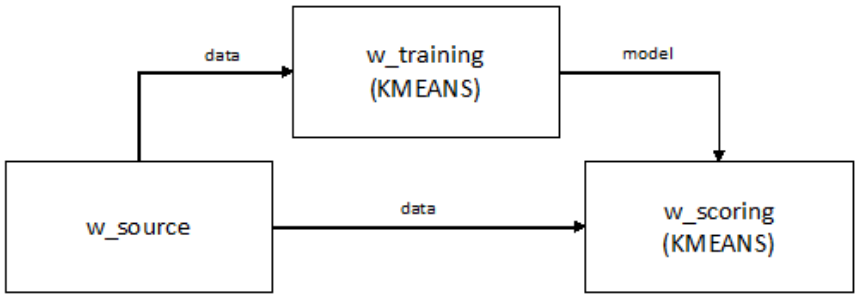
The classic k-means clustering algorithm performs two basic steps:

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:

Table A.2 *Parameters*

Property	Value Type	Property Type	Default Value	Description
`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.

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.3 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
`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:

Table A.4 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.5 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`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>
```

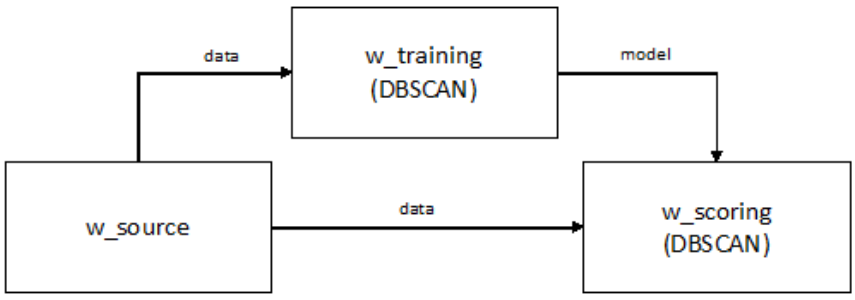
You can view the default values of the k-means algorithm parameter properties for the Score and Train windows with the command-line utility.

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:

Table A.6 Parameters

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.7 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`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:

Table A.8 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.9 *Output Mapping*

Property	Value Type	Property Type	Default Value	Description
`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.

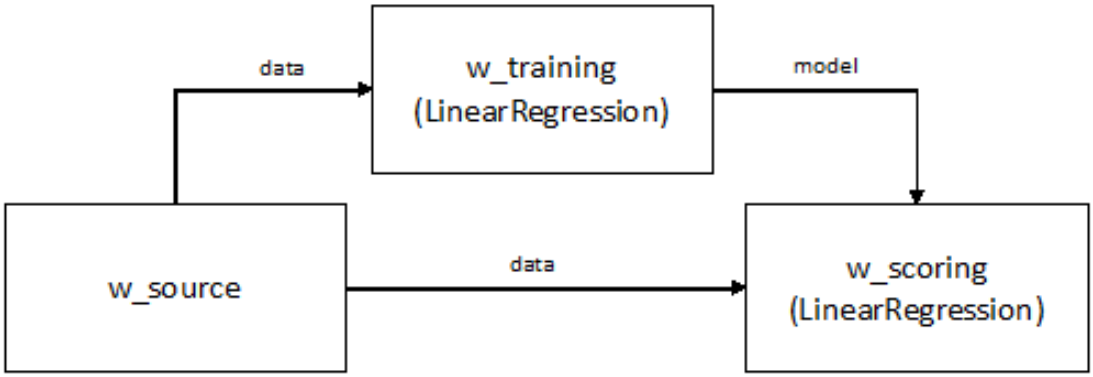
```
  <edges>
  <edge source='w_source' target='w_training' role='data'/>
  <edge source='w_source' target='w_scoring' role='data'/>
  <edge source='w_training' target='w_scoring' role='model'/>
</edges>
```

You can view the default values of the DBSCAN algorithm parameter properties for the Score and Train windows with the command-line utility.

Training and Scoring with Linear Regression

Linear regression models the relationship between a scalar dependent variable and one or more explanatory variables (or independent variables).

Consider the following example:



This continuous query includes the following:

- a Source window that receives data events that stream the data to score
- a Train window that generates and periodically updates the linear regression model
- a Score window that performs the scoring

The Source window `w_source` receives a data event from a file named `mnist_all.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`; a y coordinate of data named `y`; and 784 x coordinates.

```
<window-source name='w_source'>
       <schema>
          <fields>
             <field name='id'  type='int64' key='true'/>
             <field name='y'  type='double'/>
                        <field name='x1' type='double'/>
                        ... 1
                        <field name='x784' type='double'/>
          </fields>
       </schema>
       <connectors>
          <connector class='fs' name='publisher'>
             <properties>
                <property name='type'>pub</property>
                <property name='fstype'>csv</property>
                <property name='fsname'>./mnist_all.csv</property>
                <property name='transactional'>true</property>
                <property name='blocksize'>1</property>
             </properties>
           </connector>
        </connectors>
</window-source>
```

1 An ellipsis indicates that field name values range from x1 to x784, inclusive.

The Train window `w_training` looks at all the observations and periodically generates a new model using the linear regression algorithm. Model events are published to the Score window `w_scoring`.

```
<window-train name='w_training' algorithm='LinearRegression'>
       <parameters>
```

```
            <properties>
               <property name="nInit">60000</property>
               <property name="commitInterval">10000</property>
               <property name="dampingFactor">1</property>
               <property name="centerFlag">0</property>
               <property name="scaleFlag">0</property>
            </properties>
          </parameters>
          <input-map>
                     <properties>
               <property name="inputs">y,x1,...,x784</property> 1
               <property name="target">y</property>
            </properties>
           </input-map>
   </window-train>
```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the linear regression algorithm in the Train window:

Table A.10 Parameters

Property	Value Type	Property Type	Default Value	Description
`nInit`	int64	Optional	50	Specifies the number of data events used during initialization. The specified value must be a positive integer.
`commitInterval`	int64	Optional	10	Specifies the number of data events to process before triggering a commitment of the model to downstream scoring. The specified value must be a positive integer.
`batchSize`	int64	Optional	0	Specifies the batch size in processing the training samples. The specified value must be a positive integer. This property affects how much memory is used to buffer data events. If you have sufficient memory, set this to the maximum of `nInit` and `commitInterval`.
`dampingFactor`	double	Optional	1	Specifies the damping factor α ($0 \le \alpha \le 1$) for old data points. That is, if the current number of data events to process before triggering a commitment of the model is T, data points arriving at T would have weight 1. Data points at T — t would have weight α^t.
`centerFlag`	Boolean	Optional	0	Specifies whether to center the data (dense part) based on the first `batchSize` data events of the initialization. Specifically, the mean is computed with the first `bufferSize` data events of the initialization, and each data event is subtracted with the computed mean.

Property	Value Type	Property Type	Default Value	Description
`scaleFlag`	Boolean	Optional	0	Specifies whether to scale the data (dense part) based on the first `batchSize` data events of the initialization. Data is scaled so that the variance of the first `batchSize` number of data events is 1.

Table A.11 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`inputs`	string	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 (for example, x,y).
`target`	string	Optional	""	Specifies the target response variable. For linear regression, this must be a continuous variable.

The Score window `w_scoring` scores the data.

```
<window-score name='w_scoring'>
       <schema>
          <fields>
             <field name='id'  type='int64' key='true'/>
             <field name='y'   type='double'/>
                      <field name='yPredictOut'    type='double'/>
                      <field name='modelIdOut'     type='int64'/>
          </fields>
          </schema>
              <models>
                 <online algorithm='LinearRegression'>
                    <input-map>
                       <properties>
                           <property name="inputs">y,x1,...,x784</property>
                       </properties>
                    </input-map>
                    <output-map>
                       <properties>
                           <property name='yPredictOut'>yPredictOut</property>
                           <property name='modelIdOut'>modelIdOut</property>
                       </properties>
                    </output-map>
                  </online>
              </models>
</window-score>
```

The following properties govern the linear regression algorithm in the Score window:

Table A.12 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`inputs`	varlist	Optional	""	Specifies the list of variable names used in clustering. Variable names are defined in the input schema, and they are separated by comma in the list (for example, x,y).

Table A.13 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`yPredictOut`	variable	Optional	""	Specifies the predicted response. If not specified, it is not displayed.
`modelIdOut`	variable	Optional	""	Specifies the column or field name in the output schema that stores the ID of the model from which the score is computed. If not specified, it is not displayed.

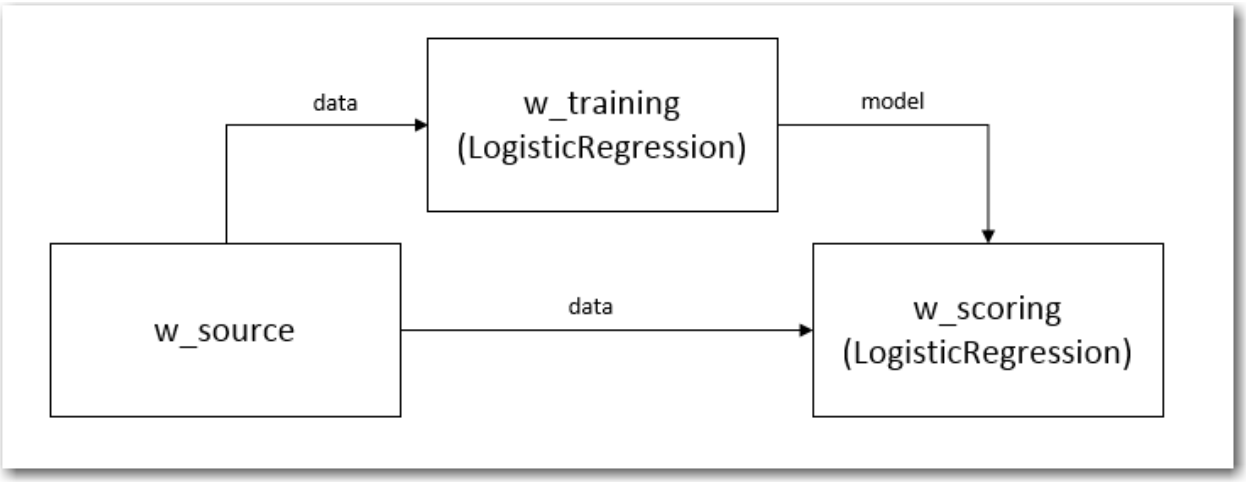
The edges are defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

