SAS® Event Stream Processing 6.2: Using Streaming Analytics

Overview

SAS Event Stream Processing Analytics enables you to use advanced analytical algorithms and machine learning techniques within an event stream processing project. Streaming analytics enables you to address common challenges with data from the Internet of Things (IoT):

- lots of disparate variables
- noisy or missing data
- redundancy in the data
- prediction of rare events

Common use cases for streaming analytics include the following:

- preprocessing, transforming, or filtering data — determining how much and what data to send from the edge to the data center
- detecting anomalies
- monitoring system stability or degradation
- processing unstructured text, audio, video, or image data in order to discern patterns or trends

Streaming Analytics Window Types

Overview

Use the following window types within your event stream processing project to implement streaming analytics:

Table 1 *Streaming Analytics Window Types*

Window Type	Description
Score	Score events with online analytical algorithms that are packaged with SAS Event Stream Processing or with algorithms in offline models.
Train	Refines the algorithm parameters of online models based on streaming event data.
Calculate	Transform data events using a variety of analytical algorithms.
Model Reader	Read models brought into SAS Event Stream Processing as analytic store files or as a combination of non-binary files.
Model Supervisor	Manage models received from Model Reader windows.

When you execute a streaming analytical algorithm within a Score, Train, or Calculate window, you must specify its name, parameter properties, 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. For more information, see "Using the RESTful API" in *SAS Event Stream Processing: Using the ESP Server*.

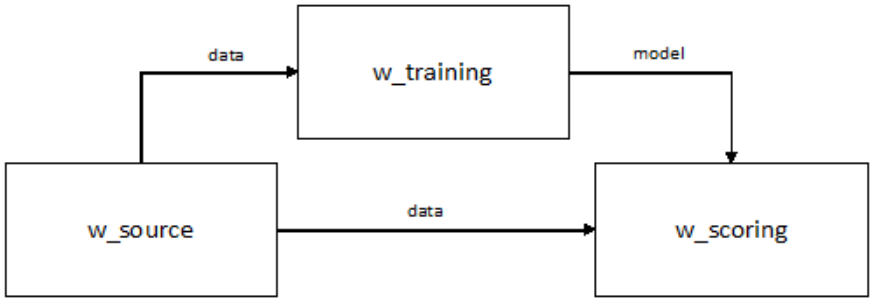
Edge Roles

Edges to or from any of the streaming analytics window types must specify a role that corresponds to the event type.

Table 2 *Edge Roles*

Role	Description
data	Sends data events between windows. A data event streams data to be processed by a streaming 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:

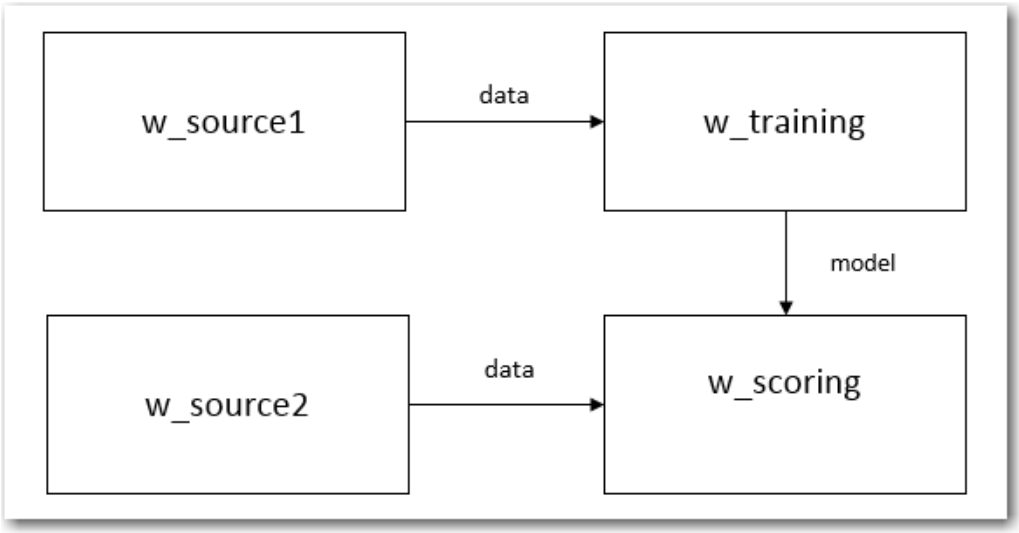


The following code specifies roles for the edges between the windows:

```
<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 Train window sends a refined model to the Score window. 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 streaming analytics window type and what properties you set to use them, use the **dfesp_analytics** command-line utility. The utility is located in **$DFESP_HOME/bin** in a UNIX environment and **%DFESP_HOME%\bin** in a Windows environment.

Table 3 *Analytics Utility Commands*

Command	Result
dfesp_analytics	Returns a list of the utility's available options
dfesp_analytics –*window_type*	Lists the algorithms available to the specified *window_type*: **train**, **score**, or **calculate**.
dfesp_analytics –reader –modelType recommender | astore	Lists the properties that you can set within a Model Reader window for a recommender system or for a model provided in an analytic store file.
dfesp_analytics*window* –algorithm *algorithm*	For a specified *algorithm*, lists a window type's properties in text format.

Command	Result
dfesp_analytics -*window* –algorithm *algorithm* –xml	For a specified *algorithm*, lists a window type's properties in XML format.
dfesp_analytics -score –type recommender	For **recommender**, lists the input-map and output-map of the model.
dfesp_analytics -score –type astore –reference *astore_file*	For a specified *astore_file*, lists the input-map and output-map of the trained model stored as a binary file with the analytic store format.
dfesp_analytics -score –type astore –reference *astore_file* –options *option(s)*	Set the specified *option(s)* for the trained model in the specified *astore_file*. It then lists the input-map and output-map associated with those specified options. You can specify a comma-separated list of key-value pairs for *options(s)*: **-options *key1=value1:key2=value2...***

For example, when you submit **$DFESP_HOME/bin/dfesp_analytics –train** on the UNIX command line, you obtain the following output:

```
train algorithms:
        DBSCAN
        KMEANS
        LinearRegression
        SVM
        LogisticRegression
```

When you submit **$DFESP_HOME/bin/dfesp_analytics –train –algorithm DBSCAN** on the UNIX command line, you obtain the following output:

```
parameters:
        epsilon: double:  (3.0)
        mu: int64:  (4)
        beta: double:  (0.3)
        lambda: double:  (0.02)
        recluster: boolean:  (1)
        reclusterFactor: double:  (2.0)
        nInit: int64:  (50)
        velocity: int64:  (1)
        commitInterval: int64:  (25)
input-map:
        inputs: varlist: double ()
```

When you submit **$DFESP_HOME/bin/dfesp_analytics –train –algorithm DBSCAN –xml** on the UNIX command line, you obtain the following output:

```
<parameters>
    <properties>
        <property name='epsilon'>3.0</property>
        <property name='mu'>4</property>
        <property name='beta'>0.3</property>
        <property name='lambda'>0.02</property>
        <property name='recluster'>1</property>
        <property name='reclusterFactor'>2.0</property>
        <property name='nInit'>50</property>
        <property name='velocity'>1</property>
        <property name='commitInterval'>25</property>
    </properties>
</parameters>
<input-map>
    <properties>
        <property name='inputs'></property>
    </properties>
</input-map>
```

Suppose that you have created an analytic store file named **svddsstate.sasast** that contains a trained Support Vector Data Description (SVDD) model. When you submit **$DFESP_HOME/bin/dfesp_analytics –score –type astore –reference svddstate.sasast** on the UNIX command line from the directory that contains the analytic store file, you obtain output that looks like this:

```
input-map:
        x1: double
        x2: double
        x3: double
        x4: double
        x5: double
        x6: double
        x7: double
        x8: double
        x9: double
        x10: double
        x11: double
        x12: double
        x13: double
        x14: double
        x15: double
        x16: double
        x17: double
        x18: double
        x19: double
        x20: double
        x21: double
        x22: double
        x23: double
output-map:
        __SVDDdistance__: double (SVDD Distance)
        __SVDDscore__: double (Label)
```

You can use information from the input map to help you code the input schema of the Source window that streams data to be scored. You can use information from the output map to help you code the schema for the Score window.

When you submit **$DFESP_HOME/bin/dfesp_analytics –reader –modelType recommender** on the UNIX command line, you obtain this type of output:

```
parameters:
        n: int32:  (-1)
        method: string: (RMF)
        itemTable: string:  ()
        itemTableDelimiter: string:  (COMMA)
        itemTableLineBreak: string:  (LF)
        userTable: string:  ()
        userTableDelimiter: string:  (COMMA)
        userTableLineBreak: string:  (LF)
        userRateInfo: string:  ()
        userRateInfoDelimiter: string:  (COMMA)
        userRateInfoLineBreak: string:  (LF)
        itemRateInfo: string:  ()
        itemRateInfoDelimiter: string:  (COMMA)
        itemRateInfoLineBreak: string:  (LF)
        ratingsByUser: string:  ()
        ratingsByUserDelimiter: string:  (COMMA)
        ratingsByUserLineBreak: string:  (LF)
        similarUsers: string:  ()
        similarUsersDelimiter: string:  (COMMA)
        similarUsersLineBreak: string:  (LF)
```

These are the properties that you can set for recommender scoring within the Model Reader window.

When you submit **$DFESP_HOME/bin/dfesp_analytics –reader –modelType astore** on the UNIX command line, you obtain this type of output:

```
parameters:
        reference: string:  ()
```

The `reference` property specifies the name and location of the analytic store file to use. No other parameters are listed in this output because those parameters cannot be predicted. They are specific to the model contained in the analytic store file.

Assume that you have an analytic store file named **rpca.sasast** that contains a trained Robust Principal Components Analysis model. The following command sets projection into the low rank space for the RPCA algorithm.

```
dfesp_analytics -score -type astore -reference rpca.sasast -options
RPCA_PROJECTION_TYPE=1
```

Using Score Windows

Score windows accept model events to make predictions for incoming data events. They generate 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.)

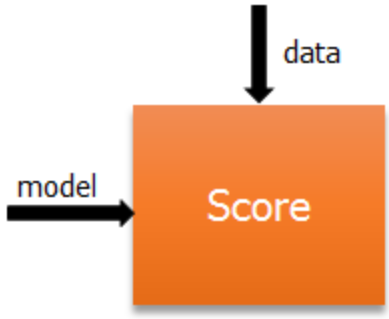


Table 4 *Properties of window-score element*

Property	Required or Optional?	Description
`name`	Required	Specifies the window name. It must start with one of the following characters: _, a-z, A-Z. The rest of the name can include the following characters: _, a-z, A-Z, 0-9.
`model-type`	Optional	Specifies the type of model to read and pass to a Score window. Valid options are **`astore`** and **`recommender`**, for analytic store and Recommender System offline models, respectively.
`pubsub`	Optional	Specifies whether to publish and subscribe to the window. When the project-level value of `pubsub` is **`manual`**, a value of **`true`** enables publishing and subscribing and a value of **`false`** disables it.

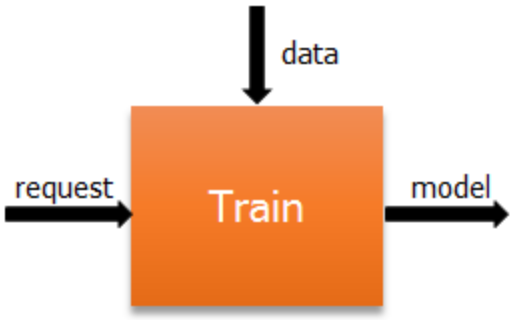
The following is a generic example of a Score window element:

```
<window-score name='w_score'>
     <schema>
       <fields>
        ...
       </fields>
     </schema>
     <models>
      ...
     </models>
   </window-score>
```

Using Train Windows

Train windows receive data events and publish model events to Score windows. They use incoming data events to develop and adjust model parameters 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.

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



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

Table 5 *Properties of the window-train element*

Property	Required or Optional?	Description
`name`	Required	Specifies the window name. It must start with one of the following characters: _, a-z, A-Z. The rest of the name can include the following characters: _, a-z, A-Z, 0-9.
`algorithm`	Required	Specifies the algorithm for the window. If the project is online, the algorithm must be specified with its short name.
`pubsub`	Optional	Specifies whether to publish and subscribe to the window. When the project-level value of `pubsub` is **`manual`**, a value of **`true`** enables publishing and subscribing and a value of **`false`** disables it.

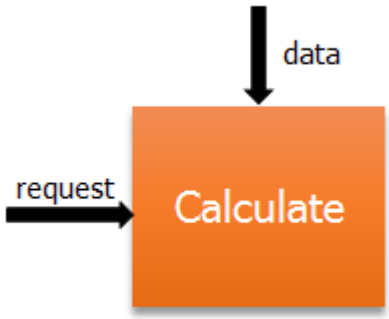
The following is a generic example of a Train window element:

```
<window-train name='w_train' algorithm='algorithm_short_name'>
     <parameters>
       <properties>
        ...
       </properties>
     </parameters>
     <input-map>
      ...
     </input-map>
   </window-train>
```

Using Calculate Windows

Overview to Calculate Windows

Calculate windows create real time, running statistics that are based on established analytical techniques. They receive data events and publish newly transformed score data into output events. (No role is assigned to the outgoing edges, so those edges do not appear in the diagram.) Calculate windows can also receive request events.



Calculate windows are designed for data normalization and transformation methods, as well as for learning models that bundle training and scoring together.

Table 6 *Properties of window-calculate element*

Property	Required or Optional?	Description
`name`	Required	Specifies the window name. It must start with one of the following characters: _, a-z, A-Z. The rest of the name can include the following characters: _, a-z, A-Z, 0-9.
`algorithm`	Required	Specifies the algorithm for the window. If the project is online, the algorithm must be specified with its short name.
`collapse-updates`	Optional	If **`true`**, multiple update blocks are collapsed into a single update block.
`exp-max-string`	Optional	Specifies the maximum size of strings that the expression

Property	Required or Optional?	Description
		engine uses for the window. The default value is `1024`.
`index`	Optional	Specifies the index type for the window. Valid options are **`'pi_RBTREE'`**, **`'pi_HASH'`**, **`'pi_LN_HASH'`**, **`'pi_CL_HASH'`**, **`'pi_FW_HASH'`**, **`'pi_EMPTY'`**, **`'pi_HLEVELDB'`**, or **`'pi_HLEVELDB_NC'`**.
`output-insert-only`	Optional	If **`true`**, prevents the window from passing non-insert events to other windows.
`pubsub`	Optional	Specifies whether to publish and subscribe to the window. When the project-level value of `pubsub` is **`manual`**, a value of **`true`** enables publishing and subscribing and a value of **`false`** disables it.
`pubsub-index`	Optional	Specifies the publish/subscribe index value. Valid options are **`'pi_RBTREE'`**, **`'pi_HASH'`**, **`'pi_LN_HASH'`**, **`'pi_CL_HASH'`**, **`'pi_FW_HASH'`**, **`'pi_EMPTY'`**, **`'pi_HLEVELDB'`**, or **`'pi_HLEVELDB_NC'`**.
`pulse-interval='interval (unit)'`	Optional	Specifies the interval at which to write a canonical batch of updates. Pass an integer for the `interval`. Valid options for the `unit` are **`microseconds`**, **`millseconds`**, **`seconds`**, **`minutes`**, **`hours`**, or **`days`**. The default is **`milliseconds`**.

The following is a generic example of a Calculate window element:

```
<window-calculate name='w_calculate' algorithm='algorithm_short_name'>
          <schema>
               <fields>
                  ...
               </fields>
            </schema>
            <parameters>
              <properties>
                ...
              </properties>
```

```
    </parameters>
    <input-map>
     ...
    </input-map>
    <output-map>
      ...
    </output-map>
    <connectors>
    ...
    </connectors>
  </window-calculate>
```

You can define a SAS Micro Analytic Service module to specify a block of code to execute within a Calculate window. The block of code can contain one or more functions. You can write this block of code in Python or DS2. You must specify a SAS Micro Analytic Service map within the Calculate window to bind functions to a Source window, which receives the input data. For more information, see “Working with SAS Micro Analytic Service Modules” .

Migrating from a Procedural Window to a Calculate Window

Support for SAS Micro Analytic Service (MAS) modules and stores has moved from the Procedural window to the Calculate window.

To migrate from a Procedural Window to a Calculate Window requires minimal change to your XML code.

Example Code 1 *Procedural Window*

```
<window-procedural name='pw_01'>
                …
</window-procedural>
<edge source='w_source' target='pw_01'/>
```

Example Code 2 *Calculate Window*

```
<window-calculate name='pw_01' algorithm='MAS'>
              …
</window-calculate>
<edge source='w_source' target='pw_01' role='data'/>
```

You can use the **`dfesp_xml_migrate`** command to convert Procedural windows to Calculate windows in your model code. However, you must manually convert DS2 code in table server mode to code that uses SAS Micro Analytic Service modules before running the tool. For more information about this command, see “Migrating XML Code across Product Releases” in *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

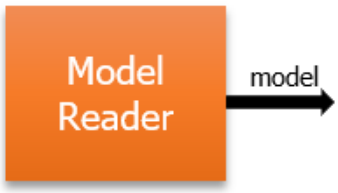
Using Model Reader Windows

In most cases, *Model Reader windows* receive request events that include the location and type of an offline model. Offline models are specified, developed, trained, and stored separately from the ESP

server. Model Reader windows publish a model event that contains the model to Score windows or to Model Supervisor windows.

For more information, see “Online Scoring and Training Using Offline Models”.

For recommender systems, you can specify offline model property values within the Model Reader window itself in addition to the request method. The window publishes a model event based on those values.



For more information, see “Using Recommender Systems”.

Table 7 *Properties of the window-model-reader element*

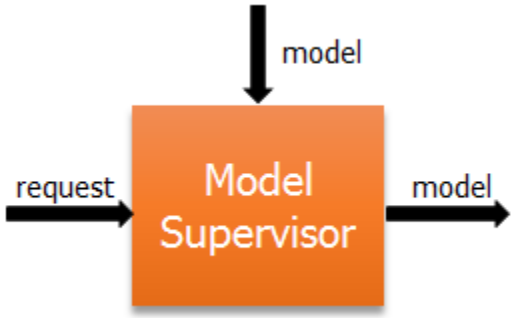
Property	Required or Optional?	Description
`name`	Required	Specifies the window name. It must start with one of the following characters: _, a-z, A-Z. The rest of the name can include the following characters: _, a-z, A-Z, 0-9.
`model-type`	Optional	Specifies the type of model to read and pass to a Score window. Valid options are **`astore`** and **`recommender`**, for analytic store and Recommender System offline models, respectively.
`pubsub`	Optional	Specifies whether to publish and subscribe to the window. When the project-level value of `pubsub` is **`manual`**, a value of **`true`** enables publishing and subscribing and a value of **`false`** disables it.

The following is a generic example of a Model Reader window element:

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

Using Model Supervisor Windows

Model Supervisor windows manage the flow of model events. Through input request events, you can control what model to deploy and when and where to deploy it. Model events are published to Score windows.



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.

Table 8 *Properties of window-model-supervisor element*

Property	Required or Optional?	Description
`name`	Required	Specifies the window name. It must start with one of the following characters: _, a-z, A-Z. The rest of the name can include the following characters: _, a-z, A-Z, 0-9.
`deployment-policy`	Optional	Specifies the deployment policy of the offline models. Valid options are **`immediate`**, which sends model events to any receiving window immediately after receiving item; and **`on-demand`**, which sends model events according to the requests specified at the command line.
`capacity`	Optional	Specifies the maximum number of offline models to allow. After the capacity is reached, older model events are discarded.
`pubsub`	Optional	Specifies whether to publish and subscribe to the window. When the project-level value of `pubsub` is **`manual`**, a value of **`true`** enables publishing and subscribing and a value of **`false`** disables it.

The following is a generic example of a Model Supervisor window element:

```
<window-model-supervisor name='w_supervisor' deployment-policy='on-demand'
capacity='1000'>
     <connectors>
```

```
        ...
        </connectors>
    </window-model-supervisor>
```

Understanding 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.

- A Train window adjusts the parameters of a model based on incoming data events. A Train window then streams the outcome of those changes through model events into a Score window.
- A Model Reader window publishes models to Score windows or Model Supervisor windows as specified by a request event.
- The Model Supervisor window controls how and when models are deployed to Score windows as specified by a request event.

You do not explicitly specify fixed model event schema; it is implicit for the query:

Table 9 *Fields in the Model Event Schema*

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.
`timestamp`	Specifies the timestamp when the model was first created.
`model_perf`	Specifies the performance metric of a model.

For example, with k-means clustering, a Train window receives data streams and computes and adjusts the centroids of the clusters. When the Score window receives the model event, it recovers the centroid information and applies it to the model. The Score window uses the centroid information to find the closest centroid for each of its incoming events, and assigns the cluster label accordingly.

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:

Table 10 *Actions Performed by Request Events*

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 analytic store 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 analytic store 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,,
```

Processing Loaded Analytic Store Files with GPUs

To process a loaded analytic store file with a graphical processing unit (GPU), specify events before the `load` request that use special keywords.

```
i,n,1,USEGPUESP,1
i,n,2,NDEVICES,1
i,n,3,DEVICE0,1
i,n,4,"action","load"
i,n,5,"type","astore"
i,n,6,"reference","YOUR_ASTORE_FILE"
i,n,7,,
```

The following keywords control GPU usage.

Table 11 *Keywords for GPU Usage*

Keyword	Description
`USEGPUESP`	Tell the ESP server to use GPUs to perform the calculation. GPUs must physically reside on the same computer system running the ESP server. Specify `1` to use GPUs, `0` not to use GPUs.
`TENSORRTESP`	Tell the ESP server to enable GPUs using Nvidia's high-performance TensorRT engine. When you specify this keyword, the analytic store mode is deployed in TensorRT format. After initially analyzing the network architecture, an optimized version of the network is deployed to score with the GPUs. **Note:** Use `TENSORRTESP` instead of `USEGPUESP` only on Nvidia Jetson TX2 Modules.
`NDEVICES`	Specify how many GPUs to use to perform the calculation. When you do not specify `NDEVICES`, the ESP server uses all GPUs on the system. **IMPORTANT** It is recommended to assign a separate GPU per Score window to avoid contention. **Note:** If your project uses multiple deep learning models, then you should distribute each model to a separate GPU.

`DEVICE`*n*	Use with `NDEVICES`. Specify which GPU to use to perform the calculation. When using multiple GPUs, specify a list starting from 0.
	For example, suppose that your system has eight GPUs. You want to use two of them, with device IDs 3 and 5.
	`i,n,4,USEGPUESP,1`
	`i,n,5,NDEVICES,2`
	`i,n,6,DEVICE0,3`
	`1,n,7,DEVICE1,5`
	`1,n,8,"action","load"`
	`...`

Note: Only deep learning models support the use of GPUs.

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,,
```

Working with SAS Micro Analytic Service Modules

Overview

SAS Micro Analytic Service is a memory-resident, high-performance program execution service. A SAS Micro Analytic Service module is essentially a named block of code that you execute within a SAS Event Stream Processing model. This block of code, which you define at the project level, can contain one or more functions. You can write functions in Python or DS2.

To use Python functions within SAS Micro Analytic modules, you must set two environment variables:

- MAS_M2PATH is the full pathname to a required Python program, `mas2py.py`
- MAS_PYPATH is the full pathname to the desired Python executable

For example:

- `export MAS_M2PATH=/opt/sas/viya/home/SASFoundation/misc/embscoreeng/mas2py.py`
- `export MAS_PYPATH=/usr/bin/python`

For more information about SAS Micro Analytic Service, see *SAS Micro Analytic Service: Programming and Administration Guide*.

Defining a SAS Micro Analytic Service Map

You define a SAS Micro Analytic Service map in a Calculate window to bind a function to a Source window. This binding acts as the input handler for the Calculate window.

Consider the following example that calculates the total cost of a stock transaction:

```
<project name='trades_proj' pubsub='auto' threads='4'>
  <mas-modules>
     <mas-module language="python" module="module1" func-names='compTotal'> <!--1-->
        <code>
         def compTotal(quantity, price):<!--2-->
```

```
            total= quantity * price
            return total
         </code>
      </mas-module>
   </mas-modules>
   <contqueries>
      <contquery name='cq'>
         <windows>
  …
          <window-calculate name='pw1' algorithm='MAS'>
            <schema-string><!--3-->
ID*:string,security:string,quantity:double,
             price:double,total:double</schema-string>
            <mas-map>
                <window-map module="module1" revision="0"
                 source="src_win" function="compTotal"/><!--4-->
            </mas-map>
          </window-calculate>

         <edges>
            <edge source='src_win' target='pw1' role='data'/><!--5-->
         </edges>
      </contquery>
   </contqueries>
</project>
```

1 SAS Micro Analytic Service module named `module1` is defined at the project level of the `trades_proj` project. It contains a function `compTotal` written in Python.

2 The Python code for `compTotal` is specified within the `<code>` element. This function acts as the input handler for all events passed from the `src_win` Source window to the `pw1` Calculate window.

3 The schema string specifies the fields that define the structure of incoming events. Data relevant to the security being traded, the quantity of shares, and the current prices are streamed into the Calculate window.

4 At the map level of the Calculate window, the window map binds the `compTotal` function of `module1` to the Source window.

5 The edge between the Source window and the Calculate window must have the role `data`.

A Calculate window can have only one SAS Micro Analytic Service module map. The MAS map can include one or more window maps.

For example, in the following code, two MAS modules containing DS2 functions are defined at the project level:

```
<mas-modules>
   <mas-module language="ds2" module="module_1" func-names='compute_volume'>
       <code>
         <![CDATA[
               ds2_options sas;
               package myModule_1/overwrite=yes;
               method compute_volume(int quantity, double price, in_out int volume);
               volume = quantity * price;
               end;
               endpackage;
          ]]>
       </code>
```

```
    </mas-module>
    <mas-module language="ds2" module="module_2" func-names='compute_price_sqr'>
       <code>
         <![CDATA[
               ds2_options sas;
               package myModule_2/overwrite=yes;
               method compute_price_sqr(int quantity, double price, in_out double
price_sqr);
               price_sqr = price * price;
               end;
               endpackage;
          ]]>
       </code>
    </mas-module>
</mas-modules>
```

The ESP server loads the MAS modules in the order in which they appear in the XML code.

Two window maps within a MAS map bind the DS2 functions to the Calculate window:

```
...
<mas-map>
   <window-map module="module_1" revision="0" source="Trades1"
function="compute_volume"/>
   <window-map module="module_2" revision="0" source="Trades2"
function="compute_price_sqr"/>
</mas-map>
...
```

Window maps have four attributes:

- `module` refers to the name of a previously defined module
- `revision` must be set to 0
- `source` is the name of the Source window for which you specify the handler
- `function` specifies the function defined in the module

For information about the XML elements that you use to define a SAS Micro Analytic Service module, see *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

Generating Multiple Derived Events

A SAS Micro Analytic Service module method can write one or more arrays. You can generate multiple derived events when at least one of the output array names matches a derived event field name.

The number of events generated is determined by the number of elements in the matching arrays at run time.

SAS Event Stream Processing requires every event to have a unique key. Therefore, a source event's key cannot be duplicated in multiple generated derived events. When the key is duplicated, SAS Event Stream Processing halts processing and returns an error. When you map a SAS Micro Analytic Service module method to a Calculate window, ensure that each derived event has a unique key value. For more information, see the *SAS Micro Analytic Service: Programming and Administration Guide*.

Consider the following input event:

```
i,n,1,field1,field2,field3
```

You can use the following SAS Micro Analytic Service module within a Calculate window to transpose the input event:

```
<mas-modules>
  <mas-module language="ds2" module="module_1" func-names='test_function'>
    <code>
      <![CDATA[
             ds2_options sas;
             package test_package;
               method test_function(nchar(50) field1, nchar(50) field2,
                                    nchar(50) field3, in_out nchar(50)
                                    stringD[3]);
                 stringD[1] = trim(field1);
                 stringD[2] = trim(field2);
                 stringD[3] = trim(field3);
               end;
             endpackage;
          ]]>
    </code>
  </mas-module>
</mas-modules>
```

The resulting output events would be as follows:

```
i,n,1,1,field1
i,n,1,2,field2
i,n,1,3,field3
```

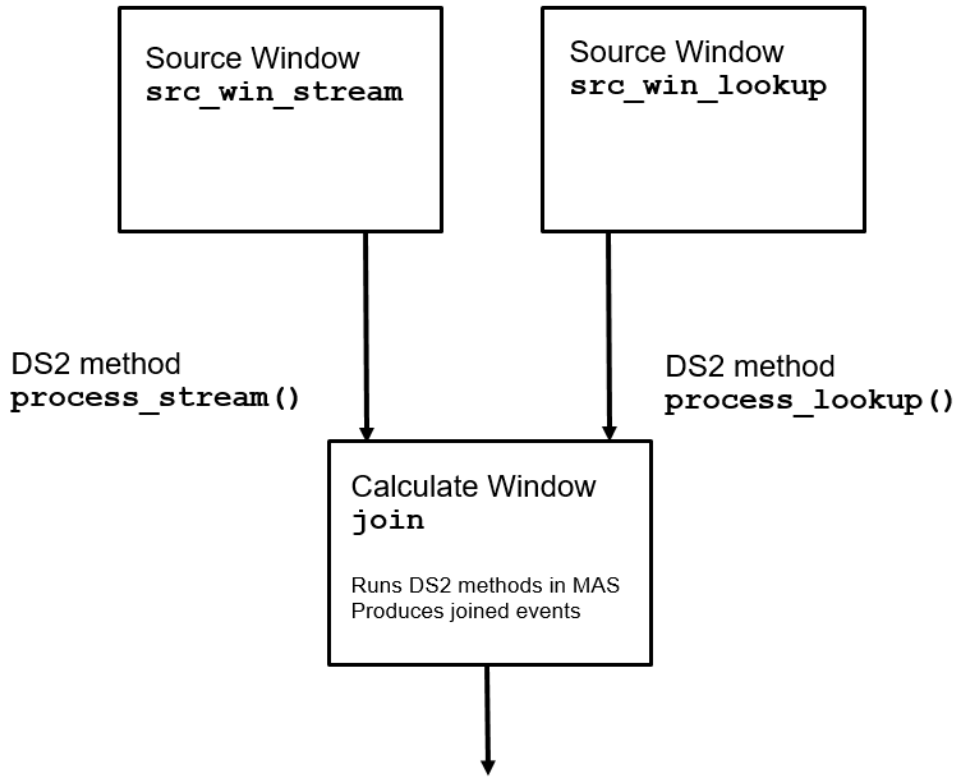
Using Shared Vectors and Shared Hash Table Data Structures

SAS Micro Analytic Service provides two ways to share data between the modules executing within a user context: shared vectors and shared hash tables. *Shared vectors* are collections of data values. The values within a vector can have a mix of data types. *Shared hash tables* are containers of vectors, where the vectors are stored and accessed by using keys. Both of these structures are thread–safe and lock-free. You can use them to share data across DS2 methods within a SAS Event Stream Processing project.

For more information about these data structures, see *SAS Micro Analytic Service: Programming and Administration Guide*

The following query shows two Source windows streaming events into a Calculate window. Two DS2 methods that use a shared hash table run within a SAS Micro Analytic Service (MAS) module. The MAS module simulates a no-regenerate inner-join. The module runs within a Calculate window and produces joined events.

Figure 1 *Continuous Query with a SAS Micro Analytic Service Module Sharing a Hash Table*



Here is the code for the MAS module:

```
<mas-modules>
   <mas-module language="ds2" module="module_1"
               func-names='process_lookup,process_stream'>
     <code>
       <![CDATA[
          ds2_options sas;
            package module_1/overwrite=yes;

            dcl package masstate /*1*/ st();
            dcl package logger logr('DF.ESP');

            method process_stream /*2*/(varchar(16) _inOpcode, bigint lookupID,
                  in_out varchar matchString,
                  in_out varchar matchDescription,
                  in_out varchar _outOpcode) ;
               dcl int rc;
               dcl varchar(32) key;

               key = lookupID;
               _outOpcode='noop'; /*3*/

               /* logr.log('d', 'key=$s', key); */
               if (0 eq st.get(key)) then do;
                   matchString=st.getString(key,0);
                   matchDescription=st.getString(key,1);
                   _outOpcode=_inOpcode; /*4*/
               end;
```

```
                /* logr.log('d', 'inOpcode=$s, outOpcode=$s', _inOpcode, _outOpcode);
*//*5*/
            end;

            method process_lookup /*6*/(bigint ID, varchar(32) value,
                   varchar(4096) description,
                   in_out varchar _outOpcode);
               dcl int rc;
               dcl varchar(32) key;

               _outOpcode='noop';
               key = ID;
               rc = st.createVector(key,2);  /*7 */
               rc = st.setString(key, 0, value); /*8*/
               rc = st.setString(key, 1, description);
               rc = st.put(key); /*9*/

               rc = st.deleteVector(key); /*10*/
            end;
          endpackage;
          ]]>
        </code>
   </mas-module>
</mas-modules>
```

1 When using a shared hash table, you use the package MASSTATE to create, share, retrieve, and delete data.

2 The method `process_stream` is designed to insert received events into a shared hash table. It can also handle Update and Delete events by looking at the `_inOpcode`.

3 Default to **`noop`**. That is, produce no event.

4 Set the `Opcode` upon a match, producing an event.

5 Uncomment this line to write debugging messages to the log.

6 The method `process_lookup` is designed to look up a string through a lookup `ID` in a shared hash table. If no match is found, then the `_outOpcode` is **`noop`**, so no event is generated.

  The method as presented handles Insert events. You could modify it to examine the `_inOpcode` and add, update, or delete an entry from the shared hash table.

7 Create a new vector.

8 Insert the events data.

9 Put the vector to the hash.

10 Delete the vector.

Here is the XML code for two Source windows:

```
<window-source name='src_win_lookup' index='pi_RBTREE'>
      <schema-string>ID*:int64,value:string,description:string</schema-string>
      <connectors>
         <connector class='fs' name='pub'>
            <properties>
               <property name='type'>pub</property>
               <property name='fstype'>csv</property>
               <property name='fsname'>lookup.csv</property>
            </properties>
```

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

<window-source name='src_win_stream' index='pi_RBTREE'>
    <schema-string>ID*:int64, lookupID:int64, price:double, quant:int64</schema-
string>
    <connectors>
       <connector class='fs' name='pub'>
         <properties>
           <property name='type'>pub</property>
           <property name='fstype'>csv</property>
           <property name='fsname'>stream.csv</property>
         </properties>
       </connector>
    </connectors>
</window-source>
```

Here is the XML code for the Calculate window containing the MAS map for the MAS methods:

```
<window-calculate name='join' algorithm='MAS'>
    <schema-string>ID*:int64,matchID:int64,matchString:string,matchDescription:string,
                  price:double,quant:int64</schema-string>
    <mas-map>
      <window-map module="module_1" revision="0" source="src_win_lookup"
function="process_lookup"/>
      <window-map module="module_1" revision="0" source="src_win_stream"
function="process_stream"/>
    </mas-map>
    <connectors>
      <connector class='fs' name='sub'>
        <properties>
          <property name='type'>sub</property>
          <property name='fstype'>csv</property>
          <property name='fsname'>result.csv</property>
          <property name='snapshot'>true</property>
        </properties>
      </connector>
    </connectors>
</window-calculate>
```

Here is the XML code for the edges between the Source windows and the Calculate window:

```
<edges>
   <edge source='src_win_lookup' target='join' role='data'/>
   <edge source='src_win_stream' target='join' role='data'/>
</edges>
```

Here is the XML code for the connector groups. You place this code outside the continuous query that contains the Source and Calculate windows.

```
<project-connectors>
      <connector-groups>
         <connector-group name="group1"> <!--1-->
            <connector-entry connector='cq_01/join/sub' state='running'/>
            <connector-entry connector='cq_01/src_win_lookup/pub' state='finished'/>
         </connector-group>
         <connector-group name="group2">
            <connector-entry connector='cq_01/src_win_stream/pub' state='finished'/>
```

```
            </connector-group>
         </connector-groups>
         <edges>
            <edge source='group1' target='group2'/>
         </edges>
      </project-connectors>
```

1 A *connector group* is a container of connector-entry elements. Connector entries can specify the state of a connector.

Now suppose that you stream the following events into the Source window src_win_lookup:

```
i,n,10001,sunw,"Workstation manufacturer"
i,n,20001,ibm,"From typewriters to mainframes"
```

Then you stream the following events into the Source window src_win_stream:

```
i,n,1,10001,101.45,100
i,n,2,20001,23.5,1000
i,n,3,20001,10.1,80
i,n,4,10001,10.2,85
```

The joined data in XML is as follows:

```
<event opcode='insert' window='project_01/cq_01/join'>
   <value name='ID'>1</value>
   <value name='matchDescription'>Workstation manufacturer</value>
   <value name='matchString'>sunw</value>
   <value name='price'>101.450000</value>
   <value name='quant'>100</value>
 </event>

 <event opcode='insert' window='project_01/cq_01/join'>
   <value name='ID'>2</value>
   <value name='matchDescription'>From typewriters to mainframes</value>
   <value name='matchString'>ibm</value>
   <value name='price'>23.500000</value>
   <value name='quant'>1000</value>
 </event>

 <event opcode='insert' window='project_01/cq_01/join'>
   <value name='ID'>3</value>
   <value name='matchDescription'>From typewriters to mainframes</value>
   <value name='matchString'>ibm</value>
   <value name='price'>10.100000</value>
   <value name='quant'>80</value>
 </event>

 <event opcode='insert' window='project_01/cq_01/join'>
   <value name='ID'>4</value>
   <value name='matchDescription'>Workstation manufacturer</value>
   <value name='matchString'>sunw</value>
   <value name='price'>10.200000</value>
   <value name='quant'>85</value>
 </event>
```

Working with SAS Micro Analytic Service Stores

Overview

A *SAS Micro Analytic Service store* is a named, versioned object repository that is maintained by SAS Event Stream Processing. A store enables convenient access to DS2, C, Python, and analytic store files in a SAS Micro Analytic Service module without requiring direct access to the ESP server's file system. You can use the RESTful API to query, create, delete, and write objects to SAS Micro Analytic Service stores.

You can use a command–line program (`dfesp_mas_stores`) to call the RESTful API directly. You can run this program in one of four modes: query, create, delete, and write.

Query mode:

dfesp_mas_stores -n *hostname* :*port* -q [<-s *store* > [<-v *version* >]]

Create mode:

dfesp_mas_stores -n *hostname* :*port* -c -s *store*

Delete mode:

dfesp_mas_stores -n *hostname* :*port* -d -s *store* <-v *version* >

Write mode:

dfesp_mas_stores -n *hostname* :*port* -w -s *store* -o *object* -f *file* <-v *version* > <-b >

Table 12 *Arguments to dfesp_mas_stores*

Argument	Description
`-n hostname:port`	Specifies the HTTP *hostname* and *port* of the running ESP server
`-q`	Queries and prints stores
`-c`	Creates a store
`-d`	Deletes a store
`-w`	Writes an object to a store
`-s store`	Specifies the name of the *store* to write to, query, create, or delete
`-v version`	Specifies the *version* of the store to write to, query, or delete

Argument	Description
`-o object`	Specifies the name of the *object* to write to the store
`-f file`	Specifies the name of the local *file* that contains the object to write to the store
`-b`	Identifies the content type as binary

Using Multi-Part Modules

Ordinarily, a DS2 package defines a single DS2–based model, either within a `<code>` element or a `<code-file>` element. You can use a DS2 package to set up an analytic store file-based model that uses multiple analytic store files. Define these analytic store files within `<module-member>` XML elements.

```
<mas-modules>
   <mas-module language="ds2" module="module_1" func-names='astoreScore'><!--4-->
     <code-file>/tmp/mas02/masscore.ds2</code-file><!--1-->
     <module-members>
       <module-member member='astore_1'
SHAkey='EB3D1CA20AA0CB74465D25EEE2290E13692AF750' type='astore'>
         <code-file>/tmp/mas02/va_model208</code-file><!--2-->
       </module-member>
       <module-member member='astore_2'
SHAkey='FB3D1CB20CC0CB74465D25EEE2290E13692AF750' type='astore'>
         <code-file>/tmp/mas02/va_model209</code-file><!--3-->
       </module-member>
     </module-members>
   </mas-module>
 </mas-modules>
```

1 The file `masscore.ds2` sets up a DS2 package.

2 The file `va_model208` contains one analytic store model to apply.

3 The file `va_model209` contains a second analytic store model to apply. The DS2 package orchestrates the execution of these analytic store files.

4 When a Calculate window invokes the `aStoreScore` method from `module1` , it can use each of the two analytic store models.

The analytic store files do not need to physically exist on the ESP server file system. They are streamed to the ESP server over the masStore REST interface.

SAS Model Manager serves as a system-of-record repository. It enables you to designate analytical models as champions within a SAS Model Manager project. An ESP project that has been deployed to a server can incorporate a SAS Model Manager project champion model. You can use SAS Event Stream Processing Studio and SAS Event Stream Manager to retrieve and deploy model artifact files (DS2 and analytic store) from SAS Model Manager to the ESP server. If a champion model is subsequently retrained or replaced, the integrated infrastructure delivers updates to the ESP server without requiring you to revise and redeploy the ESP project. For an example, see “Importing Models Created in SAS Model Manager into SAS Event Stream Processing Studio” in *SAS Event Stream Processing: Using SAS Event Stream Processing Studio*.

Note: This functionality is not supported on Microsoft Windows.

Example

Consider the following example:

1 The following command queries an ESP server and returns all the defined stores:

```
dfesp_mas_stores -n host
:port -q
```

```
<mas-stores/>
```

The response shows that no stores currently exist.

2 The following command creates a new SAS Micro Analytic Service store named SJK:

```
dfesp_mas_stores -n host
:port -c -s sjk
```

```
<response>
      <message>mas store sjk successfully created</message>
  </response>
```

3 DS2 code that is contained in a file named `code.sas` is written to an object named `ds2Code` in the newly created store:

```
dfesp_mas_stores -n host
:port -w -s sjk -o ds2Code -f code.sas
```

```
<response>
      <message>mas store model ds2Code (1.0) successfully saved for mas store sjk</message>
  </response>
```

4 DS2 code that is contained in a local file named `code-new.sas` writes to an object named `new-ds2Code` in the store:

```
dfesp_mas_stores -n espsrv01:31415 -w -s sjk -o new-ds2Code -f code-new.sas
```

```
<response>
      <message>mas store model new-ds2Code (1.0) successfully saved for mas store sjk</message>
  </response>
```

5 Python code that is contained in a local file named `code.py` writes to an object named `pythonCode` in the store:

```
dfesp_mas_stores -n <host>:<port> -w -s sjk -o pythonCode -f code.py
```

```
<response>
      <message>mas store model pythonCode (1.0) successfully saved for mas store sjk</message>
  </response>
```

6 A binary analytic store file (`gbt.sasast`) writes to an object named `gbtAstore` in the store:

```
dfesp_mas_stores -n <host>:<port> -w -b -s sjk -o gbtAStore -f gbt.sasast
```

```
<response>
      <message>mas store model gbtAstore (1.0) successfully saved for mas store sjk</message>
 </response>
```

7 Check the contents of the store:

```
dfesp_mas_stores -n host
:port -q <mas-stores>
```

```
<mas-stores>
       <mas-store name='sjk'>
           <version number='1.0'>
              <mas-object name='ds2Code'/>
              <mas-object name='pythonCode'/>
              <mas-object name='gbtAstore'/>
              <mas-object name='new-ds2Code'/>
          </version>
      </mas-store>
 </mas-stores>
```

8 Whenever you do not specify a version as you write objects to the store, the highest version, 1.0, is used. (Version 1.0 is the default version created with the store.) After you create a store, objects can take a new version. Then you can write an object to the new version of the named store. All write commands that do not specify an explicit version use the latest version in the store. Objects in named stores can be accessed from within a project using the following syntax:

```
<mas-modules>
      <mas-module language="ds2" module="module_1"
                  func-names='convert_chars'
                  mas-store='sjk' mas-store-version='1.0>
        <code-file>ds2Code</code-file>
      </mas-module>

      <mas-module language="python" module="module_2"
                  func-names='py_entry'
                  mas-store='sjk'  mas-store-version='1.0'>
        <code-file>pythonCode</code-file>
      </mas-module>

      <mas-module language="ds2" module="module_3"
                  func-names='score_events'
                  mas-store='sjk' mas-store-version='1.0>
        <code-file>new-ds2Code</code-file>
         <module-members>
           <module-member member='astore_1'
SHAkey='EB3D1CA20AA0CB74465D25EEE2290E13692AF750'
                       type='astore'>
             <code-file>gbtAstore</code-file>
           </module-member>
         </module-members>
      </mas-module>
    </mas-modules>
```

In this case, `module_1` and `module_2` are SAS Cloud Analytic Services modules that refer to basic DS2 and Python code. Note that `module_3` is a composite module that has DS2 scoring code that uses an analytic store model in a module member.

SAS Event Stream Processing hides the file storage aspect of objects in SAS Micro Analytic Service stores. However, by default the persistent storage for objects is performed in the `CONFIG` directory of your SAS Event Stream Processing installation. This might not be optimal for large–scale storage of large code bases. It is recommended that you set the environment variable DFESP_STORE to point to some location to use for persistent storage.

Online Models

Overview

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

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

For each model, input maps are tuples of (*Name*, *DataType*, *DefaultValue*) that map incoming event fields to an algorithm's variable values. Output maps are tuples of (*Name*, *DataType*, *DefaultValue*) that map the output fields that are produced by an algorithm to write variable values.

The online algorithms represented in the diagram below are packaged with SAS Event Stream Processing Analytics. Refer to the tables below for more details. For information about the offline models, see Offline on page 202 .

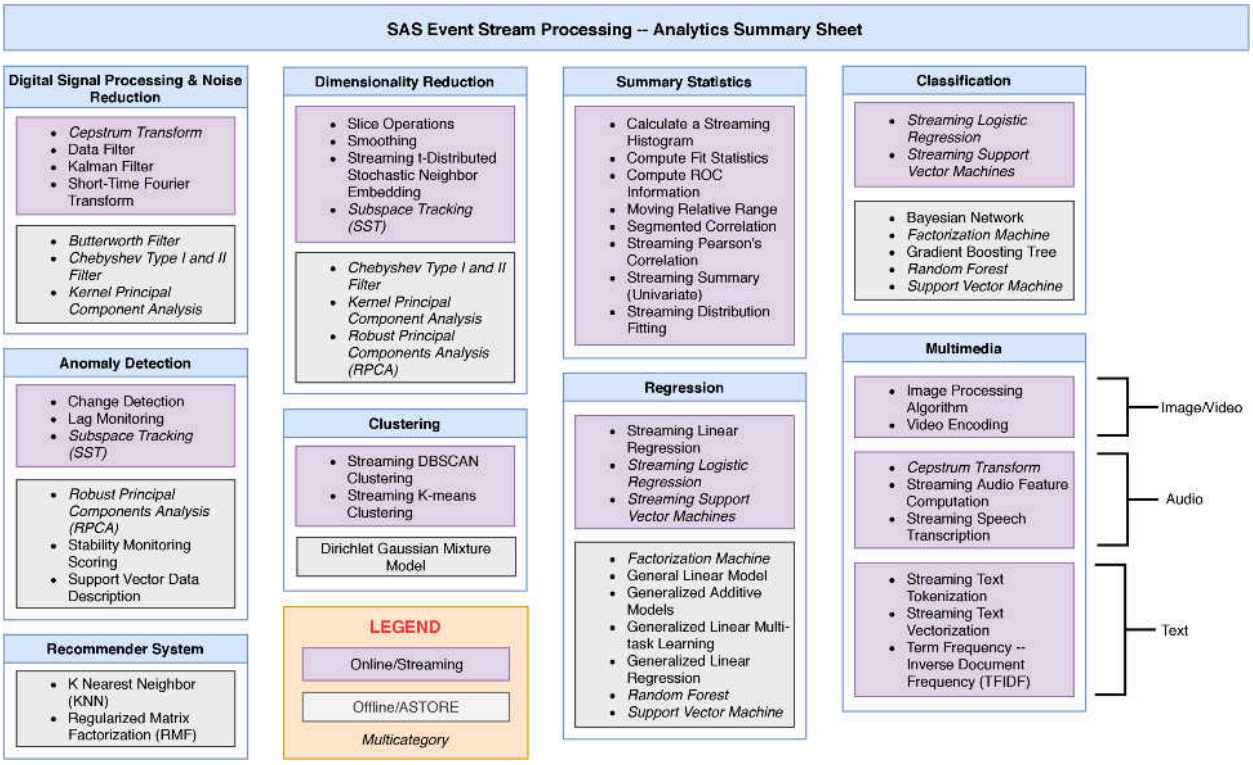


Table 13 *Digital Signal Processing and Noise Reduction*

Algorithm Name	Window Type	General Uses
Cepstrum Transform	Calculate	Cepstral analysis can be used to find out whether a signal contains periodic elements in seismic, speech, and radar signal processing. This method is very effective in digital speech processing to detect the pitch in the human speech signal and extract the transfer function of the vocal tract in voiced speech.
Kalman Filter	Calculate	Kalman filter is a recursive optimal estimation algorithm that is used to make predictions about a variable that cannot be measured directly. To achieve this, the algorithm estimates the state of a system using parameters called "state observers" that provide information about the target variable. Kalman filters use predicted system states and state observers to produce an optimal estimate of the target variable.
Short-Time Fourier Transform (STFT)	Calculate	STFT is commonly used to monitor the time-varying frequency content of a signal. It can be used to detect anomalies in a continuous stream of data, such as machine vibrations. Abnormal conditions can lead to changes in a vibration signal. STFT can be used to monitor the signal frequency band of interest. This monitoring can enable early detection of machine faults and thus lead to more efficient machine maintenance.

Table 14 *Dimensionality Reduction*

Algorithm Name	Window Type	Description
Subspace Tracking (SST)	Calculate	SST is a method to detect anomalies and system degradation in systems that generate high-frequency, high-dimensional data. Examples include monitoring the following: ■ the lighting system in a smart campus to find defective flood lights ■ a solar farm system to detect a defective panel ■ a credit card system to find a fraudulent transaction
Streaming t-Distributed Stochastic Neighbor Embedding	Train and Score	t-Distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm for dimensionality reduction that is used to visualize high-dimensional data sets. It is nonparametric and non-linear. t-SNE renders high-dimensional objects into two- or three-dimensional points, making them more suitable for human observation.
Smoothing	Calculate	The smoothing algorithm uses the median filter to replace each entry of the signal with the median of its neighboring entries within a window. The median filter is a type of non-linear digital filter that is often used to remove noise from signals. It performs better than linear filters at preserving sharp edges while reducing the noise and thus widely used in digital image processing.
Slice Operations	Calculate	The slice operation performs simple operations on a one-dimensional array: mean, median, minimum, and maximum. For example, you can use it to smooth a spectrum from STFT with a median filter.

Table 15 *Summary Statistics*

Algorithm Name	Window Type	Description
Streaming Summary (Univariate Statistics)	Calculate	Streaming summary enables you to calculate univariate statistics (N, number of missing observations, sum, mean, and so on) on a sliding window. You can use it for data exploration and for data monitoring where you send the calculated statistics downstream to trigger action.
Streaming Pearson Correlation	Calculate	Streaming Pearson correlation enables you to calculate the Pearson correlation coefficient between two variables in a sliding window. You can use it to determine the similarity or dissimilarity of the variables. This can be useful to detect when two matched signals become unmatched.
Segmented Correlation	Calculate	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.

Algorithm Name	Window Type	Description
Streaming Distribution Fitting	Calculate	The streaming distribution fitting algorithm fits a Weibull, Gamma, or Normal distribution to a series of data points in a sliding window. You can use the parameters from the fitting in downstream data monitoring.
Compute Fit Statistics	Calculate	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. These measures have a very broad range of practical applications.
Compute ROC Information	Calculate	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.
Calculate a Streaming Histogram	Calculate	A histogram graphically represents a distribution of numerical data. This algorithm processes numerical data and puts it into bins in order to generate boundaries for a histogram to fit it.
Moving Relative Range	Calculate	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. For example, you could use MRR to detect electrical disturbances in the power grid.

Table 16 *Classification Algorithms*

Algorithm Name	Window Type	Description
Streaming Logistic Regression	Train and Score	Streaming logistic regression is an approximation of the standard logistic regression model that is appropriate for streaming data. You supply training examples and mark them as belonging to one of two possible categories. A model that assigns new examples to one of the categories is built.
Streaming Support Vector Machines	Train and Score	Support vector machines are supervised learning models with associated algorithms. They apply classification and regression analysis on incoming data. You supply training examples and mark them as belonging to a category. SVMs build models that assign new examples to provided categories.

Table 17 *Clustering Algorithms*

Algorithm Name	Window Type	Description
Streaming K-Means Clustering	Train and Score	K-Means clustering is an iterative algorithm that partitions data into non-overlapping groups based on their similarity. You can use it to discover hidden structures within the data and to detect outliers. Common uses include market segmentation, image segmentation, and image compression.
Streaming DBSCAN Clustering	Train and Score	DBSCAN clustering is an unsupervised learning method to distinguish clusters of high density from clusters of low density. You can use DBSCAN clustering to cluster location data in order to identify where particular events occur.

Table 18 *Regression Algorithms*

Algorithm Name	Window Type	Description
Streaming Linear Regression	Train and Score	Streaming linear regression is an approximation of the standard linear regression model that is appropriate for streaming data.
Streaming Logistic Regression	Train and Score	Streaming logistic regression is an approximation of the standard logistic regression model that is appropriate for streaming data. You supply training examples and mark them as belonging to one of two possible categories. A model that assigns new examples to one of the categories is built.
Streaming Support Vector Machines	Train and Score	Support vector machines are supervised learning models with associated algorithms. They apply classification and regression analysis on incoming data. You supply training examples and mark them as belonging to a category. SVMs build models that assign new examples to provided categories..

Table 19 *Anomaly Detection*

Algorithm Name	Window Type	Description
Lag Monitoring	Calculate	The lag monitoring algorithm computes the cross-correlation between a target time series and one or more additional time series. Results contain the selected lags and computed cross-correlation values that correspond to minimum, maximum, and maximum absolute value cross-correlations for each of the variables.
Change Detection	Calculate	With change detection, a stream of measures is monitored and an alert is raised when values deviate from what is expected. This algorithm can be used for real-world acoustic event detection for surveillance or multimedia information retrieval.

Algorithm Name	Window Type	Description
Subspace Tracking (SST)	Calculate	SST is a method to detect anomalies and system degradation in systems that generate high-frequency, high-dimensional data. Examples include monitoring the following: ■ the lighting system in a smart campus to find defective flood lights ■ a solar farm system to detect a defective panel ■ a credit card system to find a fraudulent transaction

Table 20 Media Processing

Algorithm Name	Window Type	Description
Streaming Text Tokenization	Calculate	Tokenization splits text into tokens, such as paragraphs, sentences, or individual words. You can submit the results of tokenization to further analysis to detect or predict patterns.
Streaming Text Vectorization	Calculate	Vectorizing text creates maps from words or n-grams to a vector space. A vector space is an algebraic model to represent text documents as vectors of identifiers (for example, index terms).
Image Processing Algorithm	Calculate	Use the image processing algorithm to process streaming image data and manipulate it in real time.
Term Frequency — Inverse Document Frequency (TFIDF)	Calculate	TFIDF is a weight that shows how important a particular word is to a document in a document collection. Deriving this weight is often a necessary step before more advanced analytics such as clustering or classification. You can use TFIDF to analyze Amazon Reviews, Google News, and Twitter feeds.
Video Encoding	Calculate	Use the video encoding algorithm to parse image data in BayerRG8 format to a more common image format. JPEG is the default output format, and PNG is also supported.
Streaming Audio Feature Computation	Calculate	Audio feature computation is the process of applying an algorithm to an audio signal in order to convert it into a sequence of acoustic feature vectors. These vectors contain a serviceable numerical representation of the audio signal. You use the output of this analysis in order to perform streaming speech transcription.
Streaming Speech Transcription	Calculate	Use the streaming speech transcription algorithm to transcribe the output of the acoustic model generated by streaming audio feature computation.

Calculate windows analyze values in input events according to the specified algorithm and publish analysis results to write variable values 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:

Table 21 *Exceptions to One-to-One Correspondence between Input and Output Events for Calculate Windows*

Algorithm	Number of Output Events per Input Event
Text Tokenizer	One for each word token that is 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.

Digital Signal Processing

Overview

Digital signal processing is the process of optimizing the quality of signals and communications. Data that can be represented as a signal wave (audio, temperature, heartbeat, etc.) is manipulated and denoised using smoothing and approximation algorithms.

Applying a Cepstrum Transform

A *cepstrum* results from taking the inverse Fourier transform of the logarithm of the estimated spectrum of a signal. This application is often used in speech analysis, specifically voice pitch detection. Two variations are supported: a complex cepstrum and a real cepstrum. For more information, see *SAS Visual Forecasting: Time Series Packages*.

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 cepstrum calculation

The Source window `w_source` receives a data event.

```
<window-source name='w_source' insert-only='true' autogen-key='true'>
     <schema>
      <fields>
```

```
        <field name='datetime'  type='int64' key='true'/>
        <field name='x' type='double'/>
       </fields>
      </schema>
      <connectors>
      ...
      </connectors>
     </window-source>
```

The Calculate window `w_calculate` receives data events. It calculates the specified cepstrum.

```
<window-calculate name="w_calculate" algorithm="Cepstrum">
      <schema>
          <fields>
          <field name='datetime' type='int64' key='true'/>
          <field name='y' type='array(dbl)'/>
        </fields>
      </schema>
        <parameters>
          <properties>
           <property name="windowLength">128</property>
           <property name="windowType">1</property>
           <property name="windowParam">-1.0</property>
           <property name="overlap">0</property>
           <property name="complexCepstrum">1</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
           <property name="input">x</property>
           <property name="timeId">datetime</property>
          </properties>
        </input-map>
        <output-map>
          <properties>
           <property name="timeIdOut">datetime</property>
           <property name="cepstrumListOut">y[1-128]</property>
          </properties>
        </output-map>
      <connectors>
      ...
      </connectors>
     </window-calculate>
```

The following properties govern the Cepstrum algorithm:

Table 22 *Parameters*

Name	Value Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	128	Specifies the length of the sliding window. The value that you specify must be greater than the value you specify for `overlap`.

Name	Value Type	Required or Optional ?	Default Value	Description
windowType	int64	Optional	15	Specifies the type of the window. Options: 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, 16=Flat Top For more information about these window types, see *SAS Visual Forecasting: Time Series Packages*.
windowParam	double	Optional	-1.0	Some window types require an additional parameter. If not required for the window type selected, this value is ignored.
overlap	int64	Optional	127	Overlap between consecutive windows. Must be strictly less than the value of windowLength.
complexCepstrum	int64	Optional	0	When set to 1, calculate the complex cepstrum. When set to 0 or unspecified, calculate the real cepstrum.

Table 23 *Input Mapping*

Name	Type	Variable Type	Required or Optional ?	Default Value	Description
input	variable	double	Required	No default value	Specifies the name of the analysis variable in the input stream.
timeId	variable	int64	Required	No default value	Specifies the name of the time ID variable in the input stream. It must be uniformly spaced.

Table 24 Output Mapping

Name	Type	Variable Type	Required or Optional ?	Default Value	Description
`timeIdOut`	variable	int64	Required	No default value	Specifies the name of time ID variable in the output stream.
`cepstrumListOut`	varlist	double	Optional	`""` (empty string)	List of cepstrum variables in the output stream. Cepstrum variable values are always real numbers. The algorithm writes `windowLength` values for each output event.

The edge is defined at the end of the project.

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

You can change the values of the properties that govern the Cepstrum algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role “request.” Then, stream a **`reconfig`** request and events that change the property values.

For example, to change the values of all of the properties for Cepstrum:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","32"
i,n,3,"overlap","16"
i,n,4,"windowType","15"
i,n,5,"windowParam","-1.0"
i,n,6,"complexCepstrum","0"
i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

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

Applying a Kalman Filter

A Kalman filter is a recursive optimal estimation algorithm that is used to make predictions about a variable that cannot be measured directly. To achieve this, the algorithm estimates the state of a system using parameters called *state observers* that provide information about the target variable. Kalman filters use predicted system states and state observers to produce an optimal estimate of the target variable. That is, the error between the predicted state and the actual value of the target variable is minimized. The algorithm computation follows two steps:

- A prediction of the current system state using the estimate from the previous state and the current input of state observers

- An update step in which the next state is calculated. The prediction of the state and the error are returned

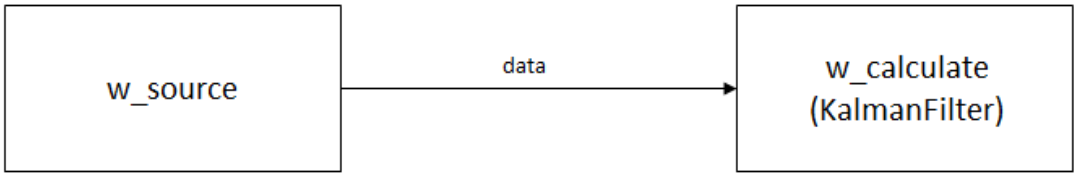
Here are the assumptions about the system

- State space model is linear.
- State and observation noise are independent, zero-mean, and Gaussian random vectors.
- Observation noise has diagonal covariance
- Observation and process noise are mutually independent.
- Observation and state transition matrices (`Z` and `T`) and the observation and state covariance matrices are known at each time step. However, they can change through reconfiguration.