```
<edges>
            <edge source='w_data'    target='w_train' role='data'/>
            <edge source='w_data'    target='w_score' role='data'/>
            <edge source='w_train'   target='w_score' role='model'/>
          </edges>
```

Training and Scoring Logistic Regression

With logistic regression, the dependent variable is categorical. Some logistic regression models use a binary dependent variable (alive or dead, yes or no, win or lose) and others use a dependent variable with more than two outcome categories.

Consider the following example:



This continuous query includes the following:

- a Source window that receives data events that stream the data to score
- a Train window that generates and periodically updates the logistic regression model
- a Score window that performs the scoring

The Source window `w_source` receives a data event from a file named `mnist_all.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`; a y coordinate of data named `y`; and 784 x coordinates.

```
<window-source name='w_source'>
   <schema>
     <fields>
         <field name='id'  type='int64' key='true'/>
         <field name='y'  type='double'/>
                    <field name='x1' type='double'/>
                    ... 1
                    <field name='x784' type='double'/>
     </fields>
   </schema>
   <connectors>
     <connector class='fs' name='publisher'>
       <properties>
          <property name='type'>pub</property>
          <property name='fstype'>csv</property>
          <property name='fsname'>.mnist_all.csv</property>
          <property name='transactional'>true</property>
          <property name='blocksize'>1</property>
       </properties>
     </connector>
   </connectors>
</window-source>
```

1 An ellipsis indicates that field name values range from x1 to x784, inclusive.

The Train window `w_training` looks at all the observations and periodically generates a new model using the logistic regression algorithm. Model events are published to the score window `w_scoring`.

```
<window-train name='w_training' algorithm='LogisticRegression'>
              <parameters>
                <properties>
                  <property name="nInit">60000</property>
                  <property name="commitInterval">10000</property>
                  <property name="dampingFactor">1</property>
                  <property name="c">1</property>
                  <property name="centerFlag">0</property>
                  <property name="scaleFlag">0</property>
                  <property name="numC">5</property>
                  <property name="ratioC">4</property>
                  <property name="choose">-1</property>
                  <property name="randSeed">123</property>
                  <property name="positiveClass">8</property>
                  <property name="augmentedValue">1</property>
                  <property name="outerIterMax">10</property>
                </properties>
              </parameters>
              <input-map>
```

```
            <properties>
                <property name="inputs">y,x1,...,x784</property> 1
                        <property name="target">y</property>
            </properties>
        </input-map>
</window-train>
```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the linear regression algorithm in the Train window:

Table A.14 Parameters

Property	Value Type	Property Type	Default Value	Description
`nInit`	int64	Optional	50	Specifies the number of data events used during initialization. The specified value must be a positive integer.
`commitInterval`	int64	Optional	10	Specifies the number of data events to process before triggering a commitment of the model to downstream scoring. The specified value must be a positive integer.
`batchSize`	int64	Optional	0	Specifies the batch size in processing the training samples. The specified value must be a positive integer. This property affects how much memory is used to buffer data events. If you have sufficient memory, set this to the maximum of `nInit` and `commitInterval`.
`dampingFactor`	double	Optional	1	Specifies the damping factor α (0 ≤ α≤ 1) for old data points. That is, if the current number of data events to process before triggering a commitment of the model is T, data points arriving at T would have weight 1. Data points at T — t would have weight α^t.
`centerFlag`	Boolean	Optional	0	Specifies whether to center the data (dense part) based on the first `batchSize` data events of the initialization. Specifically, the mean is computed with the first `bufferSize` data events of the initialization, and each data event is subtracted with the computed mean.
`scaleFlag`	Boolean	Optional	0	Specifies whether to scale the data (dense part) based on the first `batchSize` data events of the initialization. Data is scaled so that the variance of the first `batchSize` number of data events is 1.
`c`	double	Optional	0	Specifies the regularization parameter. The specified value must be positive.
`numC`	int64	Optional	1	Specifies the number of regularization parameters to try. The specified value must be positive.

Property	Value Type	Property Type	Default Value	Description
`ratioC`	double	Optional	10	Specifies the ratio in setting the set of regularization parameters.
`choose`	double	Optional	-2	Specifies the criterion in selecting the best regularization parameter. If `choose=-2`, the `c` that achieves the smallest misclassification error is used. If `choose=-1`, the `c` that achieves the smallest hinge loss is used. If choose is nonnegative, the `c` that achieves the largest `choose` score is used.
`randSeed`	int64	Optional	123	Specifies the random seed in reshuffling data events. Specify a positive value. If `randSeed=0`, the data in the buffer is not reshuffled. If `randSeed>0`, the data in the buffer is implicitly reshuffled with the corresponding random seed.
`positiveClass`	double	Optional	1	Specifies the value of the response that is treated as the positive class.
`augmentedValue`	double	Optional	1	Specifies the augmented value for handling the intercept. The specified value must be positive.
`outerIterMax`	int64	Optional	1	
`outerIterMaxInit`	int64	Optional	`outerIterMax`	Specifies the number of outer iterations used in coordinate descent for initialization data events.

Table A.15 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
`inputs`	string	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 (for example, x,y) .
`target`	string	Optional	""	Specifies the target response variable. For logistic regression, it must be a class variable.

The Score window `w_scoring` scores the data.

```
<window-score name='w_scoring'>
     <schema>
        <fields>
           <field name='id'  type='int64' key='true'/>
           <field name='y'   type='double'/>
                   <field name='yPredictOut'    type='double'/>
                   <field name='modelIdOut'     type='int64'/>
                 </fields>
        </schema>
```

```
            <models>
              <online algorithm='LogisticRegression'>
                 <input-map>
                    <properties>
                      <property name="inputs">y,x1,...,x784</property> 1
                    </properties>
                 </input-map>
                 <output-map>
                    <properties>
                      <property name='yPredictOut'>yPredictOut</property>
                      <property name='modelIdOut'>modelIdOut</property>
                    </properties>
                 </output-map>
              </online>
            </models>
  </window-score>
```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the logistic regression algorithm in the Score window:

Table A.16 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
`inputs`	varlist	Optional	""	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 (for example, x,y).

Table A.17 *Output Mapping*

Property	Value Type	Property Type	Default Value	Description
`yPredictOut`	variable	Optional	""	Specifies the predicted response. If not specified, it is not displayed.
`modelIdOut`	variable	Optional	""	Specifies the column or field name in the output schema that stores the ID of the model from which the score is computed. If not specified, it is not displayed.

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

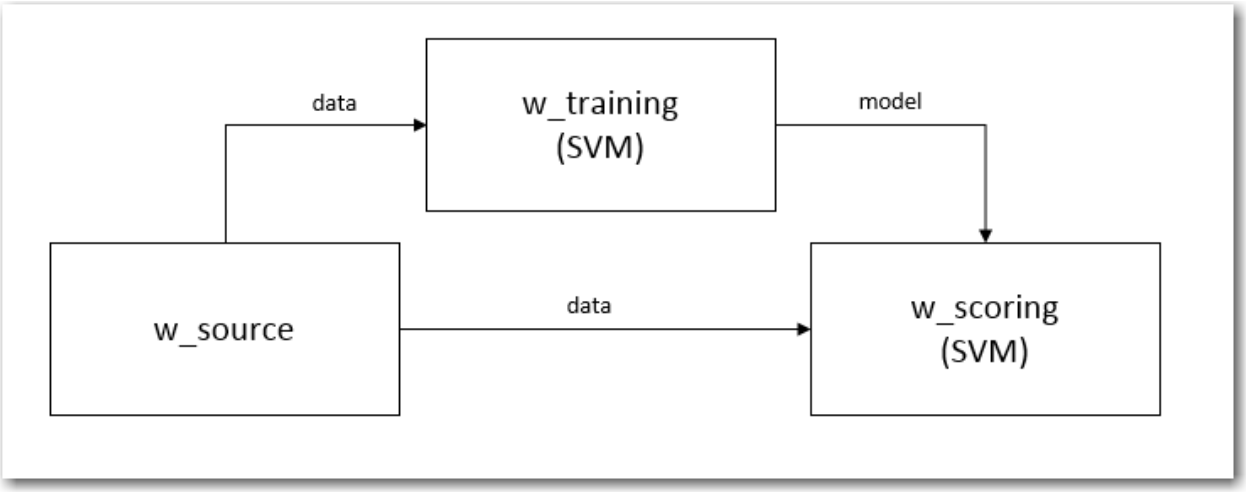
```
<edges>
            <edge source='w_data'    target='w_train' role='data'/>
            <edge source='w_data'    target='w_score' role='data'/>
            <edge source='w_train'   target='w_score' role='model'/>
          </edges>
```

Training and Scoring with Support Vector Machines

Support vector machines are supervised learning models with associated algorithms. Support vector machines apply classification and regression analysis on incoming data. You supply training examples and mark them as belonging to a category. A support vector machine builds a model that assigns new examples to that category.

A support vector machine model represents examples as points in space. Points are mapped onto this space so that examples of each category are separated by a gap. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Consider the following example:



This continuous query includes the following:

- a Source window that receives data events that stream the data to score
- a Train window that generates and periodically updates the vector machine model
- a Score window that performs the scoring

The Source window `w_source` receives a data event from a file named `mnist_all.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`; a y coordinate of data named `y`,; and 784 x coordinates.

```
<window-source name='w_source'>
              <schema>
                <fields>
                    <field name='id'  type='int64' key='true'/>
                    <field name='y'  type='double'/>
                    <field name='x1' type='double'/>
            ... 1
                    <field name='x784' 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/mnist_all.csv</property>
                    <property name='transactional'>true</property>
                    <property name='blocksize'>1</property>
```

```
            </properties>
          </connector>
        </connectors>
      </window-source>
```

1 An ellipsis indicates that field name values range from x1 to x784, inclusive.

The Train window `w_training` looks at all observations and periodically generates a new model using the support vector machines algorithm. Model events are published to the Score window `w_scoring`.

```
<window-train name='w_training' algorithm='SVM'>
        <parameters>
          <properties>
            <property name="nInit">60000</property>
            <property name="commitInterval">10000</property>
            <property name="dampingFactor">1</property>
            <property name="c">1</property>
            <property name="centerFlag">0</property>
            <property name="scaleFlag">0</property>
            <property name="numC">5</property>
            <property name="ratioC">4</property>
            <property name="choose">-1</property>
            <property name="randSeed">123</property>
            <property name="positiveClass">8</property>
            <property name="augmentedValue">1</property>
            <property name="outerIterMax">10</property>
          </properties>
        </parameters>
        <input-map>

            <properties>
            <property name="inputs">y,x1,...,x784</property> 1

            <property name="target">y</property>

          </properties>
        </input-map>
      </window-train>
```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the vector machine algorithm in the Train window:

Table A.18 Parameters

Property	Value Type	Property Type	Default Value	Description
`nInit`	int64	Optional	50	Specifies the number of data events used during initialization.
`commitInterval`	int64	Optional	10	Specifies the number of data events to process before triggering a commitment of the model to downstream scoring. The specified value must be a positive integer.

Property	Value Type	Property Type	Default Value	Description
`batchSize`	int64	Optional	0	Specifies the batch size in processing the training samples. The specified value must be a positive integer. This property affects how much memory is used to buffer data events. If you have sufficient memory, set this to the maximum of `nInit` and `commitInterval`.
`dampingFactor`	double	Optional	1	Specifies the damping factor α (0 ≤ α≤ 1) for old data points. That is, if the current number of data events to process before triggering a commitment of the model is T, data points arriving at T would have weight 1. Data points at T — t would have weight α^t.
`centerFlag`	Boolean	Optional	0	Specifies whether to center the data (dense part) based on the first `batchSize` data events of the initialization. Specifically, the mean is computed with the first `bufferSize` data events of the initialization, and each data event is subtracted with the computed mean.
`scaleFlag`	Boolean	Optional	0	Specifies whether to scale the data (dense part) based on the first `batchSize` data events of the initialization.
`c`	double	Optional	1	Specifies the regularization parameter for vector machines. The specified value must be positive.
`numC`	int64	Optional	1	Specifies the number of regularization parameters to try.
`ratioC`	double	Optional	10	Specifies the ratio in setting the set of regularization parameters. The specified value must be greater than 1.
`choose`	double	Optional	-2	Specifies the criterion in selecting the best regularization parameter. If `choose=-2`, then the `c` that achieves the smallest misclassification error is used. If `choose`=-1, then the `c` that achieves the smallest hinge loss is used. If choose is nonnegative, then the `c` that achieves the largest `choose` score is used.

Property	Value Type	Property Type	Default Value	Description
randSeed	int64	Optional	123	Specifies the random seed in reshuffling data events. Specify a positive value. If randSeed=0, the data in the buffer is not reshuffled. If randSeed>0, the data in the buffer is implicitly reshuffled with the corresponding random seed.
positiveClass	double	Optional	1	Specifies the value of the response that is treated as the positive class.
augmentedValue	double	Optional	1	Specifies the augmented value for handling the intercept. The specified value must be positive.
outerIterMax	int64	Optional	1	Specifies the number of outer iterations used in coordinate descent for non-initialization data events. The specified value must be positive.
outerIterMaxInit	int64	Optional	outerIterMax	Specifies the number of outer iterations used in coordinate descent for initialization data events. The specified value must be positive.

Table A.19 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
inputs	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 (for example, x,y).
target	string	Optional	""	Specifies the target response variable.

The Score window w_scoring scores the data.

```
<window-score name='w_scoring'>
              <schema>
                <fields>
                    <field name='id'  type='int64' key='true'/>
                    <field name='y'   type='double'/>
                    <field name='yPredictOut'    type='double'/>
                              <field name='modelIdOut'     type='int64'/>

                </fields>
              </schema>
              <models>
                <online algorithm='SVM'>
                  <input-map>
                    <properties>
                  <property name="inputs">y,x1,...,x784</property> 1
```