The notation follows the model described for *PROC SSM*, with the matrices W_t, X_t, and A_1 equal to zero. The observation disturbance ϵ_t has diagonal covariance and is specified by `obsCovList`. The state disturbance η_t has covariance Q_t. Q_t can be non-diagonal, as specified in `stateCovList`. `zList` and `tList` specify the matrices Z_t and T_t, respectively. The initial state and covariance correspond to α_1 and Q_1. The input data is Y_t and the control input is c_t.

Note: Matrices in packed form follow the convention used by the SAS/IML *SYMSQR function*.

Consider the following example:



The continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that calculates a prediction and publishes the results

The Source window `w_source` receives the data event.

```
<window-source name="w_source">
    <schema>
        <fields>
            <field-name='ID' type='int64' key='true'/>
            <field-name='x' type='double'/>
            <field name='y' type'double'/>
            <field name='tx' type='double'/>
            <field name='ty' type='doublt'/>
        </fields>
    </schema>
    <connectors>
    ...
    </connectors>
</window-source>
```

The Calculate window `w_calculate` receives data events. It calculates a prediction of the system state and publishes the results and error.

```
<window-calculate name='w_calculate' algorithm='KalmanFilter' collapse-updates='true'>
    <schema>
```

```
            <fields>
                <field name='ID' type='int64' key='true'/>
                <field name='x' type='double'/>
                <field name='y' type='double'/>
                <field name='tx' type='double'/>
                <field name='ty' type='double'/>
                <field name='pred' type='aray(dbl)'/>
                <field name='vpred' type='aray(dbl)'/>
                <field name='filt' type='aray(dbl)'/>
                <field name='vfilt' type='aray(dbl)'/>
            </fields>
        </schema>
        <parameters>
            <properties>
                <property name="stateDim">4</property>
                <property name="zlist">1,0,0,0,0,0,1,0</property>
                <property name="tList">0.5,0.5,0,0,0,0.5,0,0,0,0,0.5,0.5,0,0,0,0.5</
property>
                <property
name="stateCovList">0.00333333,0.005,0.01,0,0,0.00333333,0,0,0.005,0.01</property>
                <property name="obsCovList">0.09,0,0.09</property>
                <property name="initStateList">0,0.1,0,0.1</property>
                <property name="initStateCovList">0,0,0,0,0,0,0,0,0,0</property>
            </properties>
        </parameters>
        <input-map>
            <properties>
                <property name="inputList">x,y</property>
            </properties>
        </input-map>
        <output-map>
            <properties>
                <property name="predListOut">pred[1-4]</property>
                <property name="vpredListOut">vpred[1-10]</property>
                <property name="filtListOut">filt[1-4]</property>
                <property name="vfiltListOut">vfilt[1-10]</property>
            </properties>
        </output=map>
    </window-calculate>
```

The following properties govern the Kalman Filter algorithm in the Calculate window:

Table 25 *Parameters*

Name	Value Type	Required or Optional?	Default Value	Description
`stateDim`	int64	Required	No default value	Specifies the dimension of the state space. The value specified must be >= 1.
`zList`	double-list	Required	`""` (empty string)	Specifies a comma-separated list containing the measurement

Name	Value Type	Required or Optional?	Default Value	Description
				matrix. The size of `zList` is yDim * `stateDim`, where yDim is the length of `inputList`.
`tList`	double-list	Required	`""` (empty string)	Specifies a comma-separated list containing the state transition matrix. The size of `tList` is `stateDim` * `stateDim`.
`stateCovList`	double-list	Required	`""` (empty string)	Specifies a comma-separated list containing the state covariance symmetric matrix. In packed form, the size of `stateCovList` is `stateDim`*(`stateDim`+1)/2.
`obsCovList`	double-list	Required	`""` (empty string)	Specifies a comma-separated list containing the measurement covariance symmetric matrix. In packed form, the size of `obsCovList` is yDim*(yDim+1)/2.
`initStateList`	double-list	Optional	`0,...,0`	Specifies a comma-separated list containing the initial state vector. The size of `initStateList` is `stateDim`.
`initStateCovList`	double-list	Optional	`""` (empty string)	Specifies a comma-separated list containing the initial state covariance symmetric matrix. In packed form, the size of `initStateCovLis`

Name	Value Type	Required or Optional?	Default Value	Description
				`t` is `stateDim`*(`stateDim`+1)/2. If this parameter is not specified, the code calculates a reasonable default based on the `tList` and `stateCovList` parameters.

Table 26 *Input Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`inputList`	varlist	double	Required	No default value	Specifies the list of measurement variable names in the input stream.
`controlList`	varlist	double	Optional	`""` (empty string)	Specifies the list of control variable names in the input stream.

Table 27 *Output Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`predListOut`	varlist	double	Optional	`""` (empty string)	Specifies the list for the predicted state vector in the output stream. The length of `predListOut` is `stateDim`).
`vPredListOut`	varlist	double	Optional	`""` (empty string)	Specifies the list for the covariance of the predicted state vector in the output stream. In packed form, the length of `vPredListOut` is `stateDim`*(`stateDim`+1)/2.
`filtListOut`	varlist	double	Optional	`""` (empty string)	Specifies the list for the filtered state vector in the output stream. The length of `filtListOut` is `stateDim`.
`vFiltListOut`	varlist	double	Optional	`""` (empty string)	Specifies the list for the covariance of the filtered state vector in the output stream. In packed form, the length of `vFiltListOut` is `stateDim`*(`stateDim`+1)/2.

The edge is defined at the end of the project.

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

You can change the values of the properties that govern the Kalman Filter algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role “request”. The, stream a `reconfig` request and events that change the property values.

Here is an example of how to change the values of all the properties for Kalman Filter:

```
i,n,1,"action","reconfig"
i,n,2,"obsCovList","0.45,0,0,0.45"
i,n,3,"stateDim","4"
i,n,4,"zList","1,0,0,0,0,0,1,0"
i,n,5,"tList","0.5,0.5,0,0,0,0.5,0,0,0,0,0.5,0.5,0,0,0,0.5"
i,n,6,"stateCovList","0.00333333,0.005,0.01,0,0,0.00333333,0,0,0.005,0.01"
i,n,7,"initStateList","0,0.1,0,0.1"
i,n,8,"initStateCovList","0,0,0,0,0,0,0,0,0,0"
i,n,9,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

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

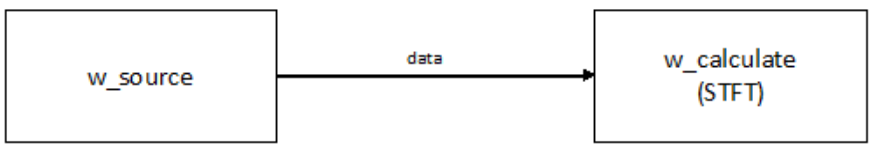
Calculating Short-Time Fourier Transforms

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. Short-time Fourier transform (STFT) computations consist of multiple “local” discrete Fourier transform computations. The input time series is divided into multiple contiguous bins, and their discrete Fourier transforms are computed in succession. The use of window functions makes the spectra smooth.

STFT is commonly used to monitor the time-varying frequency content of a signal. It can be used in digital filters to detect anomalies in a continuous stream of data. For example, vibrations indicate machine operating conditions. Abnormal conditions can lead to changes in the vibration signal. STFT can be used to monitor the signal frequency band of interest. This monitoring can enable early detection of machine faults and thus more efficient machine maintenance.

For more information about how STFT is implemented in SAS software, see the *SAS Forecast Server: Time Series Packages*.

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

In the code that follows, a Source window `w_source` receives input data. 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' autogen-key='true'>
     <schema>
      <fields>
       <field name='ID'  type='int64' key='true'/>
       <field name='y' type='double'/>
      </fields>
     </schema>
     <connectors>
     ...
     </connectors>
    </window-source>
```

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

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

Table 28 *Parameters*

Name	Variable Type	Required or Optional ?	Default Value	Description
`windowLength`	int64	Optional	`128`	Specifies the length of the sliding window. The value that you specify must be greater than the value you specify for `overlap`.
`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. For more information about these window types, see *SAS Visual Forecasting: Time Series Packages*.
`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.

Name	Variable Type	Required or Optional ?	Default Value	Description
`overlap`	int64	Optional	`127`	Specifies the overlap between consecutive windows. Must be strictly less than `windowLength`.
`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.

Table 29 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`input`	variable	double	Required	No default value	Specifies the input variable by its name in the source schema. The Calculate window analyzes this variable.
`timeId`	variable	int64	Required	No default value	Specifies the time ID variable name in the input stream. It must be uniformly spaced.

Table 30 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`keyOut`	variable	int64	Optional	`""` (empty string)	Specifies a key variable name (unique for each output event) in the output stream.
`timeIdOut`	variable	int64	Required	No default value	Specifies the time ID variable name in the output stream. There is more than one output event for a given time ID.
`binOut`	variable	int64	Optional	`""` (empty string)	Specifies the frequency bin variable name in the output stream.
`powerOut`	variable	double	Optional	`""` (empty string)	Specifies the name of the power variable in the output stream.
`phaseOut`	variable	double	Optional	`""` (empty string)	Specifies the name of the phase variable in the output stream.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
powerListOut	varlist	double	Optional	"" (empty string)	Specifies a list of power variables in the output stream.
phaseListOut	varlist	double	Optional	"" (empty string)	Specifies a list of phase variables in the output stream.

You can use one of two formats for output mapping in the Calculate window:

- a list format where one event is generated for each time ID and the event contains an array of the output power variables for the corresponding bins
- a non-list format where a separate output event is generated for each time ID and bin pair

For output mapping that uses a list format, you specify `powerListOut` or `phaseListOut` or both. You use a key of `timeIdOut`.

For output mapping that uses a non-list format, you specify `powerOut` or `phaseOut` or both. You can use a key of `keyOut` or a composite key of `timeIdOut` and `binOut`.

The following code uses non-list format:

```
<window-calculate name="w_calculate" algorithm="STFT">
      <schema>
         <fields>
          <field name='time' type='int64' key='true'/>
          <field name='bin' type='int64' key='true'/>
          <field name='power' type='double'/>
          <field name='phase' type='double'/>
         </fields>
      </schema>
        <parameters>
          <properties>
           <property name="windowLength">64</property>
           <property name="windowType">12</property>
           <property name="fftLength">256</property>
           <property name="binsInSchema">256</property>
           <property name="overlap">32</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
           <property name="input">y</property>
           <property name="timeId">ID</property>
          </properties>
        </input-map>
        <output-map>
          <properties>
           <property name="timeIdOut">time</property>
           <property name="binOut">bin</property>
           <property name="powerOut">power</property>
           <property name="phaseOut">phase</property>
          </properties>
        </output-map>
      <connectors>
```

```
      ...
      </connectors>
    </window-calculate>
```

The following code uses the list format:

```
<window-calculate name="w_calculate" algorithm="STFT">
      <schema>
        <fields>
          <field name='time' type='int64' key='true'/>
          <field name='powerlist' type='array(dbl)'/>
          <field name='phaselist' type='array(dbl)'/>
        </fields>
      </schema>
        <parameters>
          <properties>
           <property name="windowLength">64</property>
           <property name="windowType">12</property>
           <property name="fftLength">256</property>
           <property name="binsInSchema">256</property>
           <property name="overlap">32</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
           <property name="input">y</property>
           <property name="timeId">ID</property>
          </properties>
        </input-map>
        <output-map>
          <properties>
           <property name="timeIdOut">time</property>
           <property name="powerListOut">powerlist[1-256]</property>
           <property name="phaseListOut">phaselist[1-256]</property>
          </properties>
        </output-map>
      <connectors>
      ...
      </connectors>
    </window-calculate>
```

An error is generated when you specify the list and non-list formats in the same Calculate window.

Note: When `keyOut` is the key in the output schema, there is a limitation of 9999 bins for each `timeId`.

In all cases, the calculated STFT output data is organized by event fields that are specified in the schema of the Calculate window.

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 change the values of the properties that govern the STFT algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the

Calculate window with the role “request.” Then, stream a **reconfig** request and events that change the property values.

For example, to change the values of all of the properties for STFT:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","64"
i,n,3,"overlap","32"
i,n,4,"windowType","15"
i,n,5,"fftLength","64"
i,n,6,"binsInSchema","32"
i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

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.

Dimensionality Reduction

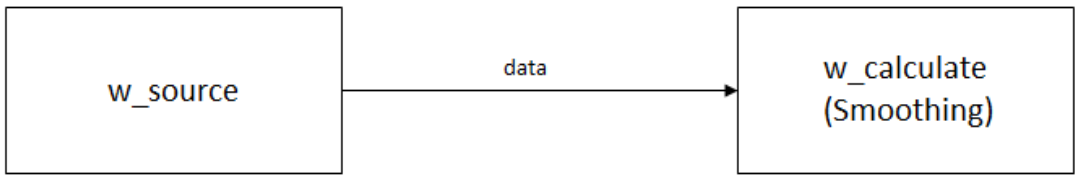
Overview

Dimensionality reduction techniques are applied in the preliminary stage of analysis to simplify and enhance the more critical features of data, making it more useable to work with and to optimize calculation performance (I.e. algorithm time complexity and storage space).

Smoothing

The smoothing algorithm uses the median filter to replace each entry of the signal with the median of its neighboring entries within a window. The median filter is a type of non-linear digital filter that is often used to remove noise from signals. It performs better than linear filters at preserving sharp edges while reducing the noise.

Consider the following example:



Here are the contents of the continuous query:

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

The Source window `w_source` receives input data.

```
<window-source insert-only="true" name="w_data">
```

```
            <schema>
              <fields>
                 <field name='id' type='int64' key='true'/>
                 <field name='x1' type='double'/>
                 <field name='x2' type='double'/>
                 <field name='x3' type='double'/>
                 <field name='x4' type='double'/>
                 <field name='x5' type='double'/>
                 <field name='x6' type='double'/>
              </fields>
            </schema>
            <connectors>
            ...
            </connectors>
          </window-source>
```

The Calculate window `w_calculate` receives data events from `w_source`. It performs the smoothing algorithm.

```
<window-calculate name='w_calculate' algorithm='Smoothing'>
           <schema>
                <fields>
                   <field name='id' type='int64' key='true'/>
                   <field name='median_x1' type='double'/>
                   <field name='median_x2' type='double'/>
                   <field name='median_x3' type='double'/>
                   <field name='median_x4' type='double'/>
                   <field name='median_x5' type='double'/>
                   <field name='median_x6' type='double'/>
                </fields>
              </schema>
              <parameters>
                <properties>
                   <property name="windowLength">3</property>
                   <property name="overlap">2</property>
                   <property name="smoothingType">Median</property>
                </properties>
              </parameters>
              <input-map>
                <properties>
                   <property name="inputList">x1, x2, x3, x4, x5, x6</property>
                </properties>
              </input-map>
              <output-map>
                 <properties>
                   <property name="outputList">median_x1, median_x2, median_x3,
median_x4, median_x5, median_x6</property>
                </properties>
              </output-map>
              <connectors>
              ...
              </connectors>
            </window-calculate>
```

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

Table 31 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	`128`	Specifies the length of the sliding window.
`smoothingType`	string	Optional	`Median`	Specifies the smoothing type. Available types are `Median`.

Table 32 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`inputList`	varlist	double	Required	No default value	Specifies the list of analysis variable names in the input stream.

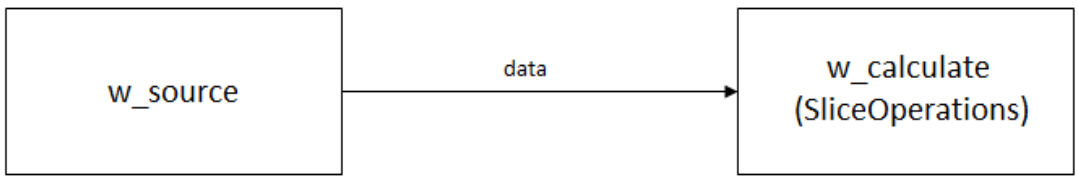
Table 33 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`outputList`	varlist	double	Optional	`""` (empty string)	Specifies the list of smoothed output variable names in the output stream.

Applying Slice Operations

The slice operation performs simple operations on a one-dimensional array: mean, median, minimum, and maximum.

Consider the following example:



Here are the contents of the continuous query:

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

The Source window `w_source` receives the data event.

```
<window-source name='w_source' insert-only='true' autogen-key='true'>
     <schema>
      <fields>
       <field name='id'  type='int64' key='true'/>
       <field name='x' type='array(dbl)'/>
      </fields>
     </schema>
     <connectors>
     ...
     </connectors>
    </window-source>
```

The Calculate window `w_calculate` receives data events. It applies the operation.

```
<window-calculate name='w_calculate' algorithm='SliceOperations' collapse-
updates='true'>
      <schema>
          <fields>
          <field name='id' type='int64' key='true'/>
          <field name='minOut' type='double'/>
          <field name='maxOut' type='double'/>
          <field name='meanOut' type='double'/>
          <field name='filtListOut' type='array(dbl)'/>
        </fields>
      </schema>
        <parameters>
          <properties>
           <property name="medianWindowLength">5</property>
          </properties>
        </parameters>
        <input-map>
          <properties>
           <property name="inputList">x[1-20]</property>
          </properties>
        </input-map>
        <output-map>
          <properties>
           <property name="filtListOut">filtListOut[1-16]</property>
           <property name="meanOut">meanOut</property>
           <property name="minOut">minOut</property>
           <property name="maxOut">maxOut</property>
          </properties>
        </output-map>
     <connectors>
     ...
     </connectors>
    </window-calculate>
```

The following properties govern the slice operations in the Calculate window:

Table 34 *Parameters*

Name	Value Type	Required or Optional?	Default Value	Description
`medianWindowLength`	int64	Optional	3	Specifies the window length for the median filter. The value should be

Name	Value Type	Required or Optional?	Default Value	Description
				between 1 and the length of the input array.

Table 35 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`inputList`	varlist	double	Required	No default value	Specifies the list of variables containing the incoming data.

Table 36 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`filtListOut`	varlist	double	Optional	`""` (empty string)	Specifies the variable for the mean filtered array output. It must have a length of N —`medianWindowLength` +1, where N is the length of the input array.
`meanOut`	variable	double	Optional	`""` (empty string)	Specifies the variable for the mean output.
`minOut`	variable	double	Optional	`""` (empty string)	Specifies the variable for the minimum value output.
`maxOut`	variable	double	Optional	`""` (empty string)	Specifies the variable for the maximum value output.

Subspace Tracking (SST)

Suppose that data contains a sequence of *n* X 1 vectors: $x(t)$. Subspace tracking (SST) estimates the covariance matrix for each vector $x(t)$ and then computes the first *p* principal eigenvectors of the covariance matrix. For each iteration at time *t*, the covariance matrix $C(t)$ is obtained by the following:

$$\mu(t) = (1-\alpha)\,\mu(t-1) + \alpha\; x(t)$$
$$C(t) = (1-\beta)\,C(t-1) + \beta(x(t) - \mu(t))(x(t) - \mu(t))^T$$

Here, α and β are the mean and covariance forgetting factors whose values are predetermined to be between 0 and 1, respectively. The first *p* principal eigenvector $W(t)$ can be obtained by the eigendecomposition of the covariance matrix.

Another method for tracking the subspace is to use the moving windows principal component analysis. In this method, there are no forget factors. Window length, however, has to be specified. For more information about window-based subspace tracking, see the MWPCA procedure in *SAS Visual Data Mining and Machine Learning: Procedures*. For more information about subspace tracking with forget factors, see the Subspace Tracking Page in *SAS Visual Forecasting: Time Series Packages*.

SST can be applied to industrial data to detect outliers and use results to identify potential errors before they occur.

Consider the following example:

Here are the contents of the continuous query:

- a Source window that receives the data to analyze
- a Calculate window that performs SST

The Source window `w_source` receives input data. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; and a series of six different `x` coordinate fields (`x1`, `x2`,`x3`,`x4`, `x5`, `x6`).

```
<window-source name='w_source'>
              <schema>
                <fields>
                   <field name='id' type='int64' key='true'/>
                   <field name='x1' type='double'/>
                   <field name='x2' type='double'/>
                   <field name='x3' type='double'/>
                   <field name='x4' type='double'/>
                   <field name='x5' type='double'/>
                   <field name='x6' type='double'/>
                </fields>
              </schema>
              <connectors>
              ...
              </connectors>
            </window-source>
```

The Calculate window `w_calculate` receives data events from `w_source`. It publishes the calculated principal components and output values of the subspace according to the SST algorithm properties that are specified.

```
<window-calculate name='w_calculate' algorithm='SST'>
          <schema>
               <fields>
                  <field name='id' type='int64' key='true'/>
                  <field name='projAngle' type='double'/>
                  <field name='residualRate' type='double'/>
                  <field name='numRank' type='int32'/>
                  <field name='prin1_x1' type='double'/>
                  <field name='prin1_x2' type='double'/>
                  <field name='prin1_x3' type='double'/>
```

```
                    <field name='prin1_x4' type='double'/>
                    <field name='prin1_x5' type='double'/>
                    <field name='prin1_x6' type='double'/>
                  </fields>
                </schema>
                <parameters>
                  <properties>
                    <property name="maxPrincipal">2</property>
                    <property name="meanForgetFactor">0.1</property>
                    <property name="covForgetFactor">0.5</property>
                    <property name="eigvalTolCumulative">1</property>
                  </properties>
                </parameters>
                <input-map>
                  <properties>
                    <property name="inputs">x1, x2, x3, x4, x5, x6</property>
                  </properties>
                </input-map>
                <output-map>
                   <properties>
                    <property name="projAngleOut">projAngle</property>
                    <property name="residualOut">residualRate</property>
                    <property name="numRankOut">numRank</property>
                    <property name="principalVecOut">prin1_x1, prin1_x2, prin1_x3,
prin1_x4, prin1_x5, prin1_x6</property>
                  </properties>
                </output-map>
                <connectors>
                ...
                </connectors>
              </window-calculate>
```

The following properties govern the SST algorithm:

Table 37 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	0	Specifies the length of the sliding window. A sliding window enables you to use multiple events to update principal components. A value of 0 denotes unlimited length. If the value is greater than 0,`covForgetFactor` and `meanForgetFactor` are ignored for updating the covariance matrix. The value that you specify must be greater than the value that you specify for `overlap`.
`overlap`	int64	Optional	-1	Specifies the overlap between consecutive windows. Must be strictly less than `windowLength`. The default value of `-1`

Name	Value Type	Required or Optional ?	Default Value	Description
				means that `overlap` is internally calculated as `windowLength` -1.
`maxPrincipal`	int32	Optional	1	Specifies the maximum number of the principal eigenvectors. Specify a value greater than 0 and less than or equal to the number of input variables.
`covForgetFactor`	double	Optional	0.5	Specifies the forgetting factor that is used to update the covariance matrix. Specify a value between 0 and 1.
`meanForgetFactor`	double	Optional	0.1	Specifies the value of the forgetting factor used to update the mean. Specify a value between 0 and 1.
`eigvalTolCumulative`	double	Optional	1	Specifies the threshold on the cumulative rate of eigenvalues. Specify a positive value less than or equal to 1.

Table 38 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional ?	Default Value	Description
`inputs`	varlist	double	Required	No default value	Specifies the list of variable names used to compute the principal subspace.

Table 39 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional ?	Default Value	Description
`PCAngleChangeOut`	variable	double	Optional	"" (empty string)	Specifies the output variable name for the angle change between the first principal component vector of two consecutive subspaces.
`PCAbsoluteAngleOut`	variable	double	Optional	"" (empty string)	Specifies the output variable name for the absolute angle of the first principal component vector.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
projAngleOut	variable	double	Optional	"" (empty string)	Specifies the output variable name of the projection angle.
numRankOut	variable	int32	Optional	"" (empty string)	Specifies the output variable name of the rank of the principal subspace.
principalVecOut	varlist	double	Optional	"" (empty string)	Specifies a list of variable names that correspond to the elements of each principal eigenvector. Suppose that the data consists of five variables: x1, x2, x3, x4, and x5. If you want to produce the third principal eigenvector, prin3, then you must specify the variable names as follows: prin1_x1, prin1_x2, prin1_x3, prin1_x4, prin1_x5, prin2_x1, prin2_x2, prin2_x3, prin2_x4, prin2_x5, prin3_x1, prin3_x2, prin3_x3, prin3_x4, and prin3_x5. That is, you must specify all of the elements of the first and second principal eigenvector to produce the elements of the third. Often, the first principal eigenvector provides the most useful information.
residualOut	variable	double	Optional	"" (empty string)	Specifies the output variable name of the residual.
projectionVecOut	varlist	double	Optional	"" (empty string)	Specifies the output variable name of the vector projected into the principal subspace.

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 SST algorithm parameter properties for the Calculate window with the command-line utility.

You can change the values of the properties that govern the SST algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role “request.” Then, stream a **`reconfig`** request and events that change the property values.

For example, to change the values of all of the properties for SST:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","64"
i,n,3,"maxPrincipal","2"
i,n,4,"covForgetFactor","1"
i,n,5,"meanForgetFactor","1"
i,n,6,"eigvalTolCumulative","0.75"
i,n,7,,
```

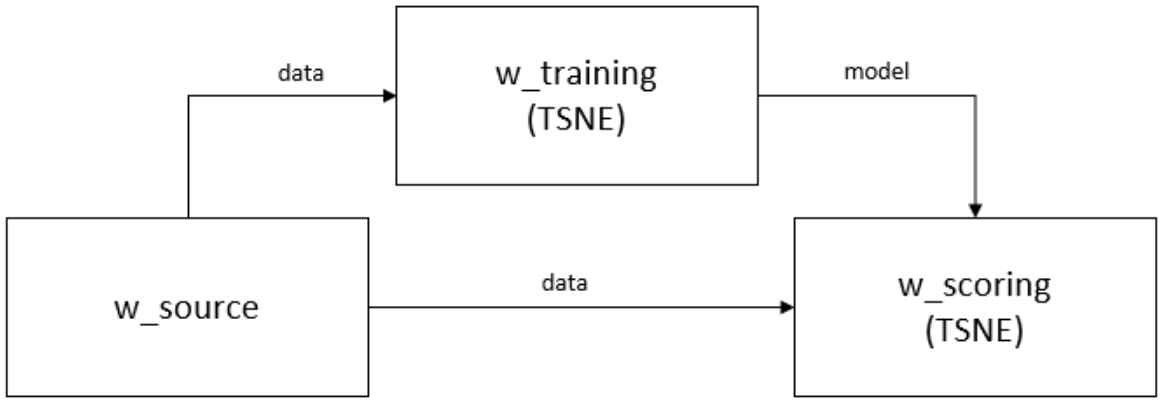
These events immediately change property values. You can change one or more property values at a time, as required.

Training and Scoring with t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm for dimensionality reduction that is used to visualize high-dimensional data sets. It is nonparametric and non-linear. t-SNE renders high-dimensional objects into two- or three-dimensional points, making them more suitable for human observation. Similar objects are modeled by nearby points and dissimilar objects are modeled by distant points. For more information, refer to Laurens Van Der Maaten’s github site.

Consider the following model:

Figure 2 *TSNE model*



This continuous query includes the following:

- a Source window that receives data to be scored and trained
- a Train window that generates and periodically updates the t-SNE model
- a Score window that performs the scoring

The Source window `w_source` receives a data event. Each observation that is streamed into the window consists of several fields that pertain to the characteristics of a flower.

```
<window-source name="w_source" pubsub="true" insert-only="true">
  <schema>
    <fields>
      <field type="int64" name="id" key="true" />
      <field type="string" name="sepal_length" />
      <field type="double" name="sepal_width" />
      <field type="double" name="petal_length" />
      <field type="double" name="petal_width" />
      <field type="double" name="species" />
    </fields>
  </schema>
  <connectors>
   ...
  </connectors>
</window-source>
```

The Train window `w_training` parses all observations and periodically generates a new model that uses the t-SNE algorithm.

```
<window-train name="w_training" algorithm="TSNE">
  <parameters>
    <properties>
       <property name="nDims">2</property>
       <property name="initSeed">54321</property>
       <property name="nLandmarks">150</property>
       <property name="perplexity">5</property>
       <property name="learnRate">200</property>
       <property name="maxIters">600</property>
       <property name="nInit">150</property>
       <property name="commitInterval">1000</property>
    </properties>
  </parameters>
  <input-map>
    <properties>
      <property name="inputs">sepal_length,sepal_width,petal_length,petal_width</
property>
    </properties>
  </input-map>
</window-train>
```

The following properties govern the TSNE algorithm in the Train window:

Table 40 *Parameters*

Name	Type	Required or Optional?	Default Value	Description
`nDims`	int32	Optional	`2`	Specifies the number of embedding dimensions to report. Specify a value of 2 or 3.
`initSeed`	int32	Optional	`12345`	Specifies the random seed to use during initialization.

Name	Type	Required or Optional?	Default Value	Description
`nLandmarks`	int32	Optional	`100`	Specifies the number of landmarks to keep in the model. Landmarks usually consist of past observations. The model retains both the original landmark observations and their embeddings.
`perplexity`	double	Optional	`30`	Specifies the perplexity parameter. This parameter determines how tightly to couple the embeddings for faraway observations. Lower perplexity values tend to produce a higher number of small, scattered groups.
`learnRate`	double	Optional	`100`	Specifies the learning rate for the gradient descent optimization procedure.
`maxIters`	int32	Optional	`500`	Specifies the maximum number of iterations for the gradient descent optimization procedure.
`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 be elapsed before triggering a commit of model to downstream scoring.

Table 41 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	double	Optional	`""` (empty string)	Specifies the list of variable names used for the high-dimensional analysis. Variable names are defined in the input schema, and they are separated by a comma (for example, **`x,y`**).

The Train window publishes models to the Score window `w_scoring`, which uses them to score incoming data.