```
            </properties>
          </input-map>
          <output-map>
            <properties>
              <property name='yPredictOut'>yPredictOut</property>
              <property name='modelIdOut'>modelIdOut</property>
            </properties>
          </output-map>
        </online>
      </models>
    </window-score>
```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the vector machine algorithm in the Score window:

Table A.20 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`inputs`	varlist	Optional	""	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 (for example, x,y). The mapping should be identical to that used in the Train window.

Table A.21 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`yPredictOut`	variable	Optional	""	Specifies the predicted response. If not specified, it is not displayed.
`modelIDOut`	variable	Optional	""	Specifies the column or field name in the output schema that stores the ID of the model from which the score is computed. If not specified, it is not displayed.

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

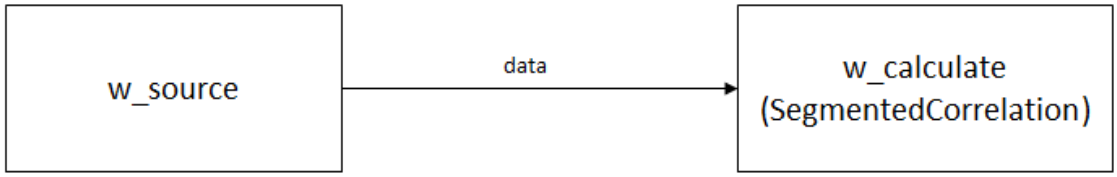
```
<edges>
            <edge source='w_data'    target='w_train' role='data'/>
            <edge source='w_data'    target='w_score' role='data'/>
            <edge source='w_train'   target='w_score' role='model'/>
          </edges>
```

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:

Table A.22 *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.

Property	Value Type	Property Type	Default Value	Description
`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`.

Table A.23 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`x`	variable	Required	No default value	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.

Table A.24 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`corOut`	variable	Optional	" " (empty string)	Specifies the output variable name for the segmented correlation.

The calculated segmented correlation output variable values 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.

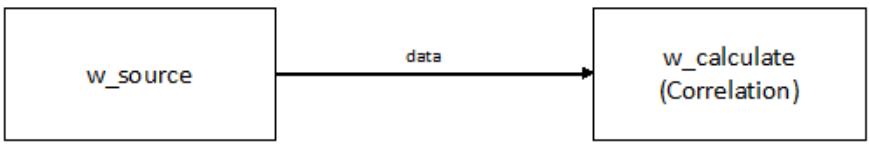
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

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

Calculating Streaming Pearson's 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:

Table A.25 Parameters

Property	Value Type	Property Type	Default Value	Description
`windowLength`	int64	Optional	`3`	Specifies the length of the sliding window. The default value is 3.

Table A.26 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.27 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`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 variable values are published through a file-and-socket connector to a CSV file named **`result.out`**.

The edge is defined at the end of the project.

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

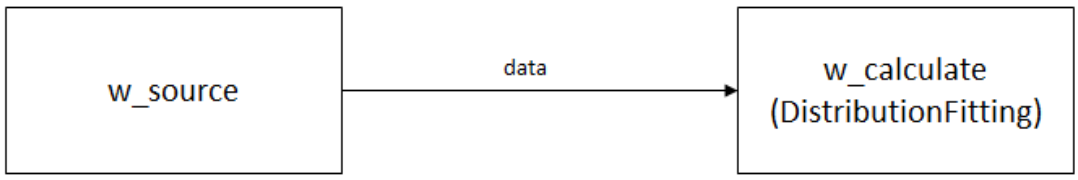
You can view the default values of the streaming correlation algorithm parameter properties for the Calculate window with the command-line utility.

Streaming Distribution Fitting

The distribution fitting algorithm fits a 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:

Table A.28 *Parameters*

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.29 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`x`	variable	Required	No default value	Specifies the input variable for distribution fitting.

Table A.30 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`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 variable values are published through a file-and-socket connector to a CSV file named **`result.out`**.

The edge is defined at the end of the project.

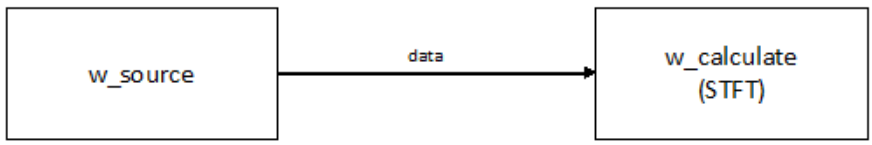
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the default values of the streaming distribution fitting algorithm parameter properties for the Calculate window with the 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:

Table A.31 Parameters

Property	Value Type	Property Type	Default Value	Description
`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. **Note:** Do not specify a value of `binsInSchema` greater than 999.
`overlap`	int64	Optional	`127`	Specifies the overlap between consecutive windows. Must be strictly less than `windowLength`.

Table A.32 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`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`.

Table A.33 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`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.
`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 variable values are published through a file-and-socket adapter to a CSV file named **`result.out`**.

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

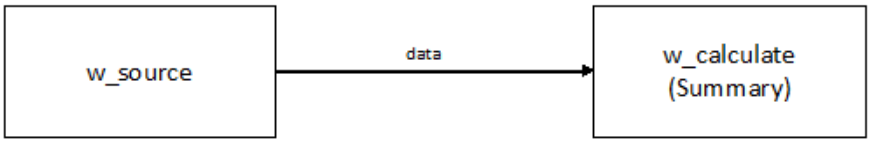
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

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

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:

Table A.34 *Parameters*

Property	Value Type	Property Type	Default Value	Description
`windowLength`	int64	Optional	3	Specifies the length of the sliding window. The default value is 3.

Table A.35 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.36 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`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).
`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 connector to a CSV file named **`result.out`**.

The edge is 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 thecommand-line utility.

Computing Fit Statistics for Scored Results

The goodness of fit of a statistical model describes how well a model fits a set of data. Goodness–of–fit measures summarize the difference between observed values and predicted values of the model under consideration. The goodness–of–fit algorithm that is provided with SAS Event Stream Processing Analytics calculates fit statistics such as the following:

- average square error
- mean square logarithmic error
- mean absolute error
- mean consequential error
- multiclass log loss

You can apply these metrics to the output of a model (from a Score window) to compare models.

For more information, see the ASSESS Procedure in *SAS Visual Data Mining and Machine Learning: Statistical Procedures*.

Consider the following example:

The continuous query includes the following:

- a Source window that receives scored data from a Score window to be analyzed
- a Calculate window that runs the algorithm calculating fit statistics

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`. `x_c` contains an observed value, and `y_c` contains a value predicted by a regression model.

```
<window-source name='w_source' insert-only='true' index='pi_EMPTY'>
              <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 Calculate window `w_calculate` receives data events. It publishes goodness–of–fit statistics according to the algorithm properties that are specified.

```
<window-calculate name='w_calculate' algorithm='FitStat'>
   <schema>
              <fields>
                <field name='id' type='int64' key='true'/>
                <field name='nOut' type='double'/>
                <field name='nmissOut' type='double'/>
                <field name='aseOut' type='double'/>
                <field name='divOut' type='double'/>
                <field name='raseOut' type='double'/>
                <field name='mceOut' type='double'/>
                <field name='mcllOut' type='double'/>
                <field name='maeOut' type='double'/>
                <field name='rmaeOut' type='double'/>
                <field name='msleOut' type='double'/>
                <field name='rmsleOut' type='double'/>
              </fields>
            </schema>
<input-map>
   <properties>
            <property name="inputs">y_c</property>
            <property name="response">x_c</property>
   </properties>
</input-map>
<output-map>
   <properties>
                <property name="nOut">nOut</property>
                <property name="nmissOut">nmissOut</property>
                <property name="aseOut">aseOut</property>
                <property name="divOut">divOut</property>
                <property name="raseOut">raseOut</property>
                <property name="mceOut">mceOut</property>
                <property name="mcllOut">mcllOut</property>
                <property name="maeOut">maeOut</property>
                <property name="rmaeOut">rmaeOut</property>
                <property name="msleOut">msleOut</property>
                <property name="rmsleOut">rmsleOut</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 algorithm in the Calculate window:

Table A.37 Parameters

Property	Value Type	Property Type	Default Value	Description
windowLength	int64	Optional	3	Specifies the length of the sliding window.

Table A.38 Input Mapping

Property	Value Type	Property Type	Default Value	Description
inputs	variable(s)	Required	No default value	Specifies input variables. For regression models, only one input variable is required. That variable specifies the predicted response. For classification models, this variable list specifies the predicted probabilities for each response class.
response	response variable	Required	No default value	Specifies the response variable (that is, the target variable).
classLabels	string list	Required for classificati on variables	No default value	Specifies the corresponding label for each predicted probability.
labelLen	int64	Optional	128	Specifies the length of the response labels.

Table A.39 Output Mapping

Property	Value Type	Property Type	Default Value	Description
nOut	variable	Optional	""	Specifies the output variable for the number of observations (N).
nmissOut	variable	Optional	""	Specifies the output variable for the number of missing values (NMISS).
aseOut	variable	Optional	""	Specifies the average square error (ASE). $\mathrm{ASE} = \frac{1}{\mathrm{N}} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ ■ y_i is the actual target value of observation *i* ■ $\hat{y}_i$ is the predicted target value of observation *i*
divOut	variable	Optional	""	Specifies the divisor of the average square error.

Property	Value Type	Property Type	Default Value	Description
`raseOut`	variable	Optional	""	Specifies the root average square error. $\mathrm{RASE} = \sqrt{\mathrm{ASE}}$
`mceOut`	variable	Optional	""	Specifies the mean consequential error. $\mathrm{MCE} = \frac{1}{\mathrm{N}} \sum_{t_i \neq \hat{t}_i} 1$
`mcllOut`	variable	Optional	""	Specifies the multiclass log loss. $\mathrm{logloss} = -\frac{1}{\mathrm{N}} \sum_{i=1}^{n} \sum_{j=1}^{m} y_{i,j} \log(p_{i,j})$
`maeOut`	variable	Optional	""	Specifies the mean absolute error.
`rmaeOut`	variable	Optional	""	Specifies the root mean absolute error.
`msleOut`	variable	Optional	""	Specifies the mean square logarithmic error.
`rmsleOut`	variable	Optional	""	Specifies the root mean square logarithmic error.

The calculated output variable values are published through a file-and-socket connector to a CSV file named **`result.out`**.

The edge is defined at the end of the project.

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

Computing Receiver Operating Characteristic (ROC) Information

Receiver operating characteristic (ROC) information shows the diagnostic ability of a classifier system as you vary its discrimination threshold. You create ROC information by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

In an ROC information table, the confusion matrix is calculated based on the event in each cutoff point.

- *m* is the total cutoff points
- *n* is the number of observations
- *N* is the sum of observation frequencies in the data
- w^i are the observation frequencies
- a_k is true positive at cutoff point *k*, $k \in [0, m-1]$
- b_k is false positive at cutoff point *k*, $k \in [0, m-1]$
- c_k is false negative at cutoff point *k*, $k \in [0, m-1]$

For more information, see the following:

- The ASSESS Procedure in *SAS Visual Data Mining and Machine Learning: Statistical Procedures*.

- Fawcett, T. (2006). “An Introduction to ROC Analysis.” *Pattern Recognition Letters* 27:861–874. Special issue on ROC analysis in pattern recognition, edited by F. Tortorella.

Consider the following example.

The continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that performs the ROC calculation

The Source window `w_source` a data event that consists of the contents of a file named `input.csv`. It receives the event through a file-and-socket connector.

```
<window-source name='w_source'>
              <schema>
                <fields>
                  <field name='id'  type='int64' key='true'/>
                  <field name='y_c' type='string'/>
                  <field name='p_0' type='double'/>
                  <field name='p_1' type='double'/>
                  <field name='p_2' 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 several variables. It publishes a confusion table.

```
<window-calculate name='w_calculate' algorithm='ROC'>
       <schema>
                <fields>
                  <field name='id' type='int64' key='true'/>
                  <field name='binIdOut' type='int64' key='true'/>
                  <field name='cutOffOut' type='double'/>
                  <field name='tpOut' type='double'/>
                  <field name='fpOut' type='double'/>
                  <field name='fnOut' type='double'/>
                  <field name='tnOut' type='double'/>
```

```
            <field name='sensitivityOut' type='double'/>
            <field name='specificityOut' type='double'/>
            <field name='ksOut' type='double'/>
            <field name='ks2Out' type='double'/>
            <field name='fHalfOut' type='double'/>
            <field name='fprOut' type='double'/>
            <field name='accOut' type='double'/>
            <field name='fdrOut' type='double'/>
            <field name='f1Out' type='double'/>
            <field name='cOut' type='double'/>
            <field name='giniOut' type='double'/>
            <field name='gammaOut' type='double'/>
            <field name='tauOut' type='double'/>
            <field name='miscEventOut' type='double'/>
          </fields>
</schema>
        <parameters>
          <properties>
             <property name='cutStep'>0.1</property>
             <property name='event'>good</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
              <property name="input">p_0</property>
              <property name="response">y_c</property>
          </properties>
        </input-map>
        <output-map>
           <properties>
              <property name="binIdOut">binIdOut</property>
              <property name="cutOffOut">cutOffOut</property>
              <property name="tpOut">tpOut</property>
              <property name="fpOut">fpOut</property>
              <property name="fnOut">fnOut</property>
              <property name="tnOut">tnOut</property>
              <property name="sensitivityOut">sensitivityOut</property>
              <property name="specificityOut">specificityOut</property>
              <property name="ksOut">ksOut</property>
              <property name="ks2Out">ks2Out</property>
              <property name="fHalfOut">fHalfOut</property>
              <property name="fprOut">fprOut</property>
              <property name="accOut">accOut</property>
              <property name="fdrOut">fdrOut</property>
              <property name="f1Out">f1Out</property>
              <property name="cOut">cOut</property>
              <property name="giniOut">giniOut</property>
              <property name="gammaOut">gammaOut</property>
              <property name="tauOut">tauOut</property>
              <property name="miscEventOut">miscEventOut</property>
          </properties>
        </output-map>
        <connectors>
          <connector class='fs' name='sub'>
            <properties>
              <property name='type'>sub</property>
```

```
            <property name='fstype'>json</property>
            <property name='header'>true</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 ROC algorithm in the Calculate window:

Table A.40 Parameters

Property	Value Type	Property Type	Default Value	Description
windowLength	int64	Optional	3	specifies the length of the sliding window

Table A.41 Input Mapping

Property	Value Type	Property Type	Default Value	Description
input	variable	Required	No default value	input variable. It specifies the predicted probability for the given event.
response	response variable	Required	No default value	specifies the response variable
event	string	Required	No default value	specifies the desired response event to be used for ROC calculations.
cutStep	double	Optional	0.01	specifies the bin width. It should be between 0 and 1. With 0.01 as the default, 100 bins are generated to fit the ROC.
labelLen	int64	Optional	128	specifies the length of response event.
windowLength	int64	Optional	0	specifies the length of sliding window

Table A.42 Output Mapping

Property	Value Type	Property Type	Default Value	Description
binIDout	variable	Optional	" " (empty string)	bin ID
cutoffOut	variable	Optional	" " (empty string)	cutoff probability
tpOut	variable	Optional	" " (empty string)	true positives
fpOut	variable	Optional	" " (empty string)	false positives

Property	Value Type	Property Type	Default Value	Description
`fnOut`	variable	Optional	" " (empty string)	false negatives
`tnOut`	variable	Optional	" " (empty string)	true negatives
`sensitivityOut`	variable	Optional	" " (empty string)	sensitivity
`specificityOut`	variable	Optional	" " (empty string)	specificity
`ksOut`	variable	Optional	" " (empty string)	Kolmogorov-Smirnov statistic
`ks2Out`	variable	Optional	" " (empty string)	KS2
`fHalfOut`	variable	Optional	" " (empty string)	F_Half
`fprOut`	variable	Optional	" " (empty string)	false positive rate
`accOut`	variable	Optional	" " (empty string)	accuracy
`fdrOut`	variable	Optional	" " (empty string)	false discovery rate
`f1Out`	variable	Optional	" " (empty string)	F1 score
`cOut`	variable	Optional	" " (empty string)	C (area under the curve) $C = \frac{\mu + \frac{\theta}{2}}{\rho}$
`giniOut`	variable	Optional	" " (empty string)	Gini coefficient $\text{Gini} = \frac{\mu - \omega}{\rho}$
`gammaOut`	variable	Optional	" " (empty string)	Goodman Kruskal Gamma $\text{gamma} = \frac{\mu - \omega}{\mu + \omega}$
`tauOut`	variable	Optional	" " (empty string)	Kendall's Tau-a $\text{tau} = \frac{\mu - \omega}{\frac{N}{2}\left(N - 1\right)}$
`miscEventOut`	variable	Optional	" " (empty string)	Misclassification rate (1– area under the curve)

For these output variables, the following is true:

$$\theta = \sum_{k=1}^{m} ((a_{k-1} - a_k)(b_{k-1} - b_k))$$

$$\mu = \sum_{k=2}^{m} \left((a_{k-1} - a_k) \right) \sum_{j=1}^{k} (b_{j-1} - b_j))$$

$$\omega = \sum_{k=1}^{m} ((a_{k-1} - a_k) \sum_{j=k+1}^{m} (b_{j-1} - b_j))$$

$$\rho = a_0 b_0, a_m = 0 \quad \text{and} \quad b_m = 0$$

The calculated output variable values are published through a file-and-socket connector to a CSV file named **`result.out`**.

The edge is defined at the end of the project.

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

Calculate a Streaming Histogram

A *histogram* graphically represents a distribution of numerical data. This algorithm processes a stream of numerical data and puts it in bins to generate boundaries for creating a histogram that fits it.

Specific features of this algorithm are as follows:

- It keeps track of the center and the height of each bin, so all the points that are encapsulated in a bin are represented by that bin center.
- It can insert every new point in a logarithmic time with respect to the number of bins. If nBins is the number of bins, the insertion needs O(lg nBins) time. Thus, you can use a lot of bins with very little time penalty.
- There is a fading factor value called alpha that fades the height of each bin. This, in effect, forgets old data and gives greater weight to the most recent data. You can set the fading factor directly by using the alpha value or by setting the half-life-steps value.
- The algorithm can calculate a good approximation of any quantile on the histogram. This approximation improves as the number of bins increases.

Consider the following example:

The continuous query includes the following:

- a Source window that receives the data to fit into a histogram
- a Calculate window that performs bucketing

The Source window `w_source` receives input data, a file named `test.csv`, through a file–and–socket connector.

```
<window-source name='w_source' insert-only='true' index='pi_EMPTY'>
              <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'>./test.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 output variable values for bin centers and heights and for quantiles.

```
<window-calculate name='w_calculate' algorithm='Histogram'>
    <schema>
      <fields>
        <field name='id'         type='int64' key='true'/>
        <field name='binCenters' type='array(dbl)'/>
        <field name='binHeights' type='array(dbl)'/>
        <field name='quantiles'  type='array(dbl)'/>
      </fields>
    </schema>
    <parameters>
      <properties>
        <property name="nBins">5</property>
        <property name="alpha">0.999</property>
        <property name="quantileList">0.25,0.50,0.75</property>
      </properties>
    </parameters>
    <input-map>
      <properties>
        <property name="input">x_c</property>
      </properties>
    </input-map>
    <output-map>
      <properties>
        <property name="binCentersOut">binCenters[1-20]</property>
        <property name="binHeightsOut">binHeights[1-20]</property>
        <property name="quantilesOut">quantiles[1-3]</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 Histogram algorithm in the Calculate window:

Table A.43 Parameters

Property	Value Type	Property Type	Default Value	Description
`nBins`	int64	Optional	`20`	Specifies the maximum number of bins in the histogram.
`alpha`	double	Optional	`1.0`	Specifies the fading out factor (0 < α <= 1). The recommended value for alpha is greater than 0.997.
`halfLifeSteps`	int64	Optional	`0`	Specifies the number of steps at which the weight of the input reaches half of its original weight.
`quantileList`	double-list	Optional	`""` (empty string)	Specifies a comma-separated list that contains quantiles to compute. Probabilities must be in the range [0,1] and sorted in ascending order.
`reportInterval`	int64	Optional	`5`	Specifies the interval of reporting histogram and quantile results (if any).

Table A.44 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`input`	variable	Required	No default value	Specifies the input variable with which to build the histogram.

Table A.45 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`binCentersOut`	variable list	Optional	`""` (empty string)	Specifies a list of output variable names for bin centers.
`binHeightsOut`	variable list	Optional	`""` (empty string)	Specifies a list of output variable names for bin heights.
`quantilesOut`	variable list	Optional	`""` (empty string)	Specifies a list of output variable names for quantiles.

The calculated output variable values are published through a file-and-socket connector to a CSV file named **`result.out`**.

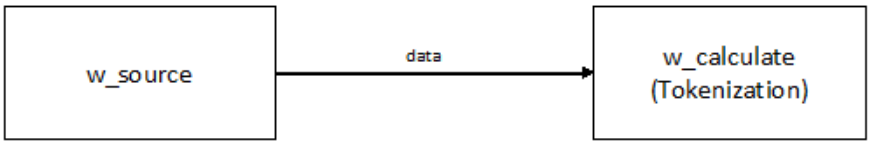
The edge is defined at the end of the project.

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

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>
      <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:

Table A.46 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
`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.

Table A.47 *Output Mapping*

Property	Value Type	Property Type	Default Value	Description
`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.

Property	Value Type	Property Type	Default Value	Description
`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 command-line utility.

Streaming Text Vectorization

The Calculate window supports text vectorization through a proprietary vectorization algorithm. Vectorizing text creates maps from words or n-grams to vector space. A *vector space* is an algebraic model to represent text documents as vectors of identifiers (for example, index terms).

Consider the following example:

This continuous query includes the following:

- a Source window that receives the text data to analyze
- a Calculate window that vectorizes 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.