```
<window-score name="w_scoring" pubsub="true">
  <schema>
    <fields>
        <field name='species' type='string'/>
```

```
          <field name='id'  type='int64' key='true'/>
          <field name='_DIM_1_' type='double'/>
          <field name='_DIM_2_' type='double'/>
      </fields>
    </schema>
    <models>
      <online algorithm="TSNE">
        <input-map>
          <properties>
            <property name="inputs">sepal_length,sepal_width,petal_length,petal_width</
property>
          </properties>
        </input-map>
        <output-map>
          <properties>
            <property name="embeddingOut1">_DIM_1_</property>
            <property name="embeddingOut2">_DIM_2_</property>
          </properties>
        </output-map>
      </online>
    </models>
    <connectors>
     ...
    </connectors>
  </window-score>
```

The following properties are unique to Score windows for TSNE models:

Table 42 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Optional	"" (empty string)	Specifies the list of variable names that are used for the high-dimensional analysis. These variables should be identical to those you specify for the Train window.

Table 43 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
embeddingOut1	variable	double	Optional	"" (empty string)	Specifies the output variable name in the output schema that stores the first embedding dimension. If not specified, the embedding dimensions column is not shown.
embeddingOut2	variable	double	Optional	"" (empty string)	Specifies the output variable name in the output schema that stores the second embedding dimension. If not

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
					specified, the embedding dimensions column is not shown.
`embeddingOut3`	variable	double	Optional	`""` (empty string)	specifies the output variable name in the output schema that stores the third embedding dimension. If not specified, the embedding dimensions column is not shown. **Note:** There can be no more than three embedding dimensions.

Edges are defined at the end of the project.

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

Summary Statistics

Overview

Summary statistics are measures that provide meaningful information about the data being analyzed beyond what can be inferred by manual inspection.

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 Statistics: 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` sends a data event.

```
<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>
               ...
              </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>
       ...
       </connectors>
     </window-calculate>
```

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

Table 44 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
`labelLen`	int64	Optional	128	Specifies the length of the response event.
`cutStep`	double	Optional	0.01	Specifies the bin width. Specify a value between 0 and 1. The default, 0.01, generates 100 bins to fit the ROC.
`windowLength`	int64	Optional	0	Specifies the length of the sliding window. The overlap between consecutive windows is `windowLength` — 1.
`event`	string	Required	No default value	Specifies the desired response event to use for ROC calculations.

Table 45 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`input`	variable	double	Required	No default value	Specifies the input variable, which is the predicted probability for the given event.
`response`	response variable	string	Required	No default value	Specifies the response variable.

Table 46 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`binIDout`	variable	int64	Optional	`""` (empty string)	Specifies the bin ID.
`cutoffOut`	variable	double	Optional	`""` (empty string)	Specifies the cutoff probability.
`tpOut`	variable	double	Optional	`""` (empty string)	Specifies the number of true positives.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
fpOut	variable	double	Optional	"" (empty string)	Specifies the number of false positives.
fnOut	variable	double	Optional	"" (empty string)	Specifies the number of false negatives.
tnOut	variable	double	Optional	"" (empty string)	Specifies the number of true negatives.
sensitivityOut	variable	double	Optional	"" (empty string)	Specifies the ROC sensitivity.
specificityOut	variable	double	Optional	"" (empty string)	Specifies the ROC specificity.
ksOut	variable	double	Optional	"" (empty string)	Specifies the Kolmogorov-Smirnov statistic.
ks2Out	variable	double	Optional	"" (empty string)	Specifies the KS2.
fHalfOut	variable	double	Optional	"" (empty string)	Specifies the F_Half.
fprOut	variable	double	Optional	"" (empty string)	Specifies the false positive rate.
accOut	variable	double	Optional	"" (empty string)	Specifies the accuracy (ACC).
fdrOut	variable	double	Optional	"" (empty string)	Specifies the false discovery rate.
f1Out	variable	double	Optional	"" (empty string)	Specifies the F1 score.
cOut	variable	double	Optional	"" (empty string)	Specifies C (area under the curve). $C = \frac{\mu + \frac{\theta}{2}}{\rho}$
giniOut	variable	double	Optional	"" (empty string)	Specifies the Gini coefficient. $\text{Gini} = \frac{\mu - \omega}{\rho}$
gammaOut	variable	double	Optional	"" (empty string)	Specifies Goodman and Kruskal's Gamma $\text{gamma} = \frac{\mu - \omega}{\mu + \omega}$

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`tauOut`	variable	double	Optional	`""` (empty string)	Specifies Kendall's Tau-a $\text{tau} = \frac{\mu - \omega}{\frac{N}{2}(N-1)}$
`miscEventOut`	variable	double	Optional	`""` (empty string)	Specifies the Misclassification rate (1– area under the curve).

For these output variables, the following is true:

$$\theta = \sum_{k=1}^{m} ((a_{k-1} - a_k)(b_{k-1} - b_k))$$
$$\mu = \sum_{k=2}^{m} ((a_{k-1} - a_k) \sum_{j=1}^{k} (b_{j-1} - b_j))$$
$$\omega = \sum_{k=1}^{m} ((a_{k-1} - a_k) \sum_{j=k+1}^{m} (b_{j-1} - b_j))$$
$$\rho = a_0 b_0, a_m = 0 \quad \text{and} \quad b_m = 0$$

The edge is defined at the end of the project.

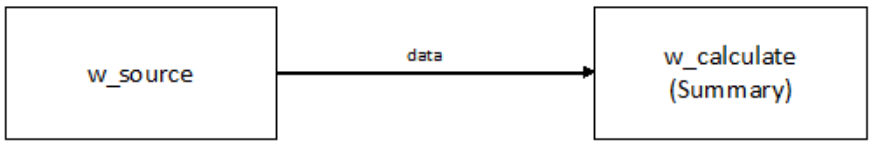
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
  </edges>
```

You can view the default values of the ROC algorithm parameter properties for the Calculate window 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:



Here are the contents of the continuous query:

- 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. 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>
     ...
     </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>
   ...
   </connectors>
 </window-calculate>
```

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

Table 47 *Parameters*

Name	Variable Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	`3`	Specifies the length of the sliding window. The default value is 3. The overlap between consecutive windows is `windowLength` — 1.
`alpha`	double	Optional	`0.05`	Specifies the coverage probability (1– α) for two-sided confidence intervals.

Table 48 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`input`	variable	double	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 49 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`nOut`	variable	int64	Optional	"" (empty string)	Specifies the output variable name for the number of observations analyzed for the incoming data events (N).
`nmissOut`	variable	int64	Optional	"" (empty string)	Specifies the output variable name for the number of missing values in the incoming data events (NMISS).

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
minOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the minimum observed value (MIN).
maxOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the maximum observed value (MAX).
sumOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the linear sum (SUM).
meanOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the mean (MEAN).
stdOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the standard deviation (STD).
varOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the sample variance (VAR).
cssOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the corrected sum of squares (CSS).
ussOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the uncorrected sum of squares (USS).
stderrOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the standard error (STDERR).
cvOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the coefficient of variation (CV).
tstatOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the t-statistic (TSTAT).
probtOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for Pr > |t| (PROBT).
uclmOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the upper confidence limit (UCLM).

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`lclmOut`	variable	double	Optional	"" (empty string)	Specifies the output variable name for the lower confidence limit (LCLM).
`skewnessOut`	variable	double	Optional	"" (empty string)	Specifies the output variable name for skewness (SKEWNESS).
`kurtosisOut`	variable	double	Optional	"" (empty string)	Specifies the output variable name for kurtosis (KURTOSIS).

The calculated summary statistics are organized into event fields that are specified in the schema of the Calculate window.

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

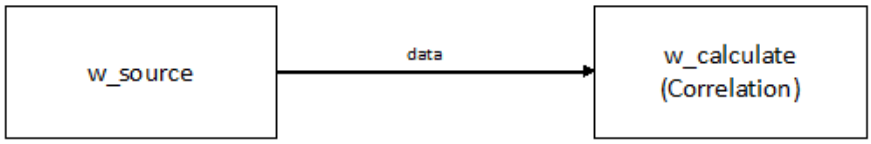
```
<edges>
    <edge source='w_source' target='w_calculate' role='data'/>
   </edges>
```

You can view the default values of the summary statistics algorithm parameter properties for the Calculate window with 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. 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>
    ...
    </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>
     ...
     </connectors>
    </window-calculate>
```

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

Table 50 *Parameters*

Name	Variable Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	`3`	Specifies the length of the sliding window. The default value is 3. The overlap between consecutive windows is `windowLength` — 1.

Table 51 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
x	variable	double	Required	No default value	Specifies the input variable X by its name in the source schema.
y	variable	double	Required	No default value	Specifies the input variable Y by its name in the source schema.

Table 52 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
corOut	variable	double	Required	"" (empty string)	Specifies the name of the output variable for the correlation between X and Y input variables.

The edge is defined at the end of the project.

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

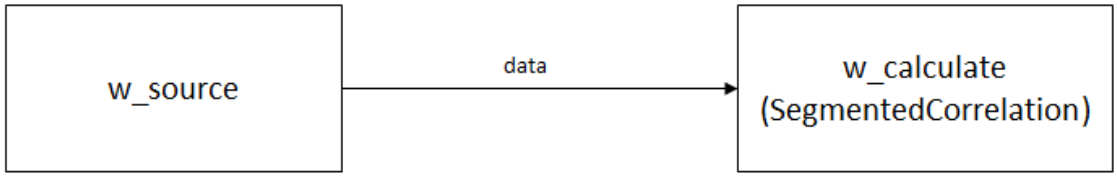
You can view the default values of the streaming correlation algorithm parameter properties for the Calculate window with the command-line utility.

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 of a variable from an incoming data stream and publishes the results

The Source window `w_source` receives a data event. 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>
     ...
     </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">4</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>
    ...
   </connectors>
  </window-calculate>
```

The segmented correlation algorithm is governed by the following properties:

Table 53 Parameters

Name	Variable Type	Required or Optional?	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. Specify the value of `minSize` to be <= the value of `sampleSize`.
`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 54 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
x	variable	double	Required	No default value	Specifies the input variable for the segmented correlation.
indicator	variable	int32	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 55 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
corOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the segmented correlation.

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.

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. For example, you could use MRR to detect electrical disturbances in the power grid.

Let X_i denote the t^{th} element of the time series. The Range and the MRR for X_i is computed as follows:

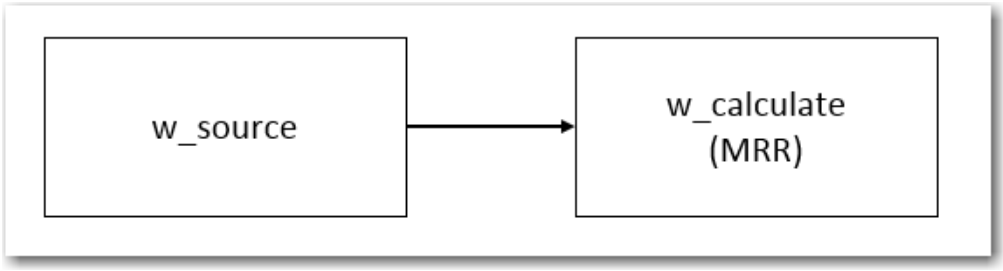
$$Range_t = Range\left(X_t,\ X_{t-1}, \ldots X_{t-M+1}\right)$$

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

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. 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>
        ...
        </connectors>
</window-source>
```

The Calculate window `w_calculate` receives data events from `w_source` and performs the moving relative range calculation using the specified algorithm parameters.

```
<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>
             ...
             </connectors>
  </window-calculate>
```

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

Table 56 *Parameters*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`rangeWindowLength`	variable	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`	variable	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 57 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`input`	variable	double	Required	No default value	Specifies the analysis variable name in the input stream.
`timeId`	variable	int64	Required	No default value	Specifies the time ID variable name in the input stream.

Table 58 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`timeIdOut`	variable	int64	Required	No default value	Specifies the time ID (key) variable name in the output stream.
`erangeOut`	variable	double	Required	No default value	Specifies the expected range variable name in the output stream.
`rangeOut`	variable	double	Optional	`""` (empty string)	Specifies the range variable name in the output stream.

The edge is defined at the end of the project.

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

You can change the values of the properties that govern the MRR algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role “request.” Then, stream a **`reconfig`** request and events that change the property values.

For example, to change the values of all of the properties for MRR:

```
i,n,1,"action","reconfig"
i,n,2,"rangeWindowLength","2"
i,n,3,"expRangeWindowLength","2"
i,n,4,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

You can view the default values of the MRR algorithm parameter properties for the Calculate window with the command-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 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 the *SAS Visual Statistics: 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. 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>
              ...
              </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>
    ...
    </connectors>
   </window-calculate>
```

The following properties govern the algorithm in the Calculate window:

Table 59 *Parameters*

Name	Value Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	3	Specifies the length of the sliding window. The overlap between consecutive windows is `windowLength` — 1.
`ClassLabels`	string-list	Optional	No default value	Specifies a comma-separated list that contains the corresponding label for each predicted probability. This parameter is required for classification models.
`labelLen`	int64	Optional	123	Specifies the length of response labels.

Table 60 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	variable(s)	double	Required	No default value	Specifies input variables. For regression models, only one input variable is required. That variable specifies the predicted response.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
					For classification models, this variable list specifies the predicted probabilities for each response class.
`response`	response variable	double | string	Required	No default value	Specifies the response variable (that is, the target variable).

Table 61 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`nOut`	variable	double	Optional	"" (empty string)	Specifies the output variable for the number of observations (N).
`nmissOut`	variable	double	Optional	"" (empty string)	Specifies the output variable for the number of missing values (NMISS).
`aseOut`	variable	double	Optional	"" (empty string)	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	double	Optional	"" (empty string)	Specifies the divisor of the average square error.
`raseOut`	variable	double	Optional	"" (empty string)	Specifies the root average square error. $\mathrm{RASE} = \sqrt{\mathrm{ASE}}$
`mceOut`	variable	double	Optional	"" (empty string)	Specifies the mean consequential error. $\mathrm{MCE} = \frac{1}{\mathrm{N}} \sum_{t_i \neq \hat{t}_i} 1$

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
mcllOut	variable	double	Optional	"" (empty string)	Specifies the multiclass log loss. $\text{logloss} = -\frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j}\,\log(p_{i,j})$
maeOut	variable	double	Optional	"" (empty string)	Specifies the mean absolute error.
rmaeOut	variable	double	Optional	"" (empty string)	Specifies the root mean absolute error.
msleOut	variable	double	Optional	"" (empty string)	Specifies the mean square logarithmic error.
rmsleOut	variable	double	Optional	"" (empty string)	Specifies the root mean square logarithmic error.

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 goodness of fit algorithm parameter properties for the Calculate window with the command-line utility.

Streaming Distribution Fitting

The distribution fitting algorithm fits a Weibull, Gamma, or Normal distribution to a variable in the incoming data stream.

The Weibull distribution is described by the following probability density function:

$$f\left(x;c,\sigma,\theta\right)=\begin{cases}\frac{c}{\sigma}\left(\frac{x-\theta}{\sigma}\right)^{c-1}\exp\left[-\left(\frac{x-\theta}{\sigma}\right)^{c}\right] & \text{for } x\geq\theta\\ 0 & \text{for } x<\theta\end{cases}$$

Here the shape parameter c > 0, the scale parameter σ > 0, and the threshold parameter is θ.

The Gamma distribution is described by the following probability density function:

$$f\left(x;\alpha,\sigma,\theta\right)=\begin{cases}\frac{1}{\Gamma(\alpha)\sigma}\left(\frac{x-\theta}{\sigma}\right)^{\alpha-1}\exp\left[-\left(\frac{x-\theta}{\sigma}\right)\right] & \text{for } x>\theta\\ 0 & \text{for } x\leq\theta\end{cases}$$

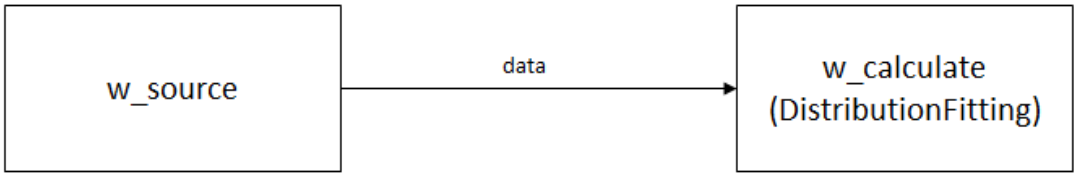
Here the shape parameter α > 0, the scale parameter σ > 0, and the threshold parameter is θ.

The Normal distribution is described by the following probability density function:

$$f\left(x;\mu,\ \sigma\right)=\frac{1}{\sigma\sqrt{2\pi}}\ \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

Here the mean is μ and the standard deviation σ > 0.

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. 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>
     ...
     </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>
          <property name="convergeOut">convergeOut</property>
         </properties>
        </output-map>
       <connectors>
       ...
       </connectors>
      </window-calculate>
```

The distribution fitting algorithm is governed by the following properties:

Table 62 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	`3`	Specifies the length of the sliding window. The value that you specify must be greater than the value you specify for `overlap`.
`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.
`beta`	double	Optional	NaN	Specifies the fixed value of β.
`alpha`	double	Optional	NaN	Specifies the fixed value of α.
`mu`	double	Optional	NaN	Specifies the fixed value of μ.
`c`	double	Optional	NaN	Specifies the fixed value of c.
`sigma`	double	Optional	NaN	Specifies the fixed value of σ.
`theta`	double	Optional	NaN	Specifies the fixed value of θ.

Table 63 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
x	variable	double	Required	No default value	Specifies the input variable for distribution fitting.

Table 64 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
betaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter β.
alphaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter α.
muOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter μ.
cOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter c.
sigmaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter σ.
thetaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter θ.
convergeOut	variable	double	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 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.

Streaming Numerical Data to Create a 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 that consists of an ID and x,y coordinates.

```
<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>
              ...
              </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>
          <property name="halfLifeSteps">2</property>
          <property name="reportInterval">10</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>
       ...
      </connectors>
</window-calculate>
```

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

Table 65 Parameters

Name	Variable Type	Required or Optional?	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.
`binRemovalThreshold`	double	Optional	`0`	Specifies the threshold for bin removal during fading mode (`alpha` < 1). Bins with heights smaller than the threshold are removed.

Name	Variable Type	Required or Optional?	Default Value	Description
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 66 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value	Specifies the input variable with which to build the histogram.

Table 67 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
binCentersOut	variable list	double	Optional	"" (empty string)	Specifies a list of output variable names for bin centers.
binHeightsOut	variable list	double	Optional	"" (empty string)	Specifies a list of output variable names for bin heights.
quantilesOut	variable list	double	Optional	"" (empty string)	Specifies a list of output variable names for quantiles.

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 histogram algorithm parameter properties for the Calculate window with the command-line utility.

Classification

Overview

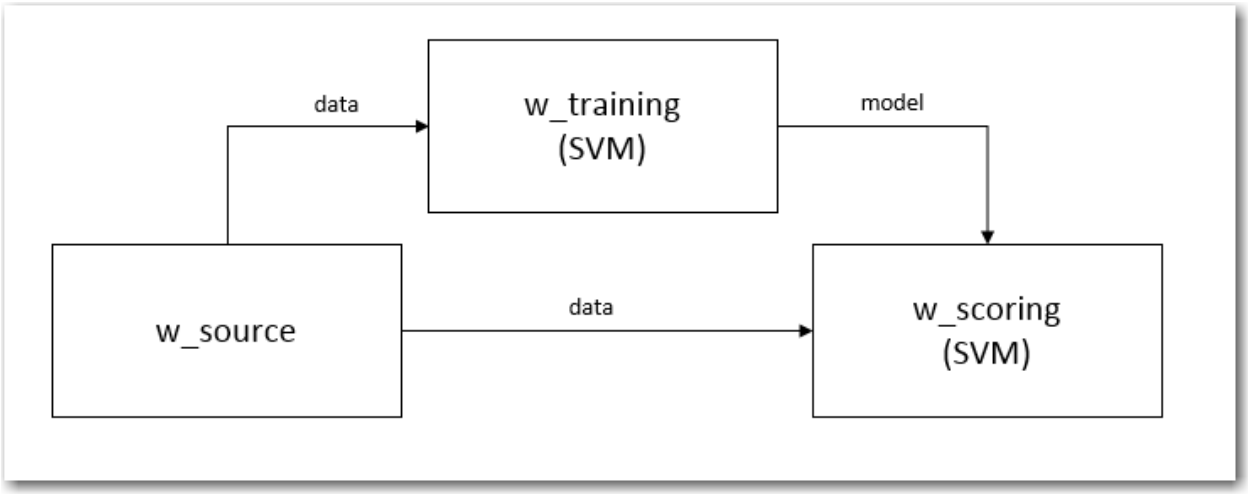
Classification algorithms are used to identify what group (class) a data point belongs to. Generally, data is classified using a set of attributes that best distinguish and define a particular class. The goal is to maximize the number of correctly labeled data points.

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. 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>
          ...
          </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="maxSparseIndex">100000</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>
              <property name="sparse">x2</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 68 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
`nInit`	int64	Optional	50	Specifies the number of data events used during initialization. Specify 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. When this is set to 0, it is reset to `nInit`.
`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 ($0 \leq \alpha \leq 1$) 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 `batchSize` data events of the initialization, and each data event is subtracted with the computed mean. When this is set to 0, it is reset to `nInit`.
`scaleFlag`	Boolean	Optional	0	Specifies whether to scale the data (dense part).

Name	Variable Type	Required or Optional?	Default Value	Description
				When this is set to 0, the data is not scaled. Otherwise, the scale vector is computed with the first `batchSize` data events during initialization so that each variable has unit length. After that, each data event is scaled with the computed scale vector.
`maxSparseIndex`	int64	Optional	1	Specifies the number of variables contained in the sparse variable, provided that it exists. Specify a nonnegative integer.
`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.
`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.

Name	Variable Type	Required or Optional?	Default Value	Description
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 69 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double | string	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).
target	variable	double | string	Optional	"" (empty string)	Specifies the target response variable. If it is not specified, then the first variable in inputs is considered as the target variable. When the target variable is missing during training, the incoming event is ignored. When it is missing during scoring, a prediction is made.
sparse	variable	string	Optional	"" (empty string)	Specifies the name of the sparse 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>
                      <property name='totalErrorOut'>totalErrorOut</property>
                      <property name='cChosenOut'>cChosenOut</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 70 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	double | string	Optional	`""` (empty string)	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 71 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`yOut`	variable	double	Optional	`""` (empty string)	Specifies the original response. If not specified, it is not displayed.
`yPredictOut`	variable	double	Optional	`""` (empty string)	Specifies the predicted response. If not specified, it is not displayed.
`modelIDOut`	variable	int64	Optional	`""` (empty string)	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.
`totalErrorOut`	variable	double	Optional	`""` (empty string)	Specifies the error between `yPredictOut` and `target`. If not specified, it is not displayed.
`cChosenOut`	variable	double	Optional	`""` (empty string)	Specifies the value of the regularization parameter whose model had the best results. 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.

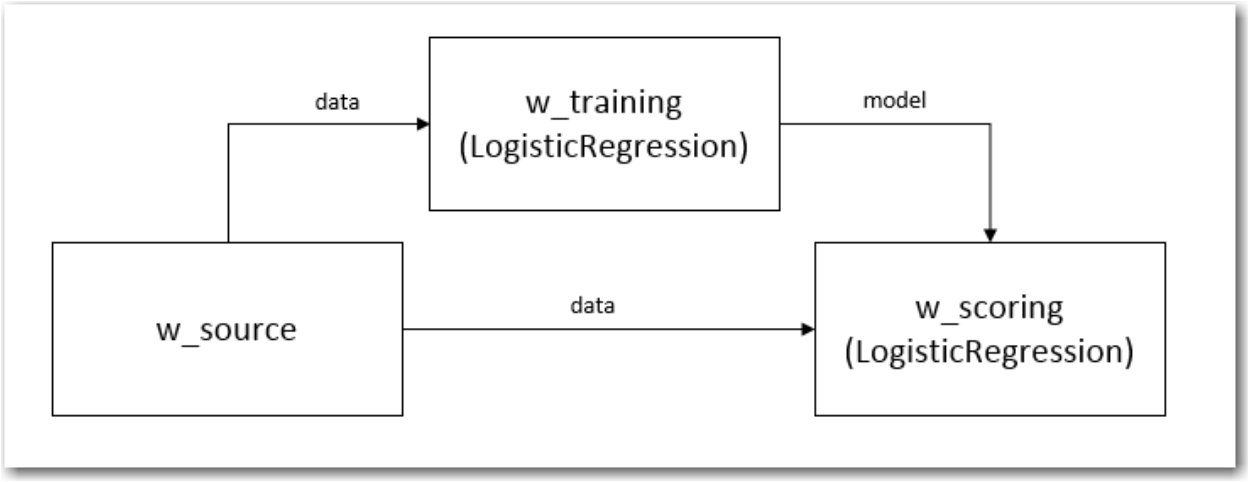
```
<edges>
            <edge source='w_data'    target='w_train' role='data'/>
            <edge source='w_data'    target='w_score' role='data'/>
            <edge source='w_train'   target='w_score' role='model'/>
          </edges>
```

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

Training and Scoring Streaming 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. When the dependent variable has more than one outcome category, it is converted to a binary classification problem by choosing a positive class and treating all other classes as one. Streaming logistic regression (LogisticRegression) is an approximation of the standard logistic regression model that is appropriate for streaming data.

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. 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>
   ...
   </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="maxSparseIndex">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 72 *Parameters*

Name	Value Type	Required or Optional?	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, then set this to the maximum of `nInit` and `commitInterval`. When this is set to 0, it is reset to `nInit` .
`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.

Name	Value Type	Required or Optional?	Default Value	Description
centerFlag	Boolean	Optional	0	Specifies whether to center the data (dense part) based on the first batchSize data events of the initialization. When this is set to 0, the data is not centered. Otherwise, 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. When this is set to 0, the data is not scaled. Otherwise, data is scaled so that the variance of the first batchSize number of data events is 1.
maxSparseIndex	int64	Optional	0	Specifies the number of variables contained in the sparse variable, if it exists. Specify a nonnegative integer.
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.
ratioC	double	Optional	10	Specifies the ratio in setting the set of regularization parameters. Specify a value greater than 1.
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

Name	Value Type	Required or Optional?	Default Value	Description
				`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 should be positive. It determines the precision of the solution with data events outside the initialization step.
`outerIterMaxInit`	int64	Optional	`outerIterMax`	Specifies the number of outer iterations used in coordinate descent for initialization data events. Specify a positive value.

Table 73 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	double | string	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y) .
`target`	variable	double | string	Optional	`""` (empty string)	Specifies the target response variable. If it is not specified, then the first variable in `inputs` is considered as the target variable. When the target variable is missing during training, the incoming event is ignored. When it is missing during scoring, a prediction is made.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
sparse	variable	string	Optional	"" (empty string)	Specifies the sparse variable that is stored in LibSVM format.

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 74 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double | string	Optional	"" (empty string)	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).

Table 75 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`yOut`	variable	double	Optional	`""` (empty string)	Specifies the original response. If not specified, it is not displayed.
`yPredictOut`	variable	double	Optional	`""` (empty string)	Specifies the predicted response. If not specified, it is not displayed.
`modelIdOut`	variable	int64	Optional	`""` (empty string)	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.
`totalErrorOut`	variable	double	Optional	`""` (empty string)	Specifies the error between `yPredictOut` and `target`. If not specified, it is not displayed.
`cChosenOut`	variable	double	Optional	`""` (empty string)	Specifies the regularization parameter whose model had the best results. 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>
```

Anomaly Detection

Overview

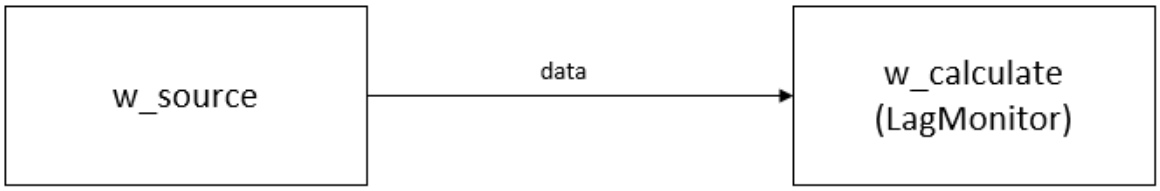
Anomaly detection is used to identify data points that are considered abnormal to the pattern of the rest of the data. Often these data points are referred to as outliers.

Lag Monitoring

The lag monitoring algorithm computes the cross-correlation between a target time series and one or more additional time series. Results contain the selected lags and computed cross-correlation values

that correspond to minimum, maximum, and maximum absolute value cross-correlations for each of the variables. The nonnegative values in the `minLag` and `maxLag` parameters control the range of lags to be computed, and both positive and negative lags are computed. The `nDigits` parameter limits the cross-correlation value comparisons to the specified number of significant figures.

Consider the following example:



Here are the contents of the continuous query:

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

The Source window `w_source` receives data that consists of an ID and a set of input variables (y1 through y7):

```
<window-source autogen-key="false" index="pi_EMPTY" insert-only="true"
name="w_source">
              <schema>
                <fields>
                  <field key="true" name="id" type="string" />
                  <field key="false" name="y1" type="double" />
                  <field key="false" name="y2" type="double" />
                  <field key="false" name="y3" type="double" />
                  <field key="false" name="y4" type="double" />
                  <field key="false" name="y5" type="double" />
                  <field key="false" name="y6" type="double" />
                  <field key="false" name="y7" type="double" />
                </fields>
              </schema>
              <connectors>
              ...
              </connectors>
            </window-source>
```

The Calculate window `w_calculate` receives data events including the values of several variables. It publishes selected lags and computed cross-correlation values that correspond to minimum, maximum, and maximum absolute value cross-correlations for each of the variables.

```
<window-calculate algorithm="LagMonitor" name="w_calculate">
              <schema>
                <fields>
                  <field key="true" name="id" type="int64" />
                  <field key="false" name="y2" type="double" />
                  <field key="false" name="y1" type="double" />
                  <field key="false" name="absCCFOut" type="double" />
                  <field key="false" name="absLagOut" type="int64" />
                  <field key="false" name="maxCCFOut" type="double" />
                  <field key="false" name="maxLagOut" type="int64" />
                  <field key="false" name="minCCFOut" type="double" />
                  <field key="false" name="minLagOut" type="int64" />
                  <field key="false" name="numComputedLagsOut" type="int64" />
```

```
    </fields>
  </schema>
  <parameters>
    <properties>
      <property name="maxLag">500</property>
      <property name="minLag">1</property>
      <property name="windowLength">10000</property>
    </properties>
  </parameters>
  <input-map>
    <properties>
      <property name="input">y2</property>
      <property name="target">y1</property>
      <property name="timeId">id</property>
    </properties>
  </input-map>
  <output-map>
    <properties>
      <property name="absCCFOut">absCCFOut</property>
      <property name="absLagOut">absLagOut</property>
      <property name="maxCCFOut">maxCCFOut</property>
      <property name="maxLagOut">maxLagOut</property>
      <property name="minCCFOut">minCCFOut</property>
      <property name="minLagOut">minLagOut</property>
      <property name="numComputedLagsOut">numComputedLagsOut</property>
      <property name="timeIdOut">id</property>
    </properties>
  </output-map>
  <connectors>
  ...
  </connectors>
</window-calculate>
```

The lag monitoring algorithm is governed by the following properties:

Table 76 *Algorithm Parameters*

Name	Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	128	Specifies the length of the sliding window. The value that you specify must be greater than the value that you specify for `overlap`.
`overlap`	int64	Optional	127	Specifies the overlap between consecutive windows. Must be less than `windowLength`.