```
<window-source name='w_source'>
      <schema>
         <fields>
            <field name='docId'   type='int64' key='true'/>
            <field name='tokenId' type='int64' key='true'/>
            <field name='word'    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.csv</property>
                 <property name='transactional'>true</property>
                 <property name='blocksize'>1</property>
                 <property name='rate'>1</property>
              </properties>
            </connector>
          </connectors>
    </window-source>
```

The Calculate window `w_calculate` receives data events and publishes word vectors created with the vectorization algorithm.

```
<window-calculate name='w_calculate' algorithm='TextVectorization'>
              <schema>
                <fields>
                  <field name='docId'   type='int64' key='true'/>
                  <field name='tokenId' type='int64' key='true'/>
                  <field name='word'    type='string'/>
                  <field name='v1' type='double'/>
                  <field name='v2' type='double'/>
                  <field name='v3' type='double'/>
                  <field name='v4' type='double'/>
                  <field name='v5' type='double'/>
                </fields>
              </schema>
              <parameters>
                <properties>
                  <property name="wordVec">wordVec1.csv</property>
                  <property name='outputDocVec'>0</property>
                  <property name="wordVecDelimiter">COMMA</property>
                </properties>
              </parameters>
              <input-map>
                <properties>
                  <property name='docId'>docId</property>
                  <property name='token'>word</property>
                </properties>
              </input-map>
              <output-map>
                <properties>
                  <property name='docIdOut'>docId</property>
                  <property name='vectorOut'>v1,v2,v3,v4,v5</property>
                </properties>
              </output-map>
            </window-calculate>
          </windows>
```

The following properties govern the vectorization algorithm in the Calculate window:

Table A.48 Parameters

Property	Value Type	Property Type	Default Value	Description
`wordVec`	string	Required	No default value	Specifies the word vector filename.
`wordVecDelimiter`	string	Optional	"COMMA"	Specifies the delimiter of the word vector file. Legal values are **`"COMMA"`**, **`"TAB"`**, or **`"SPACE"`**.
`wordVecLineBreak`	string	Optional	"LF"	Specifies the line break of the word vector file. It can be **`"LF"`**, **`"CR"`**, or **`"CRLF"`**.
`startList`	string	Optional	""	Specifies the filename of the start list, which contains the words that are considered during vectorization.
`stopList`	string	Optional	""	Specifies the filename of the stop list, which contains the words that are ignored.
`outputDocVec`	int64	Optional	0	Specifies whether to return a document vector or not. If it is set to 0, then word vectors are returned. Otherwise, document vectors are returned.

Table A.49 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`docID`	variable	Optional	""	Specifies the input variable name of a document ID. It is required when `outputDocVec` is set to nonzero.
`token`	variable	Required	No default value	Specifies the input variable name of a token.

Table A.50 Output Mapping

Property	Value Type	Property Type	Default Value	Description
`docIDOut`	variable	Optional	""	Specifies the output variable name of a document ID. It is required when `outputDocVec` is set to nonzero.
`vectorOut`	variable	Required	No default value	Specifies a list of output variable names for word or document vectors.

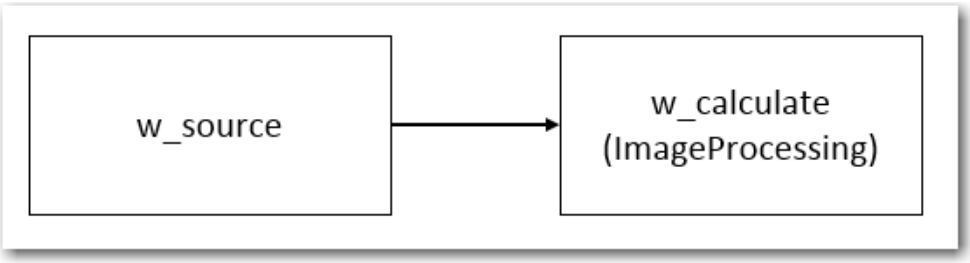
The edge is defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

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

Processing Image Data

SAS Event Stream Processing provides an image processing algorithm that you can use on streaming image data. In the Calculate window, you specify one of the following processing functions to apply to the incoming image: `resize`, `crop` `rotate`, , or `flip`.

Consider the following example:



The continuous query includes the following:

- a Source window that receives images
- a Calculate window that runs the image processing algorithm on those images

Here is code for the Source window:

```
<window-source index="pi_EMPTY" insert-only="true" name="w_source">
       <schema>
         <fields>
           <field key="true" name="id" type="int64" />
           <field key="false" name="image" type="blob" />  1
         </fields>
       </schema>
</window-source>
```

1 The input data image is a binary large object.

The following Calculate window applies the `crop` function to that image:

```
<window-calculate algorithm="ImageProcessing" name="w_calculate">
       <schema>
          <fields>
             <field key="true" name="id" type="int64" />
             <field key="false" name="resized" type="blob" />
          </fields>
        </schema>
        <parameters>
           <properties>
              <property name="function">crop</property>
              <property name="width">200</property>
              <property name="outputHeight">250</property>
              <property name="outputWidth">250</property>
              <property name="y">50</property>
              <property name="x">50</property>
              <property name="height">200</property>
           </properties>
         </parameters>
         <input-map>
            <properties>
```

```
                <property name="imageInput">image</property>
            </properties>
            </input-map>
            <output-map>
                <properties>
                    <property name="imageOutput">cropped</property>
                </properties>
            </output-map>
</window-calculate>
```

The following Calculate window applies the `resize` function to the image:

```
<window-calculate algorithm="ImageProcessing" name="w_calculate">
        <schema>
            <fields>
                <field key="true" name="id" type="int64" />
                <field key="false" name="resized" type="blob" />
            </fields>
        </schema>
        <parameters>
            <properties>
                <property name="function">resize</property>
                <property name="width">200</property>
                <property name="height">200</property>
            </properties>
        </parameters>
        <input-map>
            <properties>
                <property name="imageInput">image</property>
            </properties>
        </input-map>
        <output-map>
            <properties>
                <property name="imageOutput">resized</property>
            </properties>
        </output-map>
</window-calculate>
```

The following properties govern the image processing algorithm in the Calculate window:

Table A.51 *Parameters*

Property	Value Type	Property Type	Default Value	Description
`function`	string	Required	`"resize"`	Specifies the image processing function to be applied: `resize`, `crop`, `rotate`, or `flip`.
`preFlip`	int32	Optional	-1000	Specifies whether the input image is flipped before processing. This is used for video streaming.
`x`	int64	Optional	0	Specifies the x location. Useful for the `crop` function.
`y`	int64	Optional	0	Specifies the y location. Useful for the `crop` function.
`width`	int64	Optional	100	Specifies the width of an image. Useful for the `crop` and `resize` functions.

Property	Value Type	Property Type	Default Value	Description
`height`	int64	Optional	100	Specifies the height of an image. Useful for the `crop` function.
`outputWidth`	int64	Optional	0	Specifies the output width of an image. Useful for the `crop` function.
`outputHeight`	int64	Optional	0	Specifies the output height of an image. Useful for the `crop` and `resize` functions.
`theta`	double	Optional	0	Specifies the theta parameter (the rotation angle when you use the `rotate` function).
`type`	double	Optional	0	Specifies the type. Useful for the `flip` function.

Table A.52 *Input Mapping*

Property	Value Type	Property Type	Default Value	Description
`imageInput`	blob variable	Required	No default value	Specifies the input image.

Table A.53 *Output Mapping*

Property	Value Type	Property Type	Default Value	Description
`imageOutput`	blob variable	Required	No default value	Specifies the output image.

The edge is defined at the end of the project.

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

Computing the Moving Relative Range

The moving relative range (MRR) provides a measure of volatility for a nonstationary time series, where the mean and the variance of the series change over time.

Let X_i denote the tth element of the time series. The Range and the MRR for X_i is computed as follows:

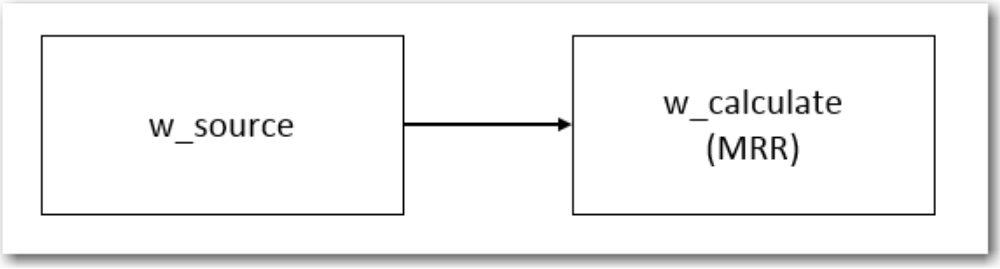
$$Range_t = Range\left(X_t,\ X_{t-1}, \ldots X_{t-M+1}\right)$$

$$MRR_t = \frac{Range_t}{Median\ \left(Range_t,\ Range_{t-1},\ \ldots,\ Range_{t-K+1}\right)}$$

M is the window length to calculate the range. K is the window length to compute the moving relative range. The process computes the range over the last M data points, and then it uses that computed range over the last K points to compute the MRR.

For more information, see “Time Filters Package for the TSMODEL Procedure” in the *SAS Visual Forecasting: Time Series Packages*.

Consider the following example:



The continuous query includes the following:

- a Source window that receives the data to analyze
- a Calculate window that performs the moving relative range calculation

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 two fields for each observation: an ID that acts as the data stream’s key, named ID, and a variable `x1` that represents an element of the time series.