Name	Type	Required or Optional?	Default Value	Description
`minLag`	int64	Optional	1	Specifies the minimum lag to consider.
`maxLag`	int64	Optional	10	Specifies the maximum lag to consider.
`nDigits`	int64	Optional	0	Specifies the number of significant digits to use when comparing cross-correlation values (0–16).
`correlationType`	string	Optional	Pearson	Specifies the correlation type. Options are `Pearson` and `Distance`.

Table 77 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`input`	variable	double	Required	No default value.	Specifies the analysis variable name in the input stream.
`target`	variable	double	Required	No default value.	Specifies the target variable name in the input stream.
`timeId`	variable	int64	Required	No default value.	Specifies the time ID variable name in the input stream. It must be equally spaced.

Table 78 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`timeIDout`	variable	int64	Required	No default value.	Specifies the time ID variable name in the output stream.
`minLagOut`	variable	int64	Optional	`""` (empty string)	Specifies the variable name for the lag that corresponds to minimum cross-correlation.
`minCCFOut`	variable	double	Optional	`""` (empty string)	Specifies the variable name for minimum cross-correlation.
`maxLagOut`	variable	int64	Optional	`""` (empty string)	Specifies the variable name for lag that corresponds to maximum cross-correlation.
`maxCCFOut`	variable	double	Optional	`""` (empty string)	Specifies the variable name for maximum cross-correlation.
`absLagOut`	variable	int64	Optional	`""` (empty string)	Specifies the variable name for lag that corresponds to maximum absolute cross-correlation.
`absCCFOut`	variable	double	Optional	`""` (empty string)	Specifies the variable name for maximum absolute cross-correlation.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`numComputedLagsOut`	variable	int64	Optional	`""` (empty string)	Specifies the variable name for the number of computed lags in the output stream.

The edge is defined at the end of the project.

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

You can change the values of the properties that govern the lag monitoring algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role “request.” Then, stream a **`reconfig`** request and events that change the property values.

For example, to change the values of all of the properties for lag monitoring:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","64"
i,n,3,"overlap","32"
i,n,4,"minLag","1"
i,n,5,"maxLag","32"
i,n,6,"nDigits","0"
i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

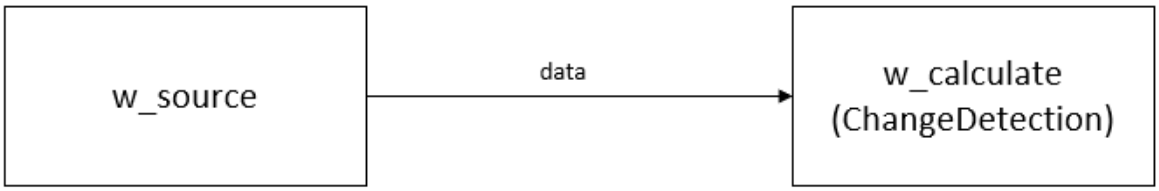
You can view the default values of the lag monitoring algorithm parameter properties for the Calculate window with the command-line utility.

Change Detection

With change detection, a stream of measures is monitored and an alert is raised when values deviate from what is expected. Often, point estimates of the measures are calculated. Then, differences between consecutive measures are inspected to determine whether they deviate from a pre-defined threshold. However, point estimates do not capture changes in the underlying distribution of the tracked measures. To capture changes in the underlying distribution, histogram intersection is used.

SAS Event Stream Processing uses Kullback-Leibler divergence (KL divergence) to detect changes through histogram intersection. KL divergence measures how one probability distribution is different from a second, reference distribution over the same variable. It can be used in the fields of applied statistics and machine learning.

Consider the following example:



The continuous query includes the following:

- a Source window that receives input data
- a Calculate window that runs the change detection algorithm on that data

Here is the code for the Source window:

```
<window-source name="w_source" insert-only="true" index="pi_EMPTY">
  <schema>
    <fields>
      <field type="int64" name="id" key="true" />
      <field type="double" name="x" />
    </fields>
  </schema>
  <connectors>
    ...
  </connectors>
</window-source>
```

The following Calculate window applies the `ChangeDetection` algorithm to that data:

```
<window-calculate name="w_calculate" algorithm="ChangeDetection">
  <schema>
    <fields>
      <field type="int64" name="id" key="true" />
      <field type="double" name="x" />
      <field type="double" name="changeVal" />
      <field type="int32" name="eval" />
      <field type="int32" name="changeDetected" />
    </fields>
  </schema>
  <parameters>
    <properties>
      <property name="maxBins">100</property>
      <property name="slidingAlpha">0.997</property>
      <property name="refWindowSize">500</property>
      <property name="maxEvalSteps">300</property>
      <property name="adaptiveEval">1</property>
      <property name="showEval">1</property>
      <property name="showAll">0</property>
    </properties>
  </parameters>
  <input-map>
    <properties>
      <property name="input">x</property>
    </properties>
  </input-map>
  <output-map>
    <properties>
      <property name="evaluatedOut">eval</property>
      <property name="changeValueOut">changeVal</property>
```

```
        <property name="changeDetectedOut">changeDetected</property>
      </properties>
    </output-map>
  </window-calculate>
```

The following parameters govern the change detection algorithm in the Calculate window:

Table 79 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
`maxBins`	int64	Optional	`50`	Specifies the maximum number of bins in the histogram for both the reference window and the sliding window.
`slidingAlpha`	double	Optional	`1.0`	Specifies the fading factor for the sliding window. Value range is 0 < α<=1.
`slidingHalfLifeSteps`	int64	Optional	`0`	Specifies the number of steps at which the weight of the input reaches half of its original weight for the sliding window.
`refWindowSize`	int64	Optional	`1000`	Specifies the size of the reference window.
`changeThreshold`	double	Optional	`0.1`	Specifies the threshold to determine whether a change occurred.
`nComparisonBins`	int64	Optional	`100`	Specifies the number of bins when computing KL-Divergence.
`maxEvalSteps`	int64	Optional	`300`	Specifies the maximum number of steps before performing a new evaluation.
`adaptiveEval`	int32	Optional	`1` (true)	Specifies whether to use the adaptive evaluation step size or not.
`showEval`	int32	Optional	`0` (false)	Specifies whether to show evaluation events regardless of whether a change is detected.
`showAll`	int32	Optional	`0` (false)	Specifies whether to show all events, regardless of whether an evaluation occurs.

Table 80 *Input Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`input`	variable	double	Required	No default value.	Specifies the input variable for change detection.

Table 81 *Output Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`evaluatedOut`	varlist	int32	Optional	`""` (empty string)	Specifies the name of the output variable that indicates whether an evaluation occurred.
`changeValueOut`	varlist	int32	Optional	`""` (empty string)	Specifies the name of the output variable that contains the change value (the difference between two KL divergence values).
`changeDetectedOut`	varlist	int32	Optional	`""` (empty string)	Specifies the name of the output variable that indicates whether a change has been detected.

The edge is defined at the end of the project.

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

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

Subspace Tracking (SST)

Suppose that data contains a sequence of *n* X 1 vectors: $x(t)$. Subspace tracking (SST) estimates the covariance matrix for each vector $x(t)$ and then computes the first *p* principal eigenvectors of the covariance matrix. For each iteration at time *t*, the covariance matrix $C(t)$ is obtained by the following:

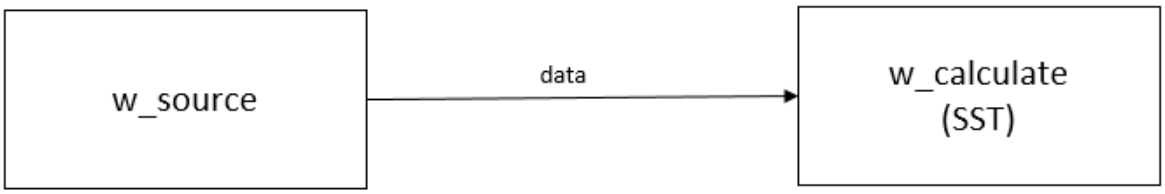
$$\mu(t) = (1-\alpha)\,\mu(t-1) + \alpha\; x(t)$$
$$C(t) = (1-\beta)\,C(t-1) + \beta(x(t) - \mu(t))(x(t) - \mu(t))^T$$

Here, α and β are the mean and covariance forgetting factors whose values are predetermined to be between 0 and 1, respectively. The first *p* principal eigenvector $W(t)$ can be obtained by the eigendecomposition of the covariance matrix.

Another method for tracking the subspace is to use the moving windows principal component analysis. In this method, there are no forget factors. Window length, however, has to be specified. For more information about window-based subspace tracking, see the MWPCA procedure in *SAS Visual Data Mining and Machine Learning: Procedures*. For more information about subspace tracking with forget factors, see the Subspace Tracking Page in *SAS Visual Forecasting: Time Series Packages*.

SST can be applied to industrial data to detect outliers and use results to identify potential errors before they occur.

Consider the following example:



Here are the contents of the continuous query:

- a Source window that receives the data to analyze
- a Calculate window that performs SST

The Source window `w_source` receives input data. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; and a series of six different `x` coordinate fields (`x1`, `x2`,`x3`,`x4`, `x5`, `x6`).

```
<window-source name='w_source'>
              <schema>
                <fields>
                   <field name='id' type='int64' key='true'/>
                   <field name='x1' type='double'/>
                   <field name='x2' type='double'/>
                   <field name='x3' type='double'/>
                   <field name='x4' type='double'/>
                   <field name='x5' type='double'/>
                   <field name='x6' type='double'/>
                </fields>
              </schema>
              <connectors>
              ...
              </connectors>
            </window-source>
```

The Calculate window `w_calculate` receives data events from `w_source`. It publishes the calculated principal components and output values of the subspace according to the SST algorithm properties that are specified.

```
<window-calculate name='w_calculate' algorithm='SST'>
          <schema>
               <fields>
                  <field name='id' type='int64' key='true'/>
                  <field name='projAngle' type='double'/>
                  <field name='residualRate' type='double'/>
                  <field name='numRank' type='int32'/>
```

```
                    <field name='prin1_x1' type='double'/>
                    <field name='prin1_x2' type='double'/>
                    <field name='prin1_x3' type='double'/>
                    <field name='prin1_x4' type='double'/>
                    <field name='prin1_x5' type='double'/>
                    <field name='prin1_x6' type='double'/>
                  </fields>
                </schema>
                <parameters>
                  <properties>
                    <property name="maxPrincipal">2</property>
                    <property name="meanForgetFactor">0.1</property>
                    <property name="covForgetFactor">0.5</property>
                    <property name="eigvalTolCumulative">1</property>
                  </properties>
                </parameters>
                <input-map>
                  <properties>
                    <property name="inputs">x1, x2, x3, x4, x5, x6</property>
                  </properties>
                </input-map>
                <output-map>
                   <properties>
                    <property name="projAngleOut">projAngle</property>
                    <property name="residualOut">residualRate</property>
                    <property name="numRankOut">numRank</property>
                    <property name="principalVecOut">prin1_x1, prin1_x2, prin1_x3,
prin1_x4, prin1_x5, prin1_x6</property>
                  </properties>
                </output-map>
                <connectors>
                ...
                </connectors>
              </window-calculate>
```

The following properties govern the SST algorithm:

Table 82 *Parameters*

Name	Value Type	Required or Optional?	Default Value	Description
`windowLength`	int64	Optional	0	Specifies the length of the sliding window. A sliding window enables you to use multiple events to update principal components. A value of 0 denotes unlimited length. If the value is greater than 0,`covForgetFactor` and `meanForgetFactor` are ignored for updating the covariance matrix. The value that you specify must be greater than the value that you specify for `overlap`.

Name	Value Type	Required or Optional?	Default Value	Description
`overlap`	int64	Optional	-1	Specifies the overlap between consecutive windows. Must be strictly less than `windowLength`. The default value of `-1` means that `overlap` is internally calculated as `windowLength` -1.
`maxPrincipal`	int32	Optional	1	Specifies the maximum number of the principal eigenvectors. Specify a value greater than 0 and less than or equal to the number of input variables.
`covForgetFactor`	double	Optional	0.5	Specifies the forgetting factor that is used to update the covariance matrix. Specify a value between 0 and 1.
`meanForgetFactor`	double	Optional	0.1	Specifies the value of the forgetting factor used to update the mean. Specify a value between 0 and 1.
`eigvalTolCumulative`	double	Optional	1	Specifies the threshold on the cumulative rate of eigenvalues. Specify a positive value less than or equal to 1.

Table 83 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	double	Required	No default value	Specifies the list of variable names used to compute the principal subspace.

Table 84 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`PCAngleChangeOut`	variable	double	Optional	`""` (empty string)	Specifies the output variable name for the angle change between the first principal component vector of two consecutive subspaces.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
PCAbsoluteAngleOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the absolute angle of the first principal component vector.
projAngleOut	variable	double	Optional	"" (empty string)	Specifies the output variable name of the projection angle.
numRankOut	variable	int32	Optional	"" (empty string)	Specifies the output variable name of the rank of the principal subspace.
principalVecOut	varlist	double	Optional	"" (empty string)	Specifies a list of variable names that correspond to the elements of each principal eigenvector. Suppose that the data consists of five variables: x1, x2, x3, x4, and x5. If you want to produce the third principal eigenvector, prin3, then you must specify the variable names as follows: prin1_x1, prin1_x2, prin1_x3, prin1_x4, prin1_x5, prin2_x1, prin2_x2, prin2_x3, prin2_x4, prin2_x5, prin3_x1, prin3_x2, prin3_x3, prin3_x4, and prin3_x5. That is, you must specify all of the elements of the first and second principal eigenvector to produce the elements of the third. Often, the first principal eigenvector provides the most useful information.
residualOut	variable	double	Optional	"" (empty string)	Specifies the output variable name of the residual.
projectionVecOut	varlist	double	Optional	"" (empty string)	Specifies the output variable name of the vector projected into the principal subspace.

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 SST algorithm parameter properties for the Calculate window with the command-line utility.

You can change the values of the properties that govern the SST algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role “request.” Then, stream a **reconfig** request and events that change the property values.

For example, to change the values of all of the properties for SST:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","64"
i,n,3,"maxPrincipal","2"
i,n,4,"covForgetFactor","1"
i,n,5,"meanForgetFactor","1"
i,n,6,"eigvalTolCumulative","0.75"
i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

Clustering

Overview

Clustering algorithms are best used for tasks involving grouping data objects into clusters that exhibit the most similarity. The goal is to maximize on distinctness between similar objects in a cluster and differences between clusters.

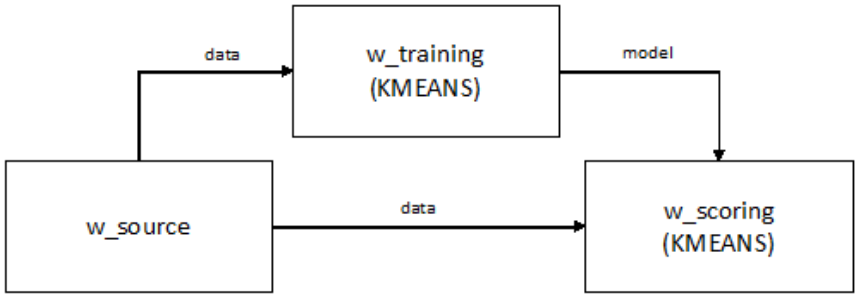
Training and Scoring with K-means Clustering

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.

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. 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>
      ...
      </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 85 Parameters

Name	Variable Type	Required or Optional?	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.
`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 86 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	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 87 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Optional	"" (empty string)	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 88 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`labelOut`	variable	int32	Optional	`""` (empty string)	Specifies the output variable in the output schema that stores the cluster label.
`minDistanceOut`	variable	double	Optional	`""` (empty string)	Specifies the output variable in the output schema that stores the distance to the nearest cluster. If not specified, the minimal distance column is not shown.
`modelIdOut`	variable	int64	Optional	`""` (empty string)	Specifies the output variable 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.

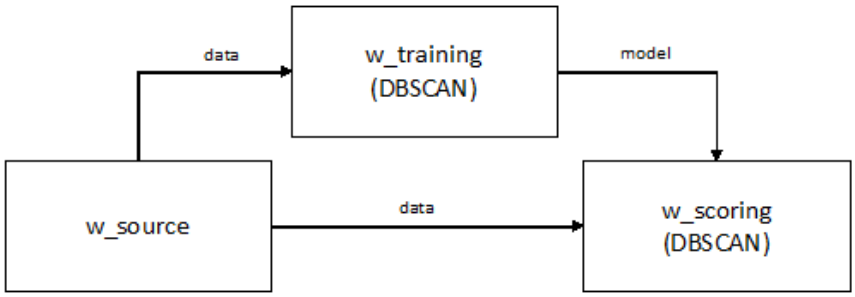
```
  <edges>
  <edge source='w_source' target='w_training' role='data'/>
  <edge source='w_source' target='w_scoring' role='data'/>
  <edge source='w_training' target='w_scoring' role='model'/>
</edges>
```

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.

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. 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>
      ...
      </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 89 *Parameters*

Name	Variable Type	Required or Optional?	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 how many timestamps should elapse before sending a model to downstream scoring.

Table 90 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	varlist	double	Required	No default value	Specifies the list of variables used in clustering. Variables 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 91 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Optional	"" (empty string)	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 92 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`labelOut`	variable	int32	Optional	"" (empty string)	Specifies the output variable name in the output schema that stores the cluster label.
`minDistanceOut`	variable	double	Optional	"" (empty string)	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	int64	Optional	"" (empty string)	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.

Regression

Regression algorithms are used to examine relationships between variables. Most commonly, this entails estimating the value of a dependent variable based on the values of independent variables that are hypothesized to have an impact on the dependent variable.

Training and Scoring with Streaming Linear Regression

Linear regression models the relationship between a scalar dependent variable and one or more explanatory variables (independent variables). Streaming linear regression (LinearRegression) is an approximation of the standard linear regression model that is appropriate for streaming data.

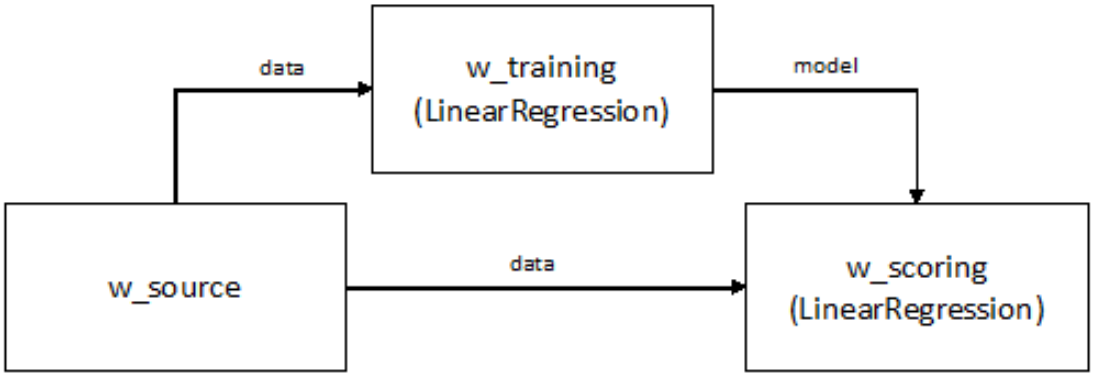
The basic linear regression model is $Y = X\beta + e$ Here, Y is the target vector, X is the data matrix, β is the vector of parameters, and e is the vector of errors. For a static data set, one way to solve for β is to use coordinate descent. After solving for β, you obtain an estimated parameter vector $\widehat{\beta}$. You can then use that vector with any new X data matrix in order to make predictions for the values of Y, that is $\widehat{Y}$.

In the streaming linear regression model, you must make predictions based on an ever-changing stream of data. After you obtain a specified number of events, `ninit`, you solve for the βs. When you reach a number of events that is equal to the `commitInterval`, you publish the most recent βs to the Score window. The Score window uses them to score all of the data published to it.

Now you have a model, but more events are streaming in. Use `batchSize` to define how many new events are required before you update the model. After the number new events is equal to `batchSize`, solve for the βs again. You do not use all of the data from the beginning, as this could get prohibitively large with time. Instead, you essentially combine the existing βs with the βs obtained from training the new batch of data.

Use the `dampingFactor` to define how much the old βs affect the new βs. After the number of new events since the last model was published to score window equals the `commitInterval`, you publish the most recent set of βs the Score window. Use the βs to score all data passed to the window. Repeat this process as needed. Instead of storing all the data and repeatedly training the data statically, you store only the previous parameter's β. This is the general goal of streaming algorithms, find a way to not store all previous data but remember what you learned from it to score data.

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. 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>
```

```
        ...
        </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 93 *Parameters*

Name	Value Type	Required or Optional?	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`.

Name	Value Type	Required or Optional?	Default Value	Description
`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`	int64	Optional	0 (false)	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 `batchSize` data events of the initialization, and each data event is subtracted with the computed mean.
`scaleFlag`	int64	Optional	0 (false)	Specifies whether to scale the data (dense part) based on the first `batchSize` data events of the initialization so that each variable has unit length. After scaling the dense part, each data event is scaled with the computed scale vector.
`maxSparseIndex`	int64	Optional	0	Specifies the number of predictor variables contained in the sparse variable, if it exists. The value should be a nonnegative integer. **Note:** Sparse linear regression problems generate dense matrix calculations. These calculations often cannot be completed fast enough for streaming data applications. Thus, it is recommended to set a small value of `maxSparseIndex`.

Table 94 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional ?	Default Value	Description
`inputs`	varlist	double | string	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
					separated by a comma in the list. All input variables must be specified.
`target`	variable	double	Optional	`""` (empty string)	Specifies the target response variable. If it is not specified, the first variable in `inputs` is considered as the target variable. For linear regression, the `target` must be a continuous variable. When the target variable is missing during training, the incoming event is ignored. When it is missing during scoring, a prediction is made.
`sparse`	variable	string	Optional	`""` (empty string)	Specifies the sparse 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 95 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional ?	Default Value	Description
`inputs`	varlist	double | string	Optional	`""` (empty string)	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by comma in the list (for example, x,y).

Table 96 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional ?	Default Value	Description
`yOut`	variable	double	Optional	`""` (empty string)	Specifies the original response. If not specified, it is not displayed.
`yPredictOut`	variable	double	Optional	`""` (empty string)	Specifies the predicted response. If not specified, it is not displayed.
`modelIdOut`	variable	int64	Optional	`""` (empty string)	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>
```

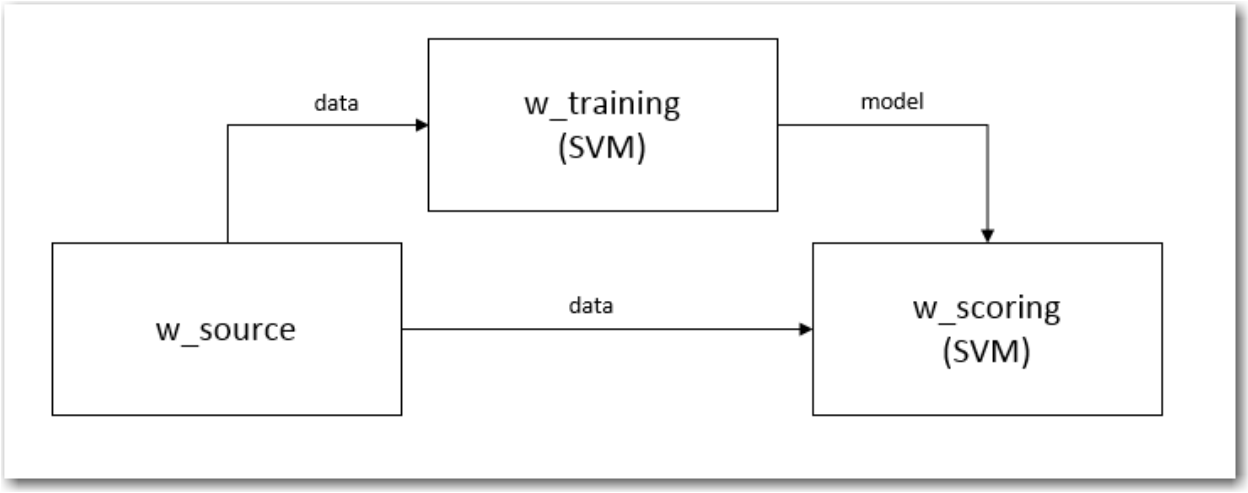
You can view the default values of the logistic regression algorithm parameter properties for the Score and Train windows with the command-line utility.

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. 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>
              ...
              </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="maxSparseIndex">100000</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>
        <property name="sparse">x2</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 97 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
`nInit`	int64	Optional	50	Specifies the number of data events used during initialization. Specify 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. When this is set to 0, it is reset to `nInit`.
`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

Name	Variable Type	Required or Optional?	Default Value	Description
				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 `batchSize` data events of the initialization, and each data event is subtracted with the computed mean. When this is set to 0, it is reset to `nInit`.
`scaleFlag`	Boolean	Optional	0	Specifies whether to scale the data (dense part). When this is set to 0, the data is not scaled. Otherwise, the scale vector is computed with the first `batchSize` data events during initialization so that each variable has unit length. After that, each data event is scaled with the computed scale vector.
`maxSparseIndex`	int64	Optional	1	Specifies the number of variables contained in the sparse variable, provided that it exists. Specify a nonnegative integer.
`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.

Name	Variable Type	Required or Optional?	Default Value	Description
`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.
`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 98 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	double | string	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).
`target`	variable	double | string	Optional	`""` (empty string)	Specifies the target response variable. If it is not specified, then the first variable in `inputs` is considered as the target variable. When the target variable is missing during training, the incoming event is ignored. When it is missing during scoring, a prediction is made.
`sparse`	variable	string	Optional	`""` (empty string)	Specifies the name of the sparse 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>
                      <property name='totalErrorOut'>totalErrorOut</property>
```

```
            <property name='cChosenOut'>cChosenOut</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 99 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	double | string	Optional	`""` (empty string)	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 100 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`yOut`	variable	double	Optional	`""` (empty string)	Specifies the original response. If not specified, it is not displayed.
`yPredictOut`	variable	double	Optional	`""` (empty string)	Specifies the predicted response. If not specified, it is not displayed.
`modelIDOut`	variable	int64	Optional	`""` (empty string)	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.
`totalErrorOut`	variable	double	Optional	`""` (empty string)	Specifies the error between `yPredictOut` and `target`. If not specified, it is not displayed.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
cChosenOut	variable	double	Optional	"" (empty string)	Specifies the value of the regularization parameter whose model had the best results. 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.

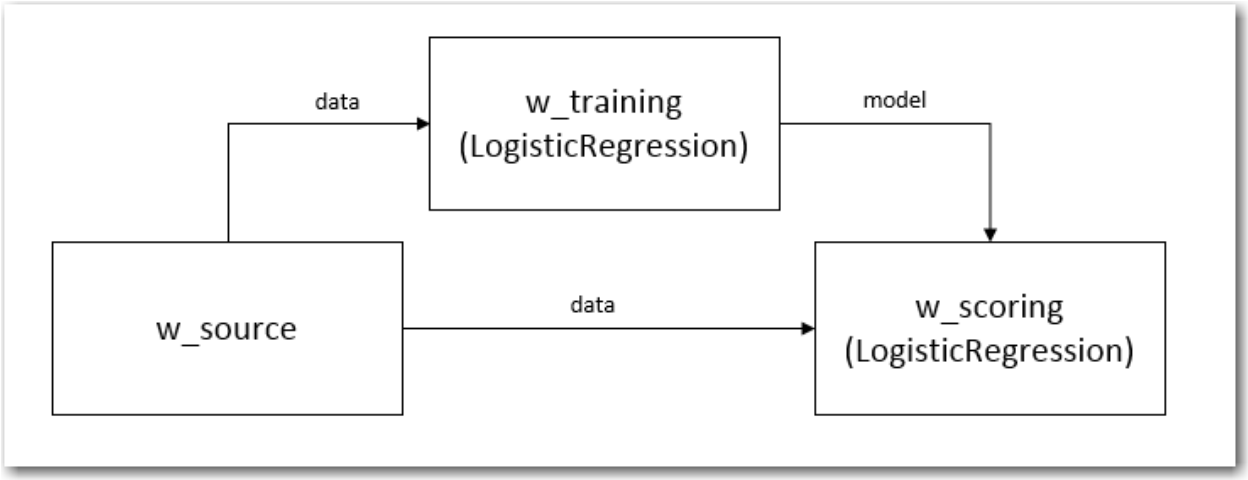
```
<edges>
            <edge source='w_data'    target='w_train' role='data'/>
            <edge source='w_data'    target='w_score' role='data'/>
            <edge source='w_train'   target='w_score' role='model'/>
          </edges>
```

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

Training and Scoring Streaming 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. When the dependent variable has more than one outcome category, it is converted to a binary classification problem by choosing a positive class and treating all other classes as one. Streaming logistic regression (LogisticRegression) is an approximation of the standard logistic regression model that is appropriate for streaming data.

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. 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>
   ...
   </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="maxSparseIndex">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 101 Parameters

Name	Value Type	Required or Optional?	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, then set this to the maximum of `nInit` and `commitInterval`. When this is set to 0, it is reset to `nInit` .
`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. When this is set to 0, the data is not centered. Otherwise, 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. When this is set to 0, the data is not scaled. Otherwise, data is scaled so that the variance of the first

Name	Value Type	Required or Optional?	Default Value	Description
				`batchSize` number of data events is 1.
`maxSparseIndex`	int64	Optional	0	Specifies the number of variables contained in the sparse variable, if it exists. Specify a nonnegative integer.
`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.
`ratioC`	double	Optional	10	Specifies the ratio in setting the set of regularization parameters. Specify a value greater than 1.
`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	Specifies the number of outer iterations used in coordinate descent for non-initialization data events. The specified value should be positive. It determines the precision of the solution with data events outside the initialization step.

Name	Value Type	Required or Optional?	Default Value	Description
outerIterMaxInit	int64	Optional	outerIterMax	Specifies the number of outer iterations used in coordinate descent for initialization data events. Specify a positive value.

Table 102 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double | string	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y) .
target	variable	double | string	Optional	"" (empty string)	Specifies the target response variable. If it is not specified, then the first variable in inputs is considered as the target variable. When the target variable is missing during training, the incoming event is ignored. When it is missing during scoring, a prediction is made.
sparse	variable	string	Optional	"" (empty string)	Specifies the sparse variable that is stored in LibSVM format.

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 103 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	double | string	Optional	`""` (empty string)	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).

Table 104 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`yOut`	variable	double	Optional	`""` (empty string)	Specifies the original response. If not specified, it is not displayed.
`yPredictOut`	variable	double	Optional	`""` (empty string)	Specifies the predicted response. If not specified, it is not displayed.
`modelIdOut`	variable	int64	Optional	`""` (empty string)	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.
`totalErrorOut`	variable	double	Optional	`""` (empty string)	Specifies the error between `yPredictOut` and `target`. If not specified, it is not displayed.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
cChosenOut	variable	double	Optional	"" (empty string)	Specifies the regularization parameter whose model had the best results. 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>
```

Media Processing

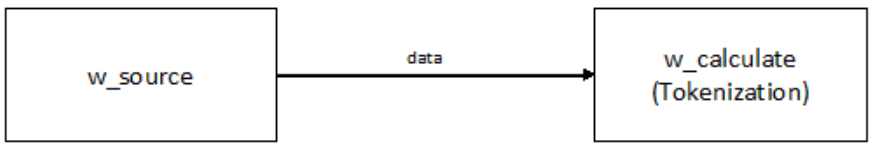
Overview

Media processing refers to algorithms applied to unstructured, visual data. This includes processing, parsing, and manipulating videos, images, audio, and text.