```
<window-source name='w_source'>
       <schema>
           <fields>
               <field name='id'  type='int64' key='true'/>
                   <field name='x1' 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 from `w_source`.

```
<window-calculate name='w_calculate' algorithm='MRR'>
       <schema>
           <fields>
               <field name='id' type='int64' key='true'/>
                        <field name='rangeOut' type='double'/>
               <field name='erangeOut' type='double'/>
           </fields>
         </schema>
         <parameters>
             <properties>
                 <property name='rangeWindowLength'>3</property>
                 <property name='expRangeWindowLength'>3</property>
```

```
            </properties>
        </parameters>
        <input-map>
            <properties>
                <property name="input">x1</property>
                <property name="timeId">id</property>
            </properties>
         </input-map>
         <output-map>
             <properties>
                 <property name="timeIdOut">id</property>
                  <property name="erangeOut">erangeOut</property>
                <property name="rangeOut">rangeOut</property>
             </properties>
         </output-map>
         <connectors>
             <connector class='fs' name='sub'>
                 <properties>
                     <property name='type'>sub</property>
                     <property name='fstype'>csv</property>
                     <property name='header'>true</property>
                     <property name='fsname'>TEST_OUTPUT/output/result.out</property>
                     <property name='snapshot'>true</property>
                 </properties>
             </connector>
          </connectors>
</window-calculate>
```

The moving relative range algorithm is governed by the following properties:

Table A.54 Parameters

Property	Value Type	Property Type	Default Value	Description
`rangeWindowLength`	int64	Optional	`128`	Specifies the window length to calculate the range (M). For a time series whose mean is changing quickly, specify a lower value.
`expRangeWindowLength`	int64	Optional	`128`	Specifies the window length to calculate the moving relative range (K). For a time series whose variance is changing quickly, specify a lower value.

Table A.55 Input Mapping

Property	Value Type	Property Type	Default Value	Description
`input`	variable	Required	No default value	Specifies the analysis variable name in the input stream.
`timeId`	variable	Required	No default value	Specifies the time ID variable name in the input stream.

Table A.56 *Output Mapping*

Property	Value Type	Property Type	Default Value	Description
`timeIdOut`	variable	Required	No default value	Specifies the time ID (key) variable name in the output stream.
`erangeOut`	variable	Required	No default value	Specifies the expected range variable name in the output stream.
`rangeOut`	variable	Optional	`""` (empty string)	Specifies the range variable name in the output stream.

The calculated output variable values are published though a file-and-socket connector to a CSV file named **`result.out`**.

The edge is defined at the end of the project.

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

Offline Projects

Overview

Offline 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 through a Model Reader window 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, see the *SAS Visual Data Mining and Machine Learning: Data Mining and Machine Learning Procedures*.

Note: When you persist a model that performs offline training and then restore the model, you must republish the model to the Model Reader window to enable the Score window to score new events. For more information, see "Persist and Restore Operations" in SAS Event Stream Processing: Creating and Using Windows .

The following algorithms are supported for offline projects:

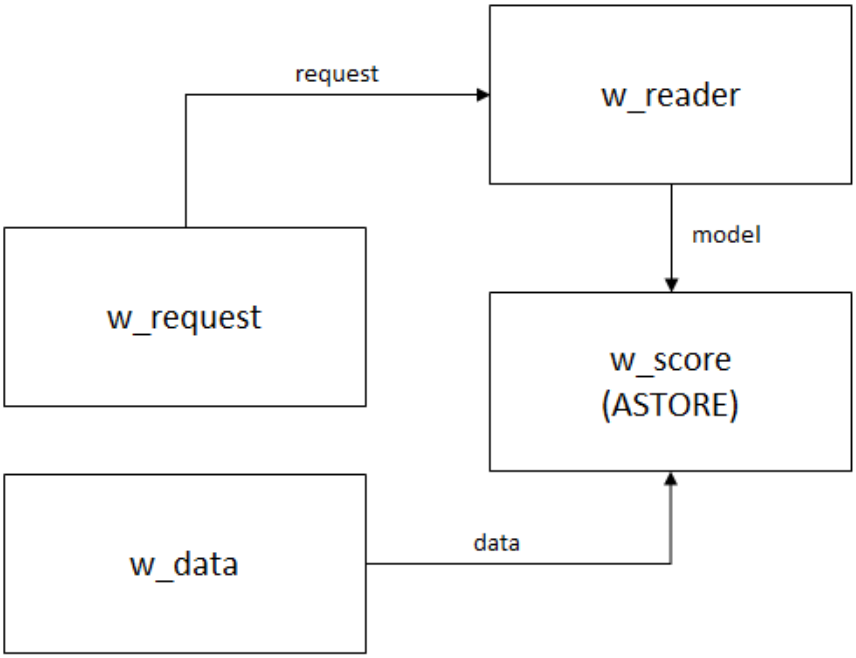
Table A.57 *Supported Algorithms*

Algorithm	Description
Support Vector Data Description	The Support Vector Data Description obtains a spherically shaped boundary around 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.
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.

Algorithm	Description
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.
Robust Principal Components Analysis	Decomposes an input matrix into a sum of two matrices: a low-rank matrix and a sparse matrix. You can use the low-rank matrix to do feature extraction and use the sparse matrix to detect anomalies.
Deep Neural Networks	Deep neural networks are artificial neural networks that have multiple hidden layers between the input and output layers. Successive model layers can learn deeper intermediate representations.
Convoluted Neural Networks	Convoluted neural networks are a type of deep neural network. They are feed-forward artificial neural networks that can be applied to analyzing visual imagery. They consist of neurons that have learnable weights and biases. **Note:** SAS Event Stream Processing does not support offline Convoluted Neural Network models that are trained using decompressed images. When you load images to a Convoluted Neural Network model definition using the loadImages action of the images action set in SAS Cloud Analytic Services, set the required DECODE parameter to FALSE before you train and export a model as an ASTORE file.
Bayesian Network	Scores a data set using a Bayesian network model.

Example: ASTORE with Random Forest

Consider the following example:



There are two Source windows. The first reads 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>
```

The file iris_esp.csv contains the data to be scored.

```
1    50    33    14    2    Setosa
2    46    34    14    3    Setosa
...
```

The second reads 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>
```

The file reader_request.csv contains a list of request events.

```
i    n    1    type    astore
i    n    2    reference    forest_iris_astore.sasast
i    n    3
```

A Model Reader window (`w_reader`) receives these requests from `w_request`, fetches the specified 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_astore`.

Note: Make sure that the ASTORE file is loaded before you stream data events through the Score window.

```
i,n,2,action,load
i,n,2,type,astore
i,n,3,reference,forest_iris_astore.sasast
i,n,4,,
```

```
<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>
```

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

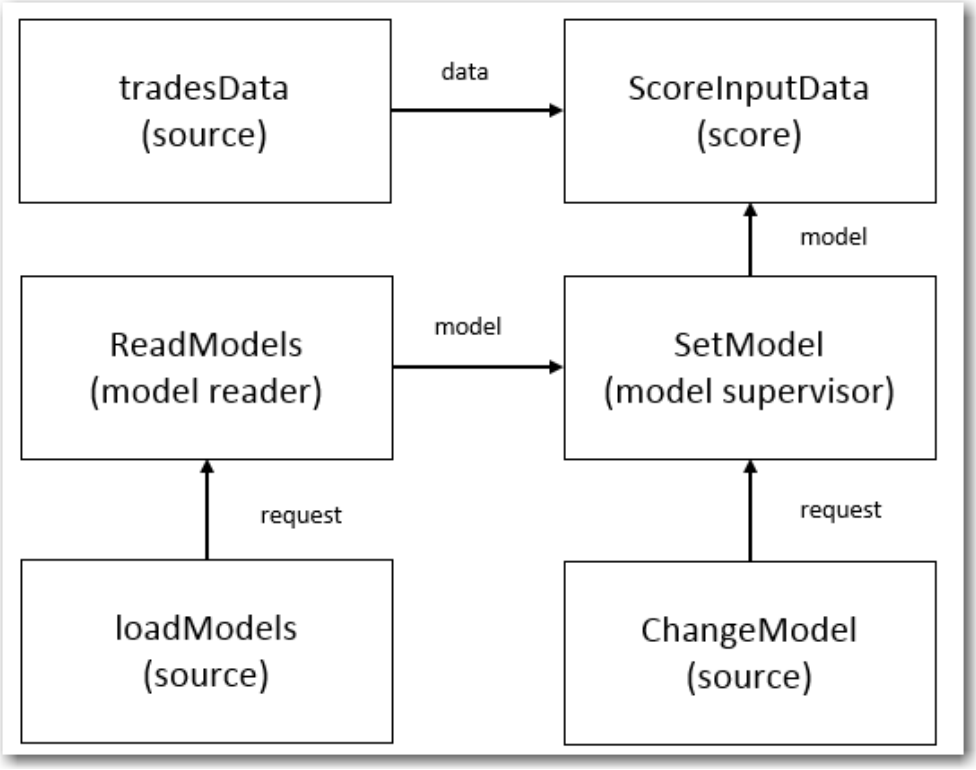
The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<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>
```

Example: Deploying Models through the Model Supervisor Window

You can use the Model Supervisor window to manage offline projects. Through various request events, you can control what model to deploy and when to deploy it.

Consider the following continuous query:



Here are the edges that connect the windows of that query:

```
<edges>
    <edge source="tradesData" target="ScoreInputData" role="data"/>
    <edge source="loadModels" target="ReadModels" role="request"/>
    <edge source="ChangeModel" target="SetModel" role="request"/>
    <edge source="ReadModels" target="SetModel" role="model"/>
    <edge source="SetModel" target="ScoreInputData" role="model"/>
 </edges>
```

Note the following:

- There are three Source windows: one that accepts data to score (tradesData), one that accepts models (loadModels), and one that accepts requests to deploy models (ChangeModel).
- The tradesData Source window streams data to the ScoreInputData Score window to be scored.
- The loadModels Source window streams request events to the ReadModels Model Reader window.
- The ChangeModel Source window streams request events to the SetModel Model Supervisor window.
- The ReadModels Model Reader window streams model events to the SetModel Model Supervisor window.
- The SetModel Model Supervisor window streams model events to the ScoreInputData Score window to use.

Here is the XML code for loadModels, which accepts incoming data through a file–and–socket connector. It reads a CSV file named modelRequest2 for incoming events.

```
<window-source pubsub="true" name="loadModels">
      <schema>
        <fields>
          <field name="req_id" type="int64" key="true"/>
          <field name="req_key" type="string" key="false"/>
          <field name="req_value" type="string" key="false"/>
        </fields>
      </schema>
```

```
    <connectors>
      <connector class="fs" name="read_loadModelsConnector">
        <properties>
          <property name="type"><![CDATA[pub]]></property>
          <property name="fsname"><![CDATA[modelRequest2.csv]]></property> 1
          <property name="fstype"><![CDATA[csv]]></property>
        </properties>
      </connector>
    </connectors>
  </window-source>
```

1 You must specify the full path of the CSV file for the window to find it.

Suppose that the file modelRequest2.csv contains records for the following eight events:

```
i,n,1,action,load 1
i,n,2,type,astore 2
i,n,3,reference,score.sasast 3
i,n,4,, 4
i,n,5,action,load 5
i,n,6,type,astore 6
i,n,7,reference,score2.sasast 7
i,n,8,, 8
```

1 All events in this sequence are Insert (Normal). The **`action`** is to **`load`** an offline ASTORE file into the engine.

2 The **`type`** of model to load is ASTORE.

3 The **`reference`** specifies the name of the ASTORE file to load, which is **`score.sasast`**.

4 This event contains an empty `req_key`

5 The **`action`** is to **`load`** an offline ASTORE file.

6 The **`type`** of model to load is ASTORE.

7 The **`reference`** specifies to load **`score2.sasast`**.

8 This event contains an empty `req_key`.

The models in the two ASTORE files stream into the ReadModels Model Reader window through a file and socket connector.

```
<window-model-reader name="ReadModels">
    <connectors>
      <connector class="fs" name="write_ReadModels_connector">
        <properties>
          <property name="type"><![CDATA[sub]]></property>
          <property name="snapshot"><![CDATA[true]]></property>
          <property name="fsname"><![CDATA[readModels.out]]></property> 1
          <property name="fstype"><![CDATA[csv]]></property>
        </properties>
      </connector>
    </connectors>
  </window-model-reader>
```

1 You must specify the full path of readModels.out, which is the output file of the Model Reader window.

Here is the code for the ChangeModel Source window:

```
<window-source pubsub="true" name="ChangeModel">
    <schema>
```

```
        <fields>
          <field name="req_id" type="int64" key="true"/>
          <field name="req_key" type="string" key="false"/>
          <field name="req_value" type="string" key="false"/>
        </fields>
      </schema>
    </window-source>
```

Suppose that you inject the following set of events into ChangeModel:

```
p,n,1,action,send 1
p,n,2,modelId,0 2
p,n,3,target,ScoreInputData 3
p,n,4,, 4
```

1 All events in this sequence are Upsert (Normal). The **`action`** is to send a model to a downstream window.

2 The modelID of the model to be used is 0.

3 The target window is **`ScoreInputData`**.

4 This event contains an empty `req_key`.

These events flow to SetModel, the Model Supervisor window:

```
<window-model-supervisor name="SetModel" deployment-policy="on-demand">
      <connectors>
        <connector name="write_SetModel_connector" class="fs">
          <properties>
            <property name="type"><![CDATA[sub]]></property>
            <property name="snapshot"><![CDATA[true]]></property>
            <property name="fsname"><![CDATA[setModel.out]]></property> 1
            <property name="fstype"><![CDATA[csv]]></property>
          </properties>
        </connector>
      </connectors>
    </window-model-supervisor>
```

1 You must set the full path to setModel.out, which is the output file of the Model Supervisor window.

Note: You could have used the file and socket adapter rather than the connector to stream models and data into these windows. In either case, the important point to remember is that you must load the analytic models before you load data to be scored.

Here is the code for the Score window. It can use one of two models to score input data: **`score`** or **`score2`**.

```
<window-score name="ScoreInputData">
      <schema>
        <fields>
          <field name="ID" type="int32" key="true"/>
          <field name="P_price" type="double" key="false"/>
          <field name="_WARN_" type="string" key="false"/>
        </fields>
      </schema>
      <models>
        <offline type="astore" reference="score.sasast">
          <output-map>
            <properties>
              <property name="P_price"><![CDATA[P_price]]></property>
              <property name="_WARN_"><![CDATA[_WARN_]]></property>
            </properties>
          </output-map>
```

```
        </offline>
        <offline type="astore" reference="score2.sasast"/>
      </models>
      <connectors>
        <connector name="write_SoreInputData_connector" class="fs">
          <properties>
            <property name="type"><![CDATA[sub]]></property>
            <property name="snapshot"><![CDATA[true]]></property>
            <property name="fsname"><![CDATA[scoreInputData.out]]></property> 1
            <property name="fstype"><![CDATA[csv]]></property>
          </properties>
        </connector>
      </connectors>
    </window-score>
```

1 You must set the full path of scoreInputData.out.

After you load these models, suppose that you inject trades data into tradesData. Events flow from that window into the Score window to be scored. Because of the Model Supervisor window's direction, ScoreInputData uses **score** (ID 0).

```
<window-source pubsub="true" insert-only="true" name="tradesData">
      <schema>
        <fields>
          <field name="ID" type="int32" key="true"/>
          <field name="symbol" type="string"/>
          <field name="currency" type="int32"/>
          <field name="udate" type="int64"/>
          <field name="msecs" type="int32"/>
          <field name="price" type="double"/>
          <field name="quantity" type="int32"/>
          <field name="venue" type="int32"/>
          <field name="broker" type="int32"/>
          <field name="buyer" type="int32"/>
          <field name="seller" type="int32"/>
          <field name="buysellflg" type="int32"/>
          <field name="time" type="stamp"/>
        </fields>
      </schema>
    </window-source>
```

When you inject the following set of events into ChangeModel, which flows into the Model Supervisor window, it directs the Score window to use **score2** (ID 1).

```
p,n,5,action,send
p,n,6,modelId,1
p,n,7,target,ScoreInputData
p,n,8,,
```