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. 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>
      ...
      </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>
      ...
      </connectors>
     </window-calculate>
```

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

Table 105 *Parameters*

Name	Type	Required or Optional?	Default Value	Description
`language`	string	Optional	`""` (empty string)	The following logic determines the value of this parameter: ■ If `language` is set by the user, use that language to get the text binary ■ If `language` is not set, check the locale environment variable `"LANG"` and use its first two characters

Name	Type	Required or Optional?	Default Value	Description
				(which represent the language) to get the text binary. If `LANG` is not set, fall back to `Universal`.

Table 106 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`doc`	variable	int64	Required	No default value	Specifies the input variable for the input document from the Source window.
`docId`	variable	string	Required	No default value	Specifies the input variable for the unique document ID.

Table 107 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`docIdOut`	variable	int64	Required	No default value	Specifies the output variable for the unique doc ID.
`tokenIdOut`	variable	int64	Required	No default value	Specifies the output variable for the unique ID of the token.
`wordOut`	variable	string	Required	No default value	Specifies the output variable for the word content in the token.
`startPosOut`	variable	int32	Optional	"" (empty string)	Specifies the output variable for the starting position of the token word.
`endPosOut`	variable	int32	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 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 a 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 that consists of a document ID and the name of a token.

```
<window-source name='w_source'>
      <schema>
         <fields>
            <field name='docId'   type='int64' key='true'/>
            <field name='tokenId' type='int64' key='true'/>
         </fields>
       </schema>
       <connectors>
          ...
       </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'/>
                </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</property>
        </properties>
      </output-map>
    </window-calculate>
  </windows>
```

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

Table 108 *Parameters*

Name	Value Type	Required or Optional?	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	**`""`** (empty string)	Specifies the filename of the start list, which contains the words that are considered during vectorization.
`stopList`	string	Optional	**`""`** (empty string)	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 109 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
docID	variable	int64	Optional	"" (empty string)	Specifies the input variable name of a document ID. It is required when outputDocVec is set to nonzero.
token	variable	string	Required	No default value	Specifies the input variable name of a token.

Table 110 Output Mapping

Name	Value Type	Variable Type	Required or Optional	Default Value	Description
docIDOut	variable	int64	Optional	"" (empty string)	Specifies the output variable name of a document ID. It is required when outputDocVec is set to nonzero.
vectorOut	variable	string	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>
```

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

Using Term Frequency — Inverse Document Frequency (TFIDF)

Term frequency — inverse document frequency, or TFIDF, is a weight that shows how important a word is to a document in a document collection. TFIDF increases proportionally to the frequency with which a word appears in a document, but it is offset by the frequency with which the word appears in the document collection. You can use TFIDF as a weighing factor in text mining or general information searches.

TFIDF is composed by two terms:

- Term Frequency (TF), which measures how frequently a term occurs in a document. A term could appear more frequently in long documents than in short ones. Thus, TF is often normalized by

dividing the number of times that a particular term occurs by the total number of terms in the document.

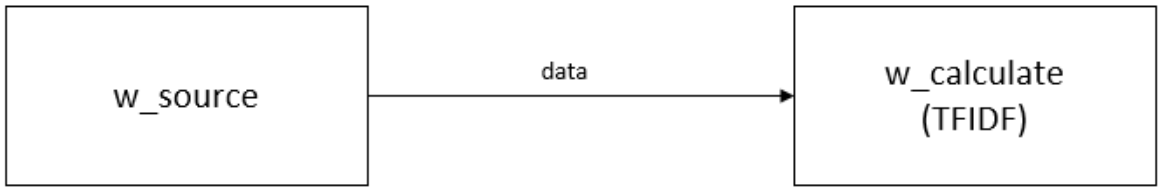
- Inverse Document Frequency (IDF), which measures the importance of a term. Some terms might occur a lot of times (for example, “is,” and “or”) but have little importance. IDF scales up rare terms and weighs down frequent terms.

$$\mathrm{IDF}(t) = \log(\text{total number of documents} / \text{number of documents that contain term } t + 1)$$

TFIDF is as follows:

$$TFIDF(t) = TF(t) * IDF(t)$$

Consider the following example:



At the project level, a single continuous query includes the following:

- a Source window that receives output from a Calculate window that produced text tokenization results
- a Calculate window that runs the TFIDF algorithm

The Source window `w_source` receives input data. The input stream is placed into three fields for each observation: two key fields named `docId` and `tokenId` and a string named `token`.

```
<window-source name='w_source'>
   <schema>
      <fields>
         <field name='docId'  type='int64' key='true'/>
         <field name='tokenId' type='int64' key='true'/>
         <field name='token'  type='string' key='false'/>
      </fields>
   </schema>
   <connectors>
   ...
   </connectors>
</window-source>
```

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

```
<window-calculate algorithm="TFIDF" name="w_calculate">
    <schema>
       <fields>
          <field key="true" name="docId" type="int64"/>
          <field key="true" name="tokenId" type="int64"/>
          <field key="false" name="token" type="string"/>
          <field key="false" name="tf" type="double"/>
          <field key="false" name="idf" type="double"/>
          <field key="false" name="tfidf" type="double"/>
       </fields>
   </schema>
```

```
    <input-map>
       <properties>
          <property name="docId">docId</property>
          <property name="tokenId">tokenId</property>
          <property name="token">token</property>
       </properties>
    </input-map>
    <output-map>
       <properties>
          <property name="docIdOut">docIdOut</property>
          <property name="tokenIdOut">tokenIdOut</property>
          <property name="tokenOut">token</property>
          <property name="tfOut">tf</property>
          <property name="idfOut">idf</property>
          <property name="tfidfOut">tfidf</property>
       </properties>
    </output-map>
    <connectors>
     ...
    </connectors>
  </window-calculate>
```

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

Table 111 Parameters

Parameter	Type	Required or Optional?	Default Value	Description
`startList`	string	Optional	`""` (empty string)	Specifies the file name of the start list, which contains the words that are evaluated during vectorization. The start list is a text file that lists one word per line. The start list is required whenever the dense vector is produced. The order of the words in the start list corresponds to the order that the TFIDF is represented in the vector output.
`stopList`	string	Optional	`""` (empty string)	Specifies the file name of the stop list, which contains words that are ignored. The stop list is a text file that lists one word per line. **Note:** When a word exists in both the start list and the stop list, the word is ignored.
`outputDenseVec`	int64	Optional	`""` (empty string)	Specifies whether word vectors or document vectors are returned. When set to 0, word vectors are returned. When set to 1, document vectors are returned.
`outputVecType`	string	Optional	`""` (empty string)	Specifies whether term frequency (TF) or term frequency — inverse document

Parameter	Type	Required or Optional?	Default Value	Description
				frequency (TFIDF) is written to the document vectors.

Table 112 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`docId`	variable	int64	Required	No default value	Specifies the input variable for the unique doc ID (key).
`tokenId`	variable	int64	Required	No default value	Specifies the input variable for the token ID (key).
`token`	variable	string	Required	No default value	Specifies the input token string.

Table 113 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`docIDOut`	variable	int64	Required	No default value	Specifies the output variable for the unique doc ID (key).
`tokenIDOut`	variable	int64	Optional	`""` (empty string)	Specifies the output variable for the unique ID of the token. It is a key when the word vectors are output. It is not a key when the document vectors are output.
`tfOut`	variable	double	Optional	`""` (empty string)	Specifies the output variable for

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
					the term frequency (TF).
idfOut	variable	double	Optional	"" (empty string)	Specifies the output variable for the inverse document frequency (IDF).
tfidfOut	variable	double	Optional	"" (empty string)	Specifies the output variable for the TFIDF.
vectorOut	variable	varlist	Optional	"" (empty string)	Specifies a list of output variable names for document vectors.
tokenOut	variable	string	Optional	"" (empty string)	Specifies the output variable for the token.

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>
```

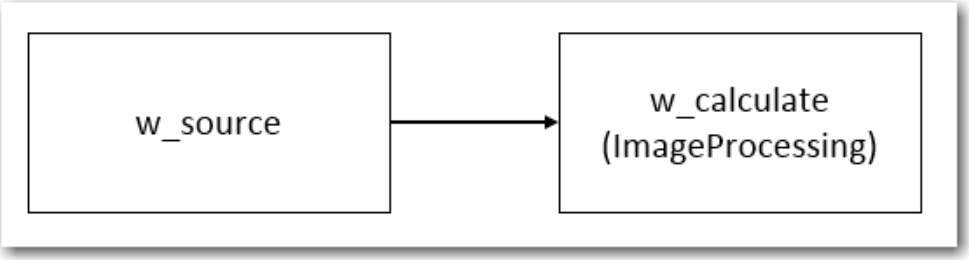
You can view the parameters and the input and output mapping properties required to set up a streaming TFIDF project with the command-line utility.

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`.

Note: For a list of the supported image formats, see the table of image formats supported by the loadImages action of the Image action set in the *SAS Visual Data Mining and Machine Learning: Programming Guide*.

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 114 *Parameters*

Name	Value Type	Required or Optional?	Default Value	Description
`function`	string	Required	`"resize"`	Specifies the image processing function to be applied: `resize`, `crop`, `rotate`, or `flip`.
`coordType`	string	Optional	`"RECT"`	Specifies the type of coordinates used for images processing. Valid values are `"COCO"`, `"RECT"`, and `"YOLO"`.
`preFlip`	int32	Optional	-1000	Specifies whether the input image is flipped before processing. This is used for video streaming. A value of `-1000` indicates no flipping. A value of `0` indicates vertical flipping. A value of `1` indicates horizontal flipping. A value of `-1` indicates horizontal and vertical flipping.
`x`	double	Optional	0	Specifies the x location. Useful for the `crop` function with `RECT` or `YOLO` coordinates.

Name	Value Type	Required or Optional?	Default Value	Description
`y`	double	Optional	0	Specifies the y location. Useful for the `crop` function with `RECT` or `YOLO` coordinates.
`xMin`	double	Optional	0	Specifies the lower x location. Useful for the `crop` function with `COCO` coordinates.
`yMin`	double	Optional	0	Specifies the lower y location. Useful for the `crop` function with `COCO` coordinates.
`xMax`	double	Optional	0	Specifies the upper x location. Useful for the `crop` function with `COCO` coordinates.
`yMax`	double	Optional	0	Specifies the upper y location. Useful for the `crop` function with `COCO` coordinates.
`width`	double	Optional	100	Specifies the width of an image. Useful for the `crop` function with `RECT` or `YOLO` coordinates and the`resize` function.
`height`	double	Optional	100	Specifies the height of an image. Useful for the `crop` function with `RECT` or `YOLO` coordinates and the`resize` function.
`outputWidth`	int32	Optional	0	Specifies the output width of an image. Useful for the `crop` function.
`outputHeight`	int32	Optional	0	Specifies the output height of an image. Useful for the `crop` function.
`theta`	double	Optional	0	Specifies the theta parameter (the rotation angle when you use the `rotate` function).
`type`	double	Optional	0	Specifies the flip type. A value of `0` indicates vertical flipping. A value of `1` indicates horizontal flipping. A value of `-1` indicates horizontal and vertical flipping.

Note: If a fixed value for a cropping parameter is provided in the XML file as well as a streaming value for the same parameter in the input mapping of the Calculate window, the streaming value takes precedence and is used in calculation.

Table 115 *Input Mapping*

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
`imageInput`	variable	blob	Required	No default value	Specifies the input image.
`xVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the x variable in the input stream.
`yVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the y variable in the input stream.
`xMinVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the xMin variable in the input stream.
`yMinVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the yMin variable in the input stream.
`xMaxVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the xMax variable in the input stream.
`yMaxVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the yMax variable in the input stream.
`widthVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the width variable in the input stream.
`heightVar`	variable	double	Optional	`""` (empty string)	Specifies the name of the height variable in the input stream.

Table 116 *Output Mapping*

Name	Value Type	Variable Type	Required or Optional	Default Value	Description
`imageOutput`	variable	blob	Optional	`""` (empty string)	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>
```

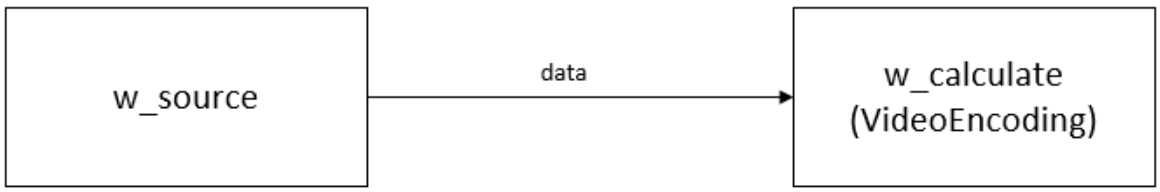
You can view the default values of the image processing algorithm parameter properties for the Calculate window with the command-line utility.

Video Encoding

The video–encoding algorithm parses image data in BayerRG8 format to a more common image format. JPEG is the default output format, and PNG is also supported.

Note: This encoder supports only data streamed from the Pylon Publisher adapter.

Consider the following example:



The continuous query includes the following:

- a Source window that receives a live video feed from a camera adapter (Pylon Publisher)
- a Calculate window that runs the video–encoding algorithm

The Source window `w_source` receives a data event that consists of video data in blob format. It receives the event through a Pylon connector.

```
<window-source index="pi_EMPTY" insert-only="true" name="w_source">
              <schema copy-keys="false">
                <fields>
                  <field key="true" name="id" type="int64" />
                  <field key="false" name="image" type="blob" />
                </fields>
              </schema>
              <connectors>
                <connector class="pylon" name="pub" type="publish">
                  <properties>
                    <property name="cameraheight">224</property>
                    <property name="cameraipaddress">10.40.19.48</property>
                    <property name="camerawidth">200</property>
                    <property name="maxframerate">1</property>
                  </properties>
                </connector>
              </connectors>
            </window-source>
```

The Calculate window `w_calculate` receives data events that consist of image data. It publishes encoded images.

```
<window-calculate algorithm="VideoEncoding" name="w_calculate">
              <schema copy-keys="false">
                <fields>
                  <field key="true" name="id" type="int64" />
                  <field key="false" name="image" type="blob" />
                  <field key="false" name="encodedImage" type="blob" />
                </fields>
```

```
    </schema>
    <parameters>
      <properties>
        <property name="height">224</property>
        <property name="inputFormat">bayerrg8</property>
        <property name="outputColorFormat">rgb</property>
        <property name="outputFormat">wide</property>
        <property name="width">225</property>
      </properties>
    </parameters>
    <input-map>
      <properties>
        <property name="videoBuffer">image</property>
      </properties>
    </input-map>
    <output-map>
      <properties>
        <property name="imageOut">encodedImage</property>
      </properties>
    </output-map>
  </window-calculate>
```

BayerRG8 is a common video–encoding format not suitable for image processing. A Calculate window can convert BayerRG8 to PNG, JPG, or wide. JPEG provides fast but lossy image compression. PNG provides lossless image compression, but it is slower than JPEG. Wide specifies no image compression. After converting the video encoding to one of these formats, you can perform object detection or image classification on the resulting image.

The width and height of the encoding window should exactly match the specifications of the adapter. In other words, if the camera is feeding a 600x800 video image, then the width and height parameters should be set to 600 and 800.

The video–encoding algorithm is governed by the following properties:

Table 117 *Parameters*

Name	Type	Required or Optional?	Default Value	Description
`inputFormat`	string	Required	**`bayerrg8`**	Specifies the input format of the source feed.
`outputColorFormat`	string	Optional	**`rgb`**	Specifies the output color space. Valid values are **`rgb`** and **`bgr`**.
`outputFormat`	string	Optional	**`jpg`**	Specifies the type of output compression. Other values are: ■ **`wide`**: no compression. This is the fastest format. ■ **`png`**; lossless compression. Slower than **`jpg`**.
`width`	int64	Required	0	Specifies the width of the input video buffer. Specify a size to match the size specified in the connector in the Source window.

Name	Type	Required or Optional?	Default Value	Description
height	int64	Required	0	Specifies the height of the input video buffer. Specify a size to match the size specified in the connector in the Source window.

Table 118 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
videoBuffer	variable	blob	Required	No default value	Specifies the name of the input video buffer

Table 119 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
imageOut	variable	blob	Required	No default value	Specifies the name of the output image

The edge is defined at the end of the project.

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

You can view the default values of the video–encoding algorithm parameter properties for the Calculate window with the command-line utility.

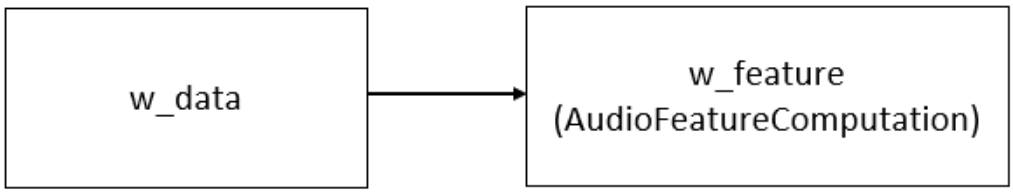
Streaming Audio Feature Computation

Audio feature computation is the process of applying an algorithm to an audio signal in order to convert it into a sequence of acoustic feature vectors. These vectors contain a serviceable numerical representation of the audio signal. You can perform further analytics using these acoustic feature vectors as input.

Note: The audio feature computation algorithm is supported on Linux (x86 architecture) and Microsoft Windows systems.

Consider the following example:

Figure 3 *Applying the Audio Feature Computation Algorithm*



This continuous query includes the following:

- a Source window that provides audio data to be analyzed
- a Calculate window that processes the audio data and converts it into a sequence of feature vectors

The Source window `w_data` creates streaming data events based on input data from a CSV file.

- The first field (`_path_`) denotes where the file originated. It might be useful to propagate this value along the entire processing chain. For example, in a speech-to-text application, it might be useful to maintain the connection between the final transcripts and the source audio files.
- The second field (`audio`) references a binary large object (blob) that comprises the audio in the current streaming data event.

```
<window-source name='w_data' insert-only='true' index='pi_EMPTY'>
   <schema>
      <fields>
         <field name='_path_' type='string' key='true'/>
         <field name='audio' type='blob'/>
      </fields>
   </schema>
   <connectors>
   ...
   </connectors>
</window-source>
```

The Calculate window `w_feature` receives those data events and processes the audio data with the audio feature computation algorithm. The parameters chosen enable you to work with Mel Frequency Cepstral Coefficient (MFCC) features that consist of 40 coefficients for each 25 millisecond long frame.

```
<window-calculate name="w_feature" output-insert-only='true' produces-only-
inserts='true'
        index='pi_EMPTY' algorithm="AudioFeatureComputation">
       <schema>
          <fields>
          <field name='_path_' type='string' key='true'/>
          <field name='_num_frames_' type='int64'/>
          <field name='f' type='array(dbl)'/>
       </fields>
   </schema>
   <parameters>
      <properties>
         <property name="frameExtractionFrameShift">10</property>
         <property name="frameExtractionFrameLength">25</property>
         <property name="frameExtractionDither">0</property>
         <property name="melBanksNBins">40</property>
         <property name="mfccNCeps">40</property>
```

```
            <property name="featureScalingMethod">STANDARDIZATION</property>
            <property name="nOutputFrames">3500</property>
         </properties>
      </parameters>
      <input-map>
         <properties>
            <property name="audioIn">audio</property>
         </properties>
      </input-map>
      <output-map>
         <properties>
            <property name="numFramesOut">_num_frames_</property>
            <property name="computedFeatureValuesOut">f</property>
         </properties>
      </output-map>
      <connectors>
      ...
      </connectors>
   </window-calculate>
```

The following properties govern the streaming audio feature computation algorithm:

Table 120 Parameters

Parameter	Type	Required or Optional?	Default Value	Description
`frameExtractionframeShift`	double	Optional	`10`	Specifies the time difference (in milliseconds) between the beginning of consecutive frames. This value must be greater than `0`.
`frameExtractionframeLength`	double	Optional	`25`	Specifies the length of a frame (in milliseconds). This value must be greater than `0`.
`frameExtractionDither`	double	Optional	`1.0`	Specifies the dithering constant. A value of `0.0` means no dithering. This value must be greater than or equal to `0`.
`frameExtractionPreemphCoeff`	double	Optional	`0.97`	Specifies the coefficient used in performing signal preemphasis. This value must be greater than or equal to `0` and

Parameter	Type	Required or Optional?	Default Value	Description
				less than or equal to `1`.
`frameExtractionRemoveDcOffset`	int32	Optional	`1`	Specifies whether the mean signal value should be subtracted from all frames. Specify `0` to represent false or `1` to represent true.
`frameExtractionWindowType`	string	Optional	**`RECTANGULAR`**	Specifies the type of window to apply to each frame upon extraction. Valid values are **`HAMMING`**, **`HANNING`**, **`RECTANGULAR`**, and **`BLACKMAN`**.
`frameExtractionRoundToPowerOfTwo`	int32	Optional	`1`	Specifies whether the window size should be rounded to the next power of two. Specify `0` to represent false or `1` to represent true.
`frameExtractionBlackmanCoeff`	double	Optional	`0.42`	Specifies the constant coefficient used for the generalized Blackman window. This value is ignored for other window types.
`frameExtractionSnipEdges`	int32	Optional	`1`	Specifies whether end effects should be handled by writing only frames that completely fit the data. Specify `0` to represent false or `1` to represent true.
`melBanksNBins`	int32	Optional	`23`	Specifies the number of triangular Mel Frequency bins. This value must be greater than or equal to `1`

Parameter	Type	Required or Optional?	Default Value	Description
`melBanksLowFreq`	double	Optional	`20`	Specifies the low cutoff frequency for the Mel Frequency bins.
`melBanksHighFreq`	double	Optional	`0`	Specifies the high cutoff frequency for the Mel Frequency bins. When negative, the value is added to the Nyquist frequency, which is half of the sampling rate of the audio.
`computeFbankFeatures`	double	Optional	`0`	Specifies whether to perform FBank feature computations. Specify `0` to represent false or `1` to represent true.
`fbankUseEnergy`	int32	Optional	`0`	Specifies whether an extra dimension that contains the computed energy should be appended to each FBank feature frame. Specify `0` to represent false or `1` to represent true.
`fbankEnergyFloor`	double	Optional	`0`	Specifies the linear floor on energy (absolute, not relative) for the FBank feature computations. This value must be greater than or equal to `0`.
`fbankRawEnergy`	int32	Optional	`1`	Specifies whether energy should be computed before preemphasis and windowing. Specify `0` to represent false or `1` to represent true.

Parameter	Type	Required or Optional?	Default Value	Description
`fbankUseLogFbank`	int32	Optional	`1`	Specifies whether the output should contain log-filterbank values. Otherwise, the output values are linear. Specify `0` to represent false or `1` to represent true.
`fbankUsePower`	int32	Optional	`1`	Specifies whether power should be used in the FBank feature computations. Otherwise, the magnitude is used. Specify `0` to represent false or `1` to represent true.
`computeMfccFeatures`	double	Optional	`0`	Specifies whether to perform mel frequency cepstral coefficients (MFCC) feature computations. Specify `0` to represent false or `1` to represent true.
`mfccNCeps`	int32	Optional	`13`	Specifies the number of cepstral coefficients in each frame, including C0. This value must be greater than or equal to `1` and less than or equal to `melBanksNBins`.
`mfccUseEnergy`	int32	Optional	`1`	Specifies whether energy (not C0) should be used in the MFCC feature computations. Specify `0` to represent false or `1` to represent true.
`mfccEnergyFloor`	double	Optional	`0`	Specifies the linear floor on energy (absolute, not relative)

Parameter	Type	Required or Optional?	Default Value	Description
				for the MFCC feature computations. This value must be greater than or equal to `0`.
`mfccRawEnergy`	int32	Optional	`1` (True)	Specifies that energy should be computed before preemphasis and windowing.
`plpCompressFactor`	double	Optional	`0.33333`	Specifies the compression factor used in the PLP feature values. This value must be greater than 0 and less than 1.
`mfccCepstralLifter`	double	Optional	`22`	Specifies the constant that controls the scaling of the MFCC feature values.
`computePlpFeatures`	double	Optional	`0`	Specifies whether to perform perceptual linear prediction (PLP) feature computations. Specify `0` to represent false or `1` to represent true.
`plpLpcOrder`	int32	Optional	`12`	Specifies the order of the linear predictive coding (LPC) analysis used in the PLP feature computations. This value must be greater than or equal to `1` and less than or equal to `25`.
`plpNCeps`	int32	Optional	`13`	Specifies the number of cepstral coefficients in each PLP feature frame, including C0. This value must be greater than or equal to `1` and less than or equal to `plpLpcOrder` + 1.

Parameter	Type	Required or Optional?	Default Value	Description
`plpUseEnergy`	int32	Optional	`1`	Specifies whether energy (not C0) should be used for the zeroth feature value in the PLP feature computations. Specify `0` to represent false or `1` to represent true.
`plpEnergyFloor`	double	Optional	`0`	Specifies the linear floor on energy (absolute, not relative) for the PLP feature computations. This value must be greater than or equal to `0`.
`plpRawEnergy`	int32	Optional	`1`	Specifies whether energy should be computed before preemphasis and windowing. Specify `0` to represent false or `1` to represent true.
`plpCompressFactor`	double	Optional	`0.33333`	Specifies the compression factor used in the PLP feature computations. This value must be greater than or equal to `0` and less than `1`.
`plpCepstralLifter`	int32	Optional	`22`	Specifies the constant that controls the scaling of the PLP feature values. This value must be greater than or equal to `0`.
`plpCepstralScale`	double	Optional	`1`	Specifies the cepstral scaling constant used in the PLP computation. This value must be greater than `0`.

Parameter	Type	Required or Optional?	Default Value	Description
`featureScalingMethod`	string	Optional	**`NONE`**	Specifies the feature scaling method to apply to the computed feature vectors. Valid values are **`NONE`** and **`STANDARDIZATION`**.
`nOutputFrames`	int32	Optional	computed	Specifies the exact number of frames to include in the output. Extra frames are dropped and missing frames are padded with zeros. When not explicitly specified, this value is set to the minimum number of frames required to contain all of the feature values for the incoming audio file. This value must be greater than or equal to `1` .
`nContextFrames`	int32	Optional	`0`	Specifies the number of context frames to append before and after the current audio frame. This value must be greater than or equal to `0`.

Table 121 *Input Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`audioIn`	variable	blob	Required	No default value	Specifies the name of the variable that contains the incoming audio data.

Table 122 *Output Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
numFramesOut	variable	int64	Required	No default value	Specifies the name of the variable that tracks the valid number of output frames.
computedFeatureValuesOut	variable	array(dbl)	Required	No default value	Specifies the name of the variable that contains the computed feature values.

Edges are defined at the end of the project. The Source window streams data into the Calculate window:

```
<edge source='w_data' target='w_feature' role='data'/>
```

The Calculate window (`w_feature`) streams its results to a Score window (`w_score`).

```
<edge source='w_feature' target='w_score' role='data'/>
```

This Score window streams its results into another Calculate window that applies the Transcription algorithm.

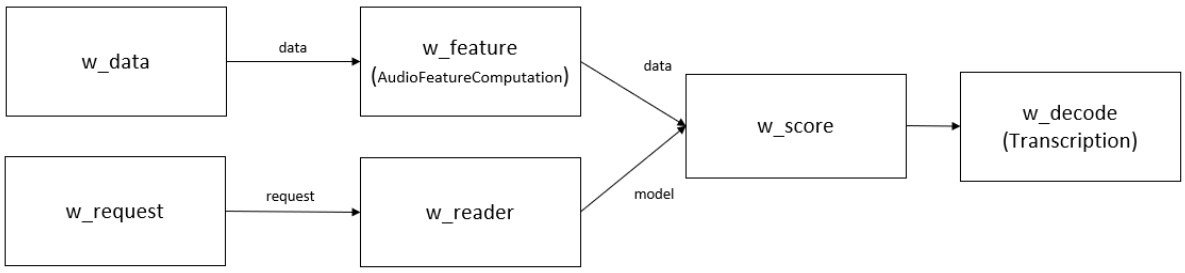
Converting Speech to Text

Conversion Overview

SAS Event Stream Processing provides two algorithms to enable conversion of speech to text: Audio Feature Computation and Transcription. You use these two algorithms together in order to convert audio input into a transcript.

Consider the following example:

Figure 4 *Audio to Speech*



- A Source window (`w_data`) streams binary audio data into a Calculate window that processes it with the Audio Feature Computation algorithm. For more information, see “Streaming Audio Feature Computation”.
- Another Source window (`w_request`) sends a request event into a Model Reader window to inject an analytic store file that contains a pre-trained acoustic model.
- The Audio Feature Computation results and the model in the analytic store file are streamed into a Score window (`w_score`).
- Results from the Score window are streamed to a Calculate window (`w_decode`) that applies the Transcription algorithm in order to produce the transcript. For more information, see “Streaming Speech Transcription”.

Here is the Source window that sends the request event and the Model Reader window that receives it:

```
<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>

   </connectors>
</window-source>

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

Here is the edge between those two windows:

```
<edge source='w_request' target='w_reader'  role='request'/>
```

The Model Reader window uses a model event to stream the analytic store file into a Score window (`w_score`) that then streams its results into a Calculate window running the Transcription algorithm.

```
<edge source='w_score'   target='w_decode'  role='data'/>
```

For more information about how to train an RNN acoustic model with the Language Model action set, see *SAS Visual Data Mining and Machine Learning: Programming Guide*.

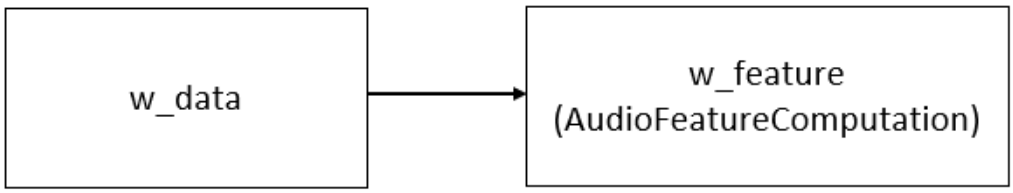
Streaming Audio Feature Computation

Audio feature computation is the process of applying an algorithm to an audio signal in order to convert it into a sequence of acoustic feature vectors. These vectors contain a serviceable numerical representation of the audio signal. You can perform further analytics using these acoustic feature vectors as input.

Note: The audio feature computation algorithm is supported on Linux (x86 architecture) and Microsoft Windows systems.

Consider the following example:

Figure 5 *Applying the Audio Feature Computation Algorithm*



This continuous query includes the following:

- a Source window that provides audio data to be analyzed
- a Calculate window that processes the audio data and converts it into a sequence of feature vectors

The Source window `w_data` creates streaming data events based on input data from a CSV file.

- The first field (`_path_`) denotes where the file originated. It might be useful to propagate this value along the entire processing chain. For example, in a speech-to-text application, it might be useful to maintain the connection between the final transcripts and the source audio files.
- The second field (`audio`) references a binary large object (blob) that comprises the audio in the current streaming data event.

```
<window-source name='w_data' insert-only='true' index='pi_EMPTY'>
   <schema>
      <fields>
         <field name='_path_' type='string' key='true'/>
         <field name='audio' type='blob'/>
      </fields>
   </schema>
   <connectors>
   ...
   </connectors>
</window-source>
```

The Calculate window `w_feature` receives those data events and processes the audio data with the audio feature computation algorithm. The parameters chosen enable you to work with Mel Frequency Cepstral Coefficient (MFCC) features that consist of 40 coefficients for each 25 millisecond long frame.

```
<window-calculate name="w_feature" output-insert-only='true' produces-only-
inserts='true'
        index='pi_EMPTY' algorithm="AudioFeatureComputation">
       <schema>
          <fields>
          <field name='_path_' type='string' key='true'/>
          <field name='_num_frames_' type='int64'/>
          <field name='f' type='array(dbl)'/>
       </fields>
   </schema>
   <parameters>
      <properties>
         <property name="frameExtractionFrameShift">10</property>
         <property name="frameExtractionFrameLength">25</property>
         <property name="frameExtractionDither">0</property>
         <property name="melBanksNBins">40</property>
         <property name="mfccNCeps">40</property>
```

```
            <property name="featureScalingMethod">STANDARDIZATION</property>
            <property name="nOutputFrames">3500</property>
         </properties>
      </parameters>
      <input-map>
         <properties>
            <property name="audioIn">audio</property>
         </properties>
      </input-map>
      <output-map>
         <properties>
            <property name="numFramesOut">_num_frames_</property>
            <property name="computedFeatureValuesOut">f</property>
         </properties>
      </output-map>
      <connectors>
      ...
      </connectors>
   </window-calculate>
```

The following properties govern the streaming audio feature computation algorithm:

Table 123 Parameters

Parameter	Type	Required or Optional?	Default Value	Description
`frameExtractionframeShift`	double	Optional	`10`	Specifies the time difference (in milliseconds) between the beginning of consecutive frames. This value must be greater than `0`.
`frameExtractionframeLength`	double	Optional	`25`	Specifies the length of a frame (in milliseconds). This value must be greater than `0`.
`frameExtractionDither`	double	Optional	`1.0`	Specifies the dithering constant. A value of `0.0` means no dithering. This value must be greater than or equal to `0`.
`frameExtractionPreemphCoeff`	double	Optional	`0.97`	Specifies the coefficient used in performing signal preemphasis. This value must be greater than or equal to `0` and

Parameter	Type	Required or Optional?	Default Value	Description
				less than or equal to `1`.
`frameExtractionRemoveDcOffset`	int32	Optional	`1`	Specifies whether the mean signal value should be subtracted from all frames. Specify `0` to represent false or `1` to represent true.
`frameExtractionWindowType`	string	Optional	**`RECTANGULAR`**	Specifies the type of window to apply to each frame upon extraction. Valid values are **`HAMMING`**, **`HANNING`**, **`RECTANGULAR`**, and **`BLACKMAN`**.
`frameExtractionRoundToPowerOfTwo`	int32	Optional	`1`	Specifies whether the window size should be rounded to the next power of two. Specify `0` to represent false or `1` to represent true.
`frameExtractionBlackmanCoeff`	double	Optional	`0.42`	Specifies the constant coefficient used for the generalized Blackman window. This value is ignored for other window types.
`frameExtractionSnipEdges`	int32	Optional	`1`	Specifies whether end effects should be handled by writing only frames that completely fit the data. Specify `0` to represent false or `1` to represent true.
`melBanksNBins`	int32	Optional	`23`	Specifies the number of triangular Mel Frequency bins. This value must be greater than or equal to `1`

Parameter	Type	Required or Optional?	Default Value	Description
melBanksLowFreq	double	Optional	20	Specifies the low cutoff frequency for the Mel Frequency bins.
melBanksHighFreq	double	Optional	0	Specifies the high cutoff frequency for the Mel Frequency bins. When negative, the value is added to the Nyquist frequency, which is half of the sampling rate of the audio.
computeFbankFeatures	double	Optional	0	Specifies whether to perform FBank feature computations. Specify 0 to represent false or 1 to represent true.
fbankUseEnergy	int32	Optional	0	Specifies whether an extra dimension that contains the computed energy should be appended to each FBank feature frame. Specify 0 to represent false or 1 to represent true.
fbankEnergyFloor	double	Optional	0	Specifies the linear floor on energy (absolute, not relative) for the FBank feature computations. This value must be greater than or equal to 0.
fbankRawEnergy	int32	Optional	1	Specifies whether energy should be computed before preemphasis and windowing. Specify 0 to represent false or 1 to represent true.

Parameter	Type	Required or Optional?	Default Value	Description
`fbankUseLogFbank`	int32	Optional	`1`	Specifies whether the output should contain log-filterbank values. Otherwise, the output values are linear. Specify `0` to represent false or `1` to represent true.
`fbankUsePower`	int32	Optional	`1`	Specifies whether power should be used in the FBank feature computations. Otherwise, the magnitude is used. Specify `0` to represent false or `1` to represent true.
`computeMfccFeatures`	double	Optional	`0`	Specifies whether to perform mel frequency cepstral coefficients (MFCC) feature computations. Specify `0` to represent false or `1` to represent true.
`mfccNCeps`	int32	Optional	`13`	Specifies the number of cepstral coefficients in each frame, including C0. This value must be greater than or equal to `1` and less than or equal to `melBanksNBins`.
`mfccUseEnergy`	int32	Optional	`1`	Specifies whether energy (not C0) should be used in the MFCC feature computations. Specify `0` to represent false or `1` to represent true.
`mfccEnergyFloor`	double	Optional	`0`	Specifies the linear floor on energy (absolute, not relative)

Parameter	Type	Required or Optional?	Default Value	Description
				for the MFCC feature computations. This value must be greater than or equal to `0`.
`mfccRawEnergy`	int32	Optional	`1` (True)	Specifies that energy should be computed before preemphasis and windowing.
`plpCompressFactor`	double	Optional	`0.33333`	Specifies the compression factor used in the PLP feature values. This value must be greater than 0 and less than 1.
`mfccCepstralLifter`	double	Optional	`22`	Specifies the constant that controls the scaling of the MFCC feature values.
`computePlpFeatures`	double	Optional	`0`	Specifies whether to perform perceptual linear prediction (PLP) feature computations. Specify `0` to represent false or `1` to represent true.
`plpLpcOrder`	int32	Optional	`12`	Specifies the order of the linear predictive coding (LPC) analysis used in the PLP feature computations. This value must be greater than or equal to `1` and less than or equal to `25`.
`plpNCeps`	int32	Optional	`13`	Specifies the number of cepstral coefficients in each PLP feature frame, including C0. This value must be greater than or equal to `1` and less than or equal to `plpLpcOrder` + 1.

Parameter	Type	Required or Optional?	Default Value	Description
`plpUseEnergy`	int32	Optional	`1`	Specifies whether energy (not C0) should be used for the zeroth feature value in the PLP feature computations. Specify `0` to represent false or `1` to represent true.
`plpEnergyFloor`	double	Optional	`0`	Specifies the linear floor on energy (absolute, not relative) for the PLP feature computations. This value must be greater than or equal to `0`.
`plpRawEnergy`	int32	Optional	`1`	Specifies whether energy should be computed before preemphasis and windowing. Specify `0` to represent false or `1` to represent true.
`plpCompressFactor`	double	Optional	`0.33333`	Specifies the compression factor used in the PLP feature computations. This value must be greater than or equal to `0` and less than `1`.
`plpCepstralLifter`	int32	Optional	`22`	Specifies the constant that controls the scaling of the PLP feature values. This value must be greater than or equal to `0`.
`plpCepstralScale`	double	Optional	`1`	Specifies the cepstral scaling constant used in the PLP computation. This value must be greater than `0`.

Parameter	Type	Required or Optional?	Default Value	Description
`featureScalingMethod`	string	Optional	**`NONE`**	Specifies the feature scaling method to apply to the computed feature vectors. Valid values are **`NONE`** and **`STANDARDIZATION`**.
`nOutputFrames`	int32	Optional	computed	Specifies the exact number of frames to include in the output. Extra frames are dropped and missing frames are padded with zeros. When not explicitly specified, this value is set to the minimum number of frames required to contain all of the feature values for the incoming audio file. This value must be greater than or equal to `1` .
`nContextFrames`	int32	Optional	`0`	Specifies the number of context frames to append before and after the current audio frame. This value must be greater than or equal to `0`.

Table 124 *Input Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`audioIn`	variable	blob	Required	No default value	Specifies the name of the variable that contains the incoming audio data.

Table 125 *Output Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
numFramesOut	variable	int64	Required	No default value	Specifies the name of the variable that tracks the valid number of output frames.
computedFeatureValuesOut	variable	array(dbl)	Required	No default value	Specifies the name of the variable that contains the computed feature values.

Edges are defined at the end of the project. The Source window streams data into the Calculate window:

```
<edge source='w_data' target='w_feature' role='data'/>
```

The Calculate window (`w_feature`) streams its results to a Score window (`w_score`).

```
<edge source='w_feature' target='w_score' role='data'/>
```

This Score window streams its results into another Calculate window that applies the Transcription algorithm.

Streaming Speech Transcription

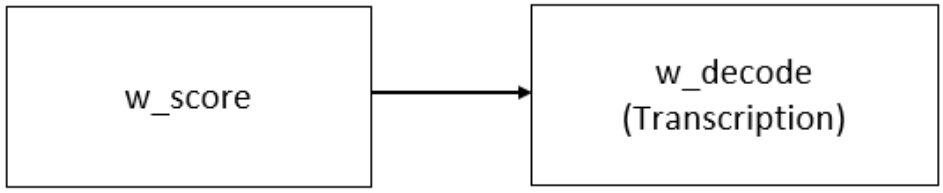
Streaming speech transcription is the process taking the output of an acoustic model and applying the natural structure and patterns of language in order to generate a final transcript. SAS Event Stream Processing provides the Transcription algorithm to implement streaming speech transcription. Two models are involved in data analysis:

- An *acoustic model* that associates acoustic feature vectors with the linguistic units that comprise the spoken utterance. This model is typically pre-trained with a large speech corpus. For all acoustic feature vectors, the model computes the probability that a given feature vector corresponds to each individual linguistic unit.
- A *language model* that is responsible for modeling word sequences in language. Basically, this model is a probability distribution. When given a specific sequence of words, it assigns a probability to the entire sequence. It then estimates the likelihood of a given phrase. When you build language models, you often assume that the probability of the occurrence of a particular word depends on the previous words in the sequence. The number of previous words that you use to determine this probability is essentially arbitrary. This leads to the idea of using an *n-gram model* to build language models. For example, a trigram (3–gram) model uses the previous two words to predict the occurrence of a particular word.

Before running the Transcription algorithm, you must supply the pre-trained acoustic model as an analytic store (ASTORE) file that is the result of a deep learning model. After an audio file has been processed by the Audio Feature Computation algorithm, it is represented as a set of vectors. Together, the analytic store file and these vectors are supplied to a score windows as shown in “Conversion Overview”.

Consider these windows:

Figure 6 *Applying the Transcription Algorithm*



- a Score window uses the acoustic model contained in an analytic store file and results from the Audio Feature Computation algorithm to generate an array of score data
- a Calculate window performs the speech transcription

The Score window `w_score` receives a model event from `w_reader`. In the input schema, the variable `_path_` specifies where to find the audio data file that has been transcribed. The variable `v` represents a vector that contains a set of probability distributions across time frames received from `w_feature`. This variable is required.

```
<window-score name='w_score'>
   <schema>
      <fields>
         <field name='_path_' type='string' key='true'/>
         <field name='v' type='array(dbl)'/>
      </fields>
   </schema>
   <models>
      <offline model-type='astore'>
         <input-map>
            <properties>
               <property name="inputDblArray">f</property>
            </properties>
         </input-map>
         <output-map>
            <properties>
               <property name="outputDblArray">v</property>
            </properties>
         </output-map>
      </offline>
   </models>
</window-score>
```

The Calculate window `w_decode` applies the Transcription algorithm to the data.

```
<window-calculate name='w_decode' algorithm='Transcription'>
   <schema>
      <fields>
         <field name='_path_' type='string' key='true'/>
         <field name='_audio_content_' type='string'/>
      </fields>
   </schema>
   <parameters>
      <properties>
         <property name="langModelPath">path/filename</property>
         <property name="columnMapPath">path/filename</property>
         <property name="blankLabel">&#x20;</property>
         <property name="spaceLabel">&amp;</property>
         <property name="nFrames">3500</property>
```

```
            <property name="alpha">2.5</property>
            <property name="beta">3.5</property>
         </properties>
      </parameters>
      <input-map>
         <properties>
            <property name="inputs">v</property>
         </properties>
      </input-map>
      <output-map>
         <properties>
            <property name="transOut">_audio_content_</property>
         </properties>
      </output-map>
      <connectors>
      ...
      </connectors>
   </window-calculate>
```

The following properties govern the speech transcription algorithm in the Calculate window:

Table 126 *Parameters*

Parameter	Type	Required or Optional?	Default Value	Description
`langModelPath`	string	Required	No default value	Specifies the path of the language model file, which is in CSV format. Each row in the file except the header represents an N-gram term. This term refers to a continuous sequence of n words and the information that it contains. The number in the first column of each row is the log probability of that row's N-gram term. The last column contains the value of the back-off weight. The words that each N-gram term contains are listed in order, starting from the second column, with one word per column.
`columnMapPath`	string	Required	No default value	Specifies the path of the column map file, which is in CSV format. Each row in the file except the header specifies the mapping between an index and a label.

Parameter	Type	Required or Optional?	Default Value	Description
				For example, suppose that the first row shows the label "A". In the `inputs` that stream into the Calculate window, the first score in each time frame indicates the predicted score for label "A". Spaces at the end of every label are considered padding except when the label is used as `blankLabel` or `spaceLabel`. For example, label "A␣" is interpreted as "A". Also, empty labels are not permitted.
`nFrames`	int64	Required	No default value	Specifies the maximum number of time frames in an audio that are used to extract acoustic features. Specify an integer value. The value of `nFrames` is determined by the input mapping parameter `inputs`. The value that you specify must be the same as the number of consecutive time frames that `inputs` provides the probability distribution over labels. If score data arriving at the Calculate window are generated by a Score window, then the model used by that Score window must be trained on the same features. It must also have the same number of time frames. When the features coming into the Score window are generated from a Calculate window that uses the AudioFeatureComputation algorithm, the value of

Parameter	Type	Required or Optional?	Default Value	Description
				`nFrames` must be the same value as the `nOutputFrames` parameter.
`blankLabel`	string	Optional	_	Specifies the string used to indicate the 'blank' label. The value should match one of the rows from the column map file that is specified by `columnMapPath`.
`spaceLabel`	string	Optional	_	Specifies the string used to indicate the ‘space’ label. The value should match one of the rows from the column map file that is specified by `columnMapPath`.
`alpha`	double	Optional	`1.0`	Specifies a tunable parameter that strengthens or weakens the influence of the language model on results. When alpha is large, the language model heavily influences transcription. Conversely, when alpha is small, the transcription depends more heavily on the acoustic model.
`beta`	double	Optional	`0.0`	Specifies a tunable parameter that strengthens or weakens the influence of the length of a sentence on results. When you choose a large value of `beta`, the transcription includes more words. A small value of beta leads to little dependency on sentence length, which leads to a transcription of fewer words.
`maxPathSize`	int64	Optional	`100`	Specifies the maximum number of paths kept as candidates of the final

Parameter	Type	Required or Optional?	Default Value	Description
				result during the decoding process. Specify a positive integer. When you choose a large value of `maxPathSize`, more candidates of transcripts are considered. This can lead to better results. When you choose a small value of `maxPathSize`, fewer candidates of word sequences are considered. This can lead to faster processing speeds.
`ngramsOrder`	int64	Optional	`3`	Specifies the highest order of N-grams to use during decoding process. Specify a positive integer no less than `1`.

Table 127 *Input Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`inputs`	varlist	array (double)	Required	No default value	Specifies name of the array that contains the incoming score data. This array contains scores returned by an acoustic model. The scores indicate the probability distribution over labels in every time frame. The Calculate window uses the probability distributions as well as the language model to generate reasonable transcription result. The scores are frame by frame in order. Scores from the same time frame strictly follow the same order, which is the order of label list specified in the column map file. For example, if the second value of the array `inputs` is 0.1 and the system does

Name	Type	Variable Type	Required or Optional?	Default Value	Description
					predict more than one label, then the probability of the second label in the first frame is estimated to be 0.1 by the acoustic model. If an audio has fewer time frames than `nFrames`, then the scores of those non-existing time frames are padded as zeros.

Table 128 *Output Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`transOut`	variable	string	Required	No default value	Specifies the variable name of the transcription result.

The edge between the Score window and Calculate window is defined at the end of the project.

```
<edges>
   <edge source='w_score' target='w_decode' role='data'/>
</edges>
```

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

Using Recommender Systems

Overview

A recommender system attempts to predict the rating or preference that someone would give an item such as a book, article, or product based on previous ratings or other items. The recommender system shipped with SAS Event Stream Processing provides two approaches:

- Regularized Matrix Factorization (RMF), which projects objects into a lower dimensional latent space in order to discover latent features that underlie the interactions between two different types of entities. In the training window, a Non-Negative Matrix Factorization (NMF) method can also be chosen to train the recommender. It produces the same model structure and parameters as RMF, but the factor parameters in NMF are constrained to be nonnegative.
- K Nearest Neighbor (KNN) classification, which takes an input measure in a feature space and assigns a class based on the K-nearest neighbors in that space.

Note: The current implementation of recommender scoring in SAS Event Stream Processing does not account for already rated items when it produces the top number of recommendations. That is,

items already rated by a user might appear in the recommendations generated for that user. Recommendations generated by the SAS Visual Analytics Recommender System Action Set ignore those items. Thus, results differ across these products.

You can apply offline or online recommender models to streaming events.

Offline Recommender Models

Generating Offline Recommender Models

To generate offline recommender models, use the Recommender System action set provided by SAS Visual Analytics. Specifically, generate the following SAS Cloud Analytic System (CAS) tables:

Table 129 *SAS Cloud Analytic System (CAS) Tables for Recommender Systems*

Recommender	Tables
RMF	■ item factor table (generated by `recommend.recomals` simultaneously with the user factor table) ■ user factor table (generated by `recommend.recomals` simultaneously with the item factor table) ■ item average rating table (generated by `recommend.recomrateinfo`) ■ user average rating table (generated by `recommend.recomrateinfo`)
KNN	■ item average rating table (generated by `recommend.recomrateinfo`) ■ user average rating table (generated by `recommend.recomrateinfo`) ■ user similarity table (generated by `recommend.recomsim`) ■ ratings by users table (this is the training data that you create to generate the user similarity table)

To use SAS Event Stream Processing to apply these models to streaming events, you must export each of those tables to CSV files.

Table 130 *Table to CSV Correspondence*

CAS Table	Required Format of CSV File	Model Parameter to Specify CSV File
item factor table	`item_id,_F0_,..._FN_`. Here, `_FN_` is the Nth latent factor variable. The CSV file can also contain a leading ID column.	`itemTable`
user factor table	`user_id,_F0_,..._FN_`. Here, `_FN_` is the Nth latent factor variable. The CSV file can also contain a leading ID column.	`userTable`
item average rating table	`item_id,_Stat_,_NRatings_`. The `_Stat_` is the average rating.	`itemRateInfo`

CAS Table	Required Format of CSV File	Model Parameter to Specify CSV File
user average rating table	`user_id,_Stat_,_NRatings_`. The `_Stat_` is the average rating.	`userRateInfo`
user similarity table	`USER_ID_1,USER_ID_2,_Sim_`. The `_Sim_` is the similarity rating.	`similarUsers`
ratings by user table	This CSV file must contain three fields that specify values for the following parameters: `userid`, `itemid`, `rating`. The fields must appear in that order.	`ratingsByUser`

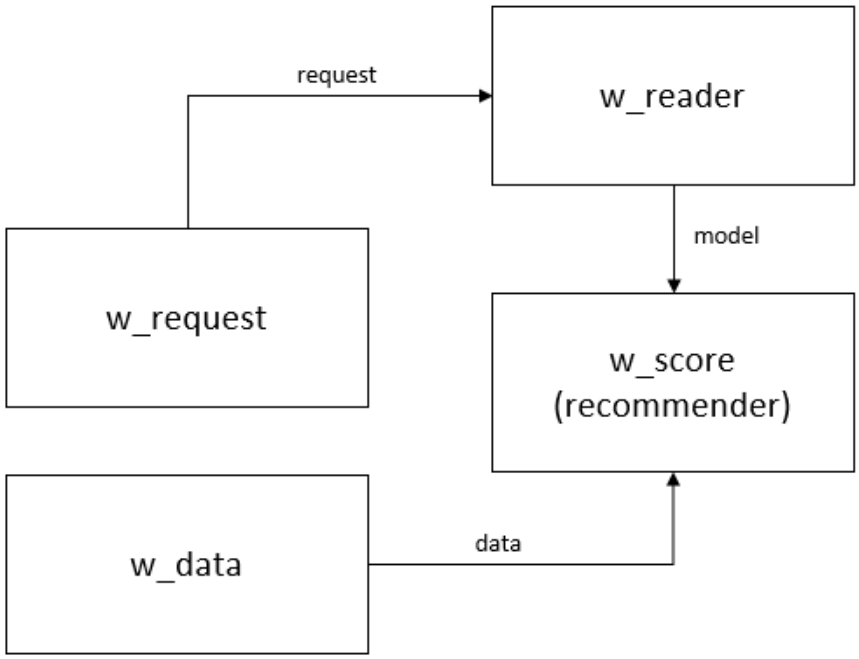
For more information about the Recommender System action set, see *SAS Visual Analytics: Programming Guide*.

You can update the parameter values of offline models dynamically or statically.

Dynamically Updating Parameter Values

The following continuous query dynamically updates the model parameter values of an offline model:

Figure 7 *Offline Recommender Model with Dynamically Updated Parameter Values*



- a Source window streams request events into a Model Reader window that specify parameter values for the offline model
- a Model Reader window streams model events that specify the model into a Score window
- a separate Source window streams data events that contain the data to be scored by the model into the Score window

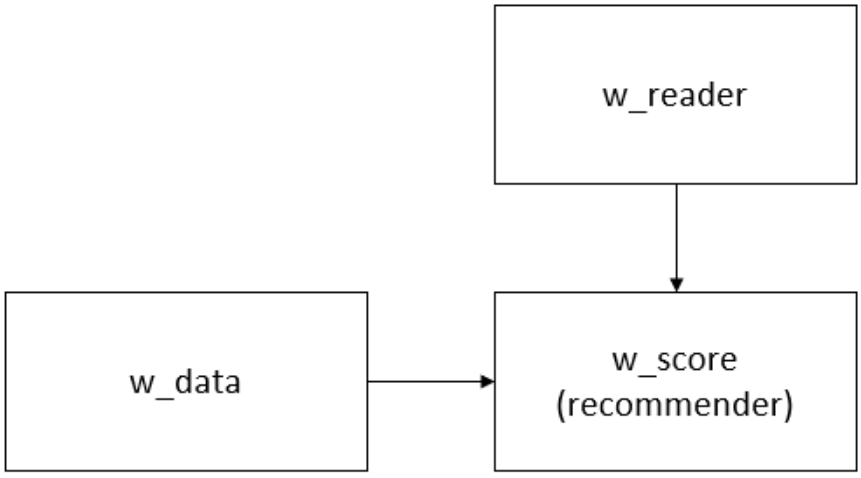
For example:

```
<window-source name="w_data" pubsub="true" insert-only="true" index="pi_EMPTY">
          <schema>
            <fields>
            ...
            </fields>
          </schema>
          <connectors>
            ...
          </connectors>
        </window-source>
        <window-source name="w_request" pubsub="true" insert-only="true"
index="pi_EMPTY">
          <schema>
            <fields>
             ...
            </fields>
          </schema>
          <connectors>
            ...
          </connectors>
        </window-source>
        <window-model-reader name="w_reader" pubsub="true" model-type="recommender" />
        <window-score name="w_score" pubsub="true">
          <schema>
            <fields>
            ...
            </fields>
          </schema>
          <models>
            <offline model-type="recommender">
              <input-map>
                <properties>
                  ...
                </properties>
              </input-map>
              <output-map>
                <properties>
                ...
                </properties>
              </output-map>
            </offline>
          </models>
          <connectors>
            ...
          </connectors>
        </window-score>
      </windows>
      <edges>
        <edge role="data" source="w_data" target="w_score" />
        <edge role="model" source="w_reader" target="w_score" />
        <edge role="request" source="w_request" target="w_read" />
      </edges>
```

Statically Updating Parameter Values

The following continuous query statically updates model parameter values:

Figure 8 *Offline Recommender Model with Statically Updated Parameter Values*



Notice that here, the Model Reader window does not receive request events from a Source window. Model parameter values are coded within the Model Reader window itself. For example:

```
<window-source name="w_data" pubsub="true" insert-only="true" index="pi_EMPTY">
          <schema>
            <fields>
            ...
            </fields>
          </schema>
          <connectors>
            ...
          </connectors>
        </window-source>
<window-model-reader name="w_reader" pubsub="true" model-type="recommender">
  <parameters>
    <properties>
      <property name="name">value</property>
      ...
    </properties>
  </parameters>
</window-model-reader>
<window-score name="w_score" pubsub="true">
   <schema>
    ...
   </schema>
   <models>
      <offline model-type="recommender">
        <input-map>
            <properties>
             ...
            </properties>
        </input-map>
        <output-map>
```

```
            <properties>
             ...
            </properties>
          </output-map>
        </offline>
      </models>
      <connectors>
      ...
      </connectors>
  </window-score>
  ...
  <edges>
     <edge role="data" source="w_data" target="w_score" />
     <edge role="model" source="w_read" target="w_score" />
  </edges>
```

Specifying Model Parameters

The following model parameters are common to the RMF (or NMF) and KNN recommender systems:

Table 131 *Common Model Parameters for RMF and KNN*

Parameter	Type	Required or Optional?	Default Value	Description
`method`	string	Optional	**`RMF`**	Specifies the algorithm to be used for recommendation. Valid values are **`RMF`** or **`KNN`**.
`userRateInfo`	string	Required	No default value	Specifies the name of the user average rating CSV file. These can be used as bias terms. See Table 130 for details.
`userRateInfoDelimeter`	string	Optional	**`"COMMA"`**	Specifies the delimiter used in the user average rating CSV file. Valid values are **`"COMMA"`**, **`"TAB"`**, or **`"SPACE"`**.
`userRateInfoLineBreak`	string	Optional	**`"LF"`**	Specifies the line break character used in the user average rating CSV file. Valid values are **`"LF"`**, **`"CF"`**, or **`"CRLF"`**.
`userRateInfoIndex`	string	Optional	**`"N"`**	Specifies whether the user average rating CSV file has row indices. Valid values are **`"Y"`** or **`"N"`**.
`itemRateInfo`	string	Required	No default value	Specifies the name of the item average rating CSV file. See Table 130 for details. This file is used with popularity-based cold start.

Parameter	Type	Required or Optional?	Default Value	Description
itemRateInfoDelimeter	string	Optional	"COMMA"	Specifies the delimiter used in the item average rating CSV file. Valid values are "COMMA", "TAB", or "SPACE".
itemRateInfoLineBreak	string	Optional	"LF"	Specifies the line break character used in the item average rating CSV file. Valid values are "LF", "CF", or "CRLF".
itemRateInfoIndex	string	Optional	"N"	Specifies whether the item average rating CSV file has row indices. Valid values are "Y" or "N".

The following model parameters are specific to an RMF or NMF recommender model:

Table 132 *Model Parameters for the RMF or NMF Algorithms*

Parameter	Type	Required or Optional?	Default Value	Description
itemTable	string	Required	No default value	Specifies the name of the item factor CSV file. It contains the low-rank features of each item.
itemTableDelimeter	string	Optional	"COMMA"	Specifies the delimiter used in the item factor CSV file. Valid values are "COMMA", "TAB", or "SPACE".
itemTableLineBreak	string	Optional	"LF"	Specifies the line break character used in the item factor CSV file. Valid values are "LF", "CF", or "CRLF".
itemTableIndex	string	Optional	"N"	Specifies whether the item factor CSV file has row indices. Valid values are "Y" or "N".
userTable	string	Required	No default value	Specifies the name of the user factor CSV file.
userTableDelimiter	string	Optional	"COMMA"	Specifies the delimiter used in the user factor CSV file. Valid values are "COMMA", "TAB", or "SPACE".
userTableLineBreak	string	Optional	"LF"	Specifies the line break character used in the user factor CSV file.

Parameter	Type	Required or Optional?	Default Value	Description
				Valid values are `"LF"`, `"CF"`, or `"CRLF"`.
`userTableIndex`	string	Optional	`"N"`	Specifies whether the user factor CSV file has row indices. Valid values are `"Y"` or `"N"`.
`ratingsByUser`	string	Optional	No default value	Specifies the name of the user ratings CSV file.
`ratingsByUserDelimiter`	string	Optional	`"COMMA"`	Specifies the delimiter used in the user ratings CSV file. Valid values are `"COMMA"`, `"TAB"`, or `"SPACE"`.
`ratingsByUserLineBreak`	string	Optional	`"LF"`	Specifies the line break character used in the user ratings CSV file. Valid values are `"LF"`, `"CF"`, or `"CRLF"`.
`ratingsByUserIndex`	string	Optional	`"N"`	Specifies whether the user ratings CSV file has row indices. Valid values are `"Y"` or `"N"`.

The following model parameters are to a KNN recommender model:

Table 133 *Model Parameters for the KNN Algorithm*

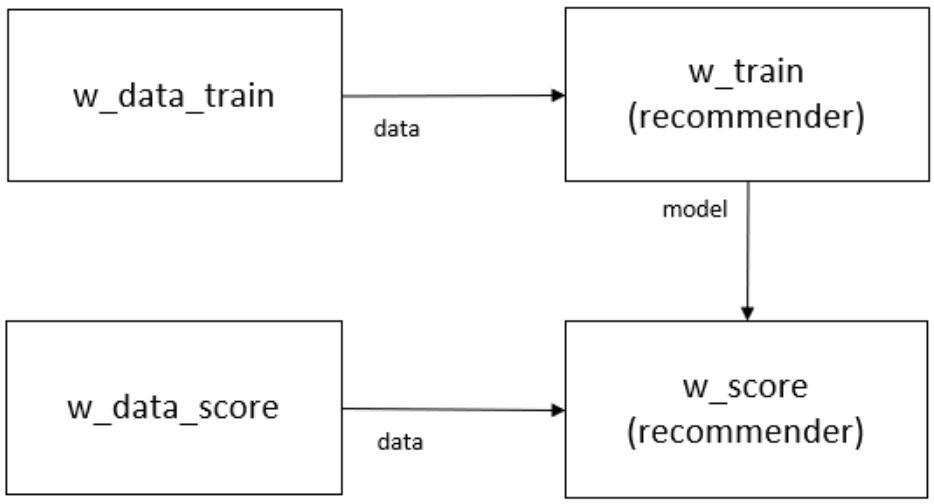
Parameter	Type	Required or Optional?	Default Value	Description
`k`	int64	Optional	`20`	Specifies the maximum number of nearest neighbors when calculating the rating of each user.
`similarUsers`	string	Required	No default value	Specifies the name of the user similarity CSV file. See Table 130 for details.
`similarUsersDelimeter`	string	Optional	`"COMMA"`	Delimiter used in the user similarity CSV file. Valid values are `"COMMA"`, `"TAB"`, or `"SPACE"`.
`similarUsersLineBreak`	string	Optional	`"LF"`	Specifies the line break character used in the user similarity CSV file. Valid values are `"LF"`, `"CF"`, or `"CRLF"`.

Parameter	Type	Required or Optional?	Default Value	Description
`similarUsersIndex`	string	Optional	`"N"`	Specifies whether the user similarity CSV file has row indices. Valid values are `"Y"` or `"N"`.
`ratingsByUser`	string	Required	No default value	Specifies the name of the user ratings CSV file. This CSV file must contain three fields that specify values for the following parameters: `userid`, `itemid`, `rating`. The fields must appear in that order.
`ratingsByUserDelimiter`	string	Optional	`"COMMA"`	Delimiter used in the user ratings file. Valid values are `"COMMA"`, `"TAB"`, or `"SPACE"`.
`ratingsByUserLineBreak`	string	Optional	`"LF"`	Specifies the line break character used in the user ratings file. Valid values are `"LF"`, `"CF"`, or `"CRLF"`.
`ratingsByUserIndex`	string	Optional	`"N"`	Specifies whether the user ratings CSV file has row indices. Valid values are `"Y"` or `"N"`.

Online Recommender Models

The following continuous query applies an online recommender model that is being trained online.

Figure 9 *Scoring Data with a Recommender Model Being Trained*



- a Source window streams training data to a Train window
- a Train window streams trained recommender models to downstream windows
- a second Source window streams data to be scored to the Score window

There are two ways to train a recommender model:

- with initial training data sets
- with streaming data

Here is an example of a Train window that uses an initial training data set:

```
<window-train name="w_train" pubsub="true" algorithm="recommender">
          <parameters>
            <properties>
              <property name="method">RMF</property>
              <property name="initMethod">BATCH_TRAINING</property>
              <property name="initModelFormat">CSV4</property>
              ...
              <property name="ratingsByUser">./input/ratings_by_user2.csv</property>
              <property name="ratingsByUserDelimiter">COMMA</property>
              <property name="ratingsByUserLineBreak">LF</property>
              <property name="ratingsByUserIndex">N</property>
              <property name="userTableFilename">./input/userFactorBook.csv></
property>
              <property name="itemTableFilename">./input/itemFactorBook.csv</property>
              <property name="userBiasFilename">./input/userBiasBook.csv</property>
              <property name="itemBiasFilename">./input/itemBiasBook.csv</property>
            </properties>
          </parameters>
          <input-map>
            <properties>
             ...
            </properties>
          </input-map>
        </window-train>
```

Training an Online Recommender Model

The following properties tune the recommender algorithm within the Train window.

Table 134 *Input Mapping*

Name	Type	Required or Optional?	Default Value	Description
`user`	string	Required	No default value	Specifies the incoming user ID.
`item`	string	Required	No default value	Specifies the incoming item ID.
`rating`	double	Required	No default value	Specifies the rating value.

You can reconfigure some of the parameter values in the Train window while data is streaming through the model. Create an edge between the Source window and the Train window with the role “request.” Then, stream a reconfig request and events that change parameter values. For example:

```
i,n,1,"action","reconfig"
i,n,2,"maxIters","20"
i,n,3,"commitInterval","50"
i,n,4,"trainInterval","50"
i,n,5,"updateItemFactor","2"
i,n,6,"scale","1"
i,n,7,,
```

Table 135 *Parameters for the Train Window*

Name	Type	Required or Optional?	Default Value	Description	Parameter Value Can Be Reconfigured
`method`	string	Optional	**`RMF`**	Specifies the method of recommendation. Valid values are **`RMF`** or **`NMF`**.	No
`maxUsers`	int64	Optional	`200000`	Specifies the maximum number of users. The limit is `32767`.	No
`maxItems`	int64	Optional	`200000`	Specifies the maximum number of items. The limit is `32767`.	No
`maxRatings`	int64	Optional	`100000`	Specifies the maximum number of stored ratings per user and per item. Any incoming rating beyond this capacity causes the removal of the oldest rating.	No
`nFactors`	int32	Optional	`3`	Specifies the number of latent features per user and per item, including the user bias and item bias. There must be at least two, in which case the user and item bias terms are the only features that are used in the model.	No

Name	Type	Required or Optional?	Default Value	Description	Parameter Value Can Be Reconfigured
regL2	double	Optional	1e–2	Specifies a value that improves matrix conditions in the training algorithm.	Yes
maxInitIters	int32	Optional	20	Specifies the number of iterations in initial training.	No
maxIters	int32	Optional	3	Specifies the number of iterations in each online training episode triggered by streaming training events.	Yes
initMethod	string	Optional	BATCH_TRAINING	Specifies the method used to initialize the model Valid values are "BATCH_TRAINING", "MODEL_READING", or "ONLINE_TRAINING".	No
initSeed	int32	Optional	–1	Specifies the desired seed of the pseudo-random number generator. –1 sets the seed to the current time.	Yes
initModelFormat		Optional	CSV4	When initMethod="MODEL_READING", specifies the format of the pre-trained model used to initialize the Train window. Valid values are "CSV4" or "CSV2".	No
nInit	int64	Optional	500	Specifies the number of training events required to initiate online training (that is, start the first episode of model update on streaming data).	No
commitInterval	int64	Optional	100	Specifies the number of training events	Yes

Name	Type	Required or Optional?	Default Value	Description	Parameter Value Can Be Reconfigured
				required between two consecutive model commitments to score window.	
`trainInterval`	int64	Optional	`100`	Specifies the number of training events between two consecutive online training episodes.	Yes
`updateItemFactor`	int32	Optional	`1`	Specifies whether to update item factors: A value of 0 indicates not to update item factors. A nonzero value indicates to update items factors.	Yes
`hasUserBias`	int32	Optional	`1`	Specifies whether the model contains user bias terms: A value of 0 indicates no user bias terms. A nonzero value indicates user bias terms.	Yes
`hasItemBias`	int32	Optional	`1`	Specifies whether the model contains item bias terms: A value of 0 indicates no item bias terms. A nonzero value indicates item bias terms	Yes
`hasCommonBias`	int32	Optional	`1`	Specifies whether the model contains common bias terms: A value of zero indicates no common bias terms. A nonzero value indicates common bias terms.	Yes
`logToConsole`	int32	Optional	`1`	Specifies whether training information is displayed in the console. A value of 0 indicates that the information is not displayed. A nonzero	Yes

Name	Type	Required or Optional?	Default Value	Description	Parameter Value Can Be Reconfigured
				value indicates that the information is displayed.	
scale	int32	Optional	0	Specifies how to rescale ratings to [0,10]. A value of 0 indicates no scaling. A nonzero value indicates scaling.	Yes
userTableName	string	Optional	"" (empty string)	Specifies the CSV file to which the user factor matrix is saved.	No
itemTableName	string	Optional	"" (empty string)	Specifies the CSV file to which the item factor matrix is saved.	No
userBiasName	string	Optional	"" (empty string)	Specifies the CSV file to which the user bias terms are saved.	No
itemBiasName	string	Optional	"" (empty string)	Specifies the CSV file to which the item bias terms are saved.	No

When initMethod="BATCH_TRAINING", the RMF recommender model is initialized with a single CSV file that contains an initial training set. Use the value of ratingsByUser to specify the name of the training set. Specify values for the remaining parameters within the Train window.

Table 136 *Training Parameters When initMethod="BATCH_TRAINING"*

Name	Type	Required or Optional?	Default Value	Description
ratingsByUser	string	Optional	"" (empty string)	Name of the CSV file that contains the initial training data.
ratingsByUserDelimeter	string	Optional	"COMMA"	Delimiter used in the initial training data file. Valid values are "COMMA", "TAB", or "SPACE".
ratingsByUserLineBreak	string	Optional	"LF"	Line break character used in the initial training data file. Valid values are "LF", "CR", or "CRLF".

Name	Type	Required or Optional?	Default Value	Description
ratingsByUserIndex	string	Optional	"N"	Specifies whether the ratings model file has row indices.

The RMF recommender model is imported from a specified source when initMethod="MODEL_READING". The model is defined by four CSV files when initModelFormat="CSV4" or two CSV files when initModelFormat="CSV2".

Model parameters when initModelFormat="CSV4" are as follows:

Table 137 *Training Parameters When initMethod="MODEL_READING" and "initModelFormat="CSV4"*

Parameter	Type	Required or Optional?	Default Value	Description
itemTable	string	Required	"" (empty string)	Name of the CSV file that contains the item model without biases.
itemTableIndex	string	Optional	"Y"	Specifies whether the item model file contains row indexes.
userTable	string	Required	"" (empty string)	Specifies the name of the CSV file that contains the user model without biases.
userTableIndex	string	Optional	"Y"	Specifies whether the user model file contains row indexes.
userRateInfo	string	Required	"" (empty string)	Specifies the name of the CSV file that contains the user biases.
userRateInfoIndex	string	Optional	"N"	Specifies whether the user biases file contains row indexes.
itemRateInfo	string	Required	"" (empty string)	Name of the CSV file that contains the item biases.
itemRateInfoIndex	string	Optional	"N"	Specifies whether the item biases file contains row indexes.
csvDelimeter	string	Optional	"COMMA"	Delimiter used in the CSV files. Valid values are "COMMA", "TAB", or "SPACE".
csvLineBreak	string	Optional	"LF"	Line break character used in CSV files. Valid values are "LF", "CR", or "CRLF".

Model parameters when `initModelFormat="CSV2"` are as follows:

Table 138 *Training Parameters When initMethod="MODEL_READING" and "initModelFormat="CSV2"*

Parameter	Type	Required or Optional?	Default Value	Description
`itemTableWithBias`	string	Required	`""` (empty string)	Name of the CSV file that contains the item model with biases.
`itemTableWithBiasIndex`	string	Optional	`"N"`	Specifies whether the item model file has row indexes.
`userTableWithBias`	string	Optional	`""` (empty string)	Name of the CSV file that contains the user model with biases.
`userTableWithBiasIndex`	string	Optional	`"N"`	Specifies whether the user model file has row indexes.
`csvDelimiter`	string	Optional	`"COMMA"`	Delimiter used in the CSV files. Valid values are `"COMMA"`, `"TAB"`, or `"SPACE"`.
`csvLineBreak`	string	Optional	`"LF"`	Line break character used in the CSV files. Valid values are `"LF"`, `"CR"`, or `"CRLF"`.

The model is initialized with streaming data when `initMethod="ONLINE_TRAINING"`.

Here is a Train window that uses online training. Notice that the only reference to external CSV files is the value of `initModelFormat`. You can use an offline model to initialize a model that you trained online.

```
<window-train name='w_train' algorithm='recommender'>
            <parameters>
              <properties>
                <property name="nFactors">3</property>
                <property name="method">RMF</property>
                <property name="initMethod">ONLINE_TRAINING</property>
                <property name="initModelFormat">CSV4</property>
                <property name="maxUsers">200000</property>
                <property name="maxItems">200000</property>
                <property name="maxInitIters">20</property>
                <property name="maxIters">10</property>
                <property name="commitInterval">1000</property>
                <property name="trainInterval">500</property>
                <property name="nInit">1000</property>
                <property name="updateItemFactor">1</property>
                <property name="hasCommonBias">0</property>
                <property name="hasUserBias">1</property>
                <property name="hasItemBias">1</property>
                <property name="scale">0</property>
```

```
      <property name="maxRatings">100000</property>
      <property name="regL2">1e-2</property>
    </properties>
  </parameters>
  <input-map>
    <properties>
      ...
    </properties>
  </input-map>
</window-train>
```

Scoring Data with Recommender Models

Score windows use the following properties for both KNN and RMF recommender models:

Table 139 *Parameters*

Name	Type	Required or Optional?	Default Value	Description
`nTopRecoms`	int32	Optional	`50`	The number of top recommendations to make to each user.
`filterRatedItems`	int32	Optional	`0`	The option for filtering items in the recommended list. A value of `0` indicates no filtering. A positive value indicates removing previously rated items. A negative value indicates removing previously unrated items.

Table 140 *Input Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`user`	variable	string	Required	No default value	Specifies the targeted user ID to make recommendations **Note:** Specify this as a key field.

Table 141 *Output Mapping*

Name	Type	Variable Type	Required or Optional?	Default Value	Description
`itemOut`	variable	string	Required	No default value	Specifies the IDs of each recommended item **Note:** Specify this as a key field.
`ratingOut`	variable	double	Required	No default value	Specifies the predicted rating for each recommended item
`rankOut`	variable	double	Required	No default value	Specifies the relative ranking values for each recommended item. The highest ranking is specified as 1, the second-highest as 2, and so on.

Online Scoring and Training Using Offline Models

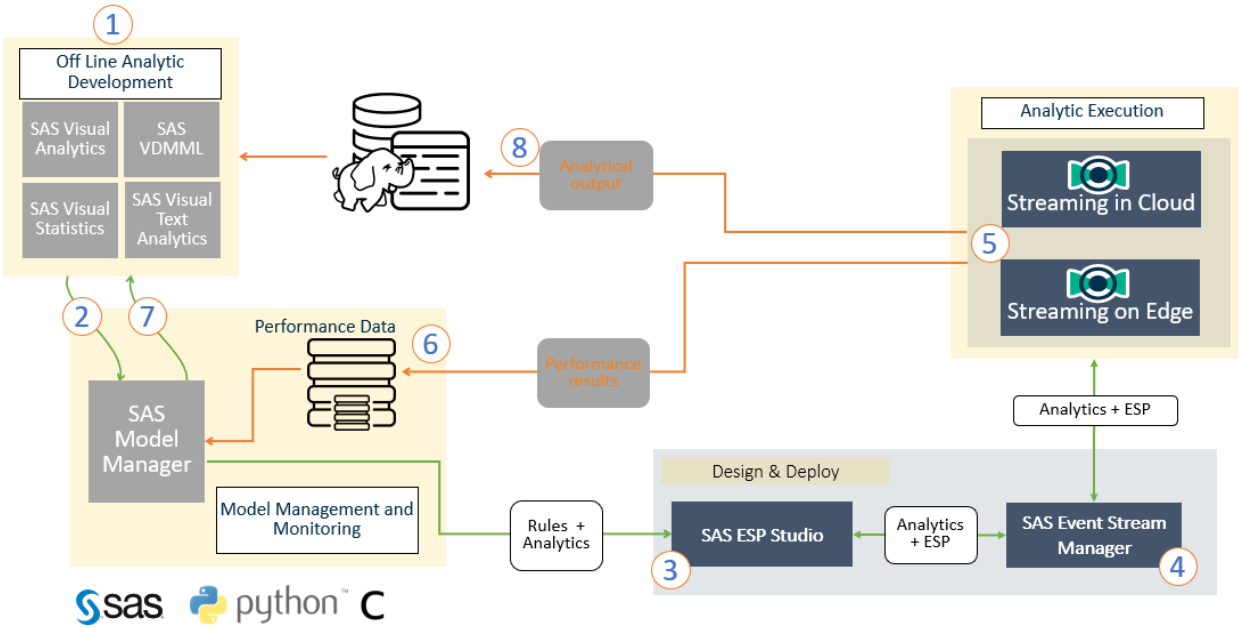
Overview

Offline models are specified, developed, trained, and stored separately from the ESP server. SAS Event Stream Processing supports a variety of offline models from sources such as the following:

- SAS Visual Data Mining and Machine Learning
- SAS Visual Text Analytics
- JMP Software
- Open-source frameworks such as TensorFlow

The following figure depicts how you develop and manage offline models for use with SAS Event Stream Processing.

Figure 10 *Offline Analytical Model Development and Management*



1 Use SAS Visual Data Mining and Machine Learning and SAS Visual Text Analytics to analyze structured and unstructured data. Use SAS Visual Analytics and SAS Visual Statistics to examine and understand patterns, trends, and relationships in data. Based on the resulting analysis, develop an offline analytical model.

2 Register offline models with SAS Model Manager to assess and compare candidate models for champion model selection.

3 Import champion models into SAS Event Stream Processing Studio to include in event stream processing projects. Run those projects with SAS Event Stream Processing to apply the analytics to event streams.

4 Use SAS Event Stream Manager to deploy SAS Event Stream Processing projects to edge servers and cloud environments.

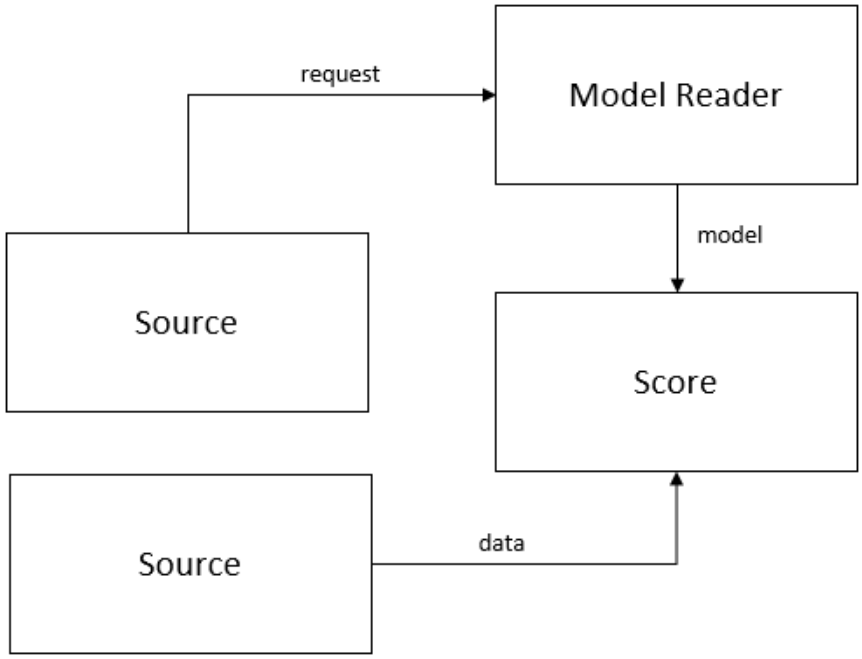
5 Execute deployed projects on edge servers and in cloud environments.

6 Use performance data from those deployed projects to monitor champion models' performance.

7 Use SAS Visual Data Mining and Machine Learning and SAS Visual Text Analytics to tune deployed models for optimal performance.

8 Store analysis results offline.

To apply offline models to streaming data with SAS Event Stream Processing, use a Source window to stream them into a Model Reader window through a request event. Then, use the Model Reader window to stream the offline model into a Score window through a model event. You can use a separate Source window to stream data to be scored by the model into the Score window through a data event.



Most offline models developed with SAS software are stored in analytic store (ASTORE) files. An *ASTORE file* is a binary file that contains the model's state after it completes the training phase of data analysis. Other offline models, such as Recommender Scoring, are stored in combinations of non-binary files.

Score windows contain input maps that specify the properties of the data to be scored. When you use an offline model in an ASTORE file, there is a strict correspondence between these properties and the input schema of the Source window that streams data into the Score window.

For example, suppose that the following is a property in the input map of the Score window:

```
<property name='LOAN'>LOAN</property>
```

Here, `name='LOAN'` refers to the input variable LOAN that is coded in the ASTORE file. The specified value of the property, "LOAN," corresponds to a field name in the input schema of the Source window that streams the data to be scored.

Thus, if you specified `'LOAN_ABC'` as a field name in the input schema of the Source window, then you would need to change the value of the property in the input map of the Score window:

```
<property name='LOAN'>LOAN_ABC</property>
```

Note: When you persist an offline model and then restore it, you must republish the model to the Model Reader window through a request event. This enables the Score window to score new events. For more information, see "Implementing Persist and Restore Operations" in *SAS Event Stream Processing: Using Source and Derived Windows*.

Loading Models Stored in Analytic Store Files

Before you load models that are stored in analytic store files into a Model Reader window:

1. Create the analytic store file with the appropriate SAS procedure or action set. When you use a SAS procedure, use the SAVESTATE statement to create a binary representation of the model.
2. Use the DOWNLOAD statement of the ASTORE procedure to retrieve that representation and produce a local analytic store file that contains it.
3. Move the analytic store file to a location where your SAS Event Stream Processing project can find it.
4. Code a Source window to find the analytic store file.
5. Code a Model Reader window to get the analytic store file from that Source window.

You can use models stored in analytic store files that apply the following algorithms:

Table 142 *Supported Algorithms*

Algorithm	Description	General Uses	SAS Procedure or action set to create the analytic store file
Association Rule Mining	Performs mining on data to discover potential patterns and associations between variables. The action generates rules from the data that identify frequent item sets. It is useful in scenarios that involve thousands of distinct items that are grouped into a hierarchy.	Use association rule mining for marketing, inventory management, and analyzing consumer behavior.	Association Rule Mining action set For more information, see *SAS Visual Data Mining and Machine Learning: Deep Learning Programming Guide*.
Batch Reinforcement Learning	Trains on a history of agent interactions with an environment. Batch reinforcement methods rely on a sequential record of (s, a, r, s') tuples, where s is the state, a is the action, r is the reward, s' is the next state.	Use batch reinforcement for optimization problems such as medical treatment, customer journeys, emergency response movements, and educational practices.	Reinforcement Learning Action set For more information, see *SAS Visual Data Mining and Machine Learning: Reinforcement Learning Programming Guide*.
Bayesian Network	Scores data using a Bayesian network model. A Bayesian network is a directed acyclic graphical	Use Bayesian networks for documentation classification, biomonitoring, and gene	BNET procedure For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*. For an introduction to the topic, see "Working with Bayesian Networks" in *SAS*

Algorithm	Description	General Uses	SAS Procedure or action set to create the analytic store file
	model in which nodes represent random variables and the links between nodes represent conditional dependency of the random variables.	regulatory networks.	*Visual Analytics: Working with SAS Visual Data Mining and Machine Learning* .
Butterworth Filter Chebyshev Type I or Type II Filter	Provides a digital filter that can operate on a time series signal to selectively extract specific frequency spectrum of the signal. In digital signal processing, filtering is used primarily to remove unwanted parts of the signal, such as noise, and to extract useful parts of the signal.	Use these filters for digital signal processing (EKG, vibration data, and so on).	timeFilters action set `filterDesign` action `filterType="lowpass | highpass | bandpass | bandstop", filterName="butterworth | cheby1 | cheby2", ... filterOutTable="`*`astorefile`*`"` Move that *astorefile* to a location where your project can find it. For more information, see *SAS Visual Forecasting: Forecasting Procedures* **Note:** The `timeId` input must be equally spaced.
Dirichlet Gaussian Mixture Model	Performs a cluster analysis with a Gaussian mixture model (GMM). The GMM is a probabilistic model that assumes all the data points are generated from a mixture of Gaussian distributions. The Dirichlet process is used to find the number of clusters in the data.	Use Dirichlet Gaussian mixture model for brain imaging or topic modeling.	GMM procedure For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*.
Factorization Machine	Provides a general predictor similar to support vector machines. It can estimate reliable parameters under very high sparsity.	When data sets are very large and are sparse, you can apply a factorization machine to the data set to extract the most important or	FACTMAC procedure For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*. For an introduction to the topic, see "Working with Factorization Machines" in *SAS Visual Analytics: Working with SAS Visual Data Mining and Machine Learning* .

Algorithm	Description	General Uses	SAS Procedure or action set to create the analytic store file
		latent or hidden features.	

General Linear Model	Fits linear regression models using the method of least squares.	Use the general linear model to quantify the relationship between several independent or predictor variables and a dependent or criterion variable. This model applies to a wide range of domains.	Regression action set
			`glm` action
			Generate a binary representation of the model with the STORE argument:
			`...`
			`store="trained_model"`
			Then generate a binary file with the DOWNLOAD argument of the ASTORE procedure:
			`...`
			`download rstore="trained_model" store=binary_file`
			Move that *binary_file* to a location where your project can find it.
			For more information, see *SAS Visual Statistics: Programming Guide*.

Generalized Additive Models	Fits generalized additive models by penalized likelihood.	Use generalized additive models in time series studies of the health effects of air pollution, or on satellite-derived data to predict fishery resources.	Generalized Additive Model action set
			`gampl` action
			Generate a binary representation of the model with the STORE argument:
			`...`
			`store="binary_file"`
			Then generate a binary file with the DOWNLOAD argument of the ASTORE procedure:
			`...`
			`download rstore="trained_model" store=binary_file`
			Move that *binary_file* to a location where your project can find it.
			For more information, see *SAS Visual Statistics: Programming Guide*.

Generalized Linear Multi-task Learning	Implements the multi-task learning technique for least squares loss with ℓ_1 and graph structure penalizations. The technique solves multiple related sparse linear	Use the generalized linear multi-task learning model to predict the following: ■ survival of patients in different hospitals	MTLEARN procedure For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*.

Algorithm	Description	General Uses	SAS Procedure or action set to create the analytic store file
	regression problems simultaneously. A graph structure encodes the relationships between the problems. The analytic store file scores data using a graph-regularized multi-task regression model.	■ student test results across a set of schools ■ air pollution indexes across a set of observation stations	

Generalized Linear Regression Model	Fits generalized linear regression models and logistic regression models using iterative optimization methods.	Use the generalized linear regression model to derive an equation for the effects of independent variables on dependent variables.	Regression action set
			`genmod` action
			Generate a binary representation of the model with the STORE argument
			`...`
			`store="trained_model"`
			Then generate a binary file with the DOWNLOAD argument of the ASTORE procedure:
			`...`
			`download rstore="trained_model"`
			`store=binary_file`
			Move that *binary_file* to a location where your project can find it.
			For more information, see *SAS Visual Statistics: Programming Guide*.

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.	Use gradient boosting tree for anomaly detection in supervised learning settings when data is unbalanced, such as with DNA sequences or credit card transactions.	GRADBOOST procedure For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*. For an introduction to the topic, see "Working with Gradient Boosting Models" in *SAS Visual Analytics: Working with SAS Visual Data Mining and Machine Learning*.
Hidden Markov Model	Scores data using hidden Markov models. A hidden	Use hidden Markov models in the areas of	HMM procedure This ASTORE is capable of producing multiple rows in the output map. To enable

Algorithm	Description	General Uses	SAS Procedure or action set to create the analytic store file

	Markov model is a bivariate discrete time process, where the first variate (the state), is a (hidden) Markov chain. The second variate, the observable process, is a sequence of independent random variables such that the conditional distribution of the second variate depends only on the first variate. Gaussian mixture models, regression models, mixture regression models, autoregressive error models, and vector autoregressive models with exogenous variables can be treated as special cases of hidden Markov models.	economics (for example, macroeconomics , marketing science), finance (for example, risk management, asset pricing, quantitative trading), science (for example, gene finding, protein secondary structure recognition, multiple sequence alignment), and engineering (for example, robotics, pattern recognition of speech and handwriting, machine status monitoring).	this feature, add the following fields to the output map:
			`<window-score name='w_score'>`
			`<schema>`
			`<fields>`
			`...`
			`<field name='row_id' type='int64' key='true/>`
			`</fields>`
			`</schema>`
			`<models>`
			`<offline model-type='astore'>`
			`<output-map>`
			`<properties>`
			`<property-name='_row_id_'>row_id</property>`
			`</properties>`
			`</output-map>`
			`</offline>`
			For more information, see *SAS Econometrics: Econometrics Procedures*.

Kernel Principal Component Analysis (KPCA)	Calculates the projection of data onto the principal components in high dimensional reproducing kernel Hilbert space (RKHS) using techniques of kernel methods. Fast and exact scoring options are available depending on how training step is performed.	Use KPCA to do nonlinear dimensionality reduction and to identify underlying nonlinear patterns in the data.	kernalPca action set KPCA procedure Move that *astorefile* to a location where your project can find it. For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*.
Random Forest	Builds decision trees at training	Use random forest for early	FOREST procedure

Algorithm	Description	General Uses	SAS Procedure or action set to create the analytic store file
	time. It writes the class of trees that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees.	disease detection or data mining to predict credit card default.	For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*. For an introduction to the topic, see "Working with Forests" in *SAS Visual Analytics: Working with SAS Visual Data Mining and Machine Learning* .

Robust Principal Components Analysis (RPCA)	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.
	Use the analytic store file to project new observations into the principal components space or into the low rank space.
		Use RPCA for robust data dimensionality reduction. For example, you can predict the energy output of a solar farm. You can also use RPCA for anomaly detection in video data by extracting the foreground (moving objects) from the background.	RPCA procedure
			To specify the type of projection, you must set the value of RPCA_PROJECTION_TYPE in a load request to the Model Reader window. Set RPCA_PROJECTION_TYPE = 0 to indicate projection into the principal components space. Use RPCA_PROJECTION_TYPE = 1 to indicate projection into the low rank space.A value of 2 produces the same result as a value of 1. However, the sparse part of the scoring data is stored in the scoring result table.
			For example:
			`i,n,1,"action","load"`
			`i,n,2,"type","astore"`
			`i,n,3,"RPCA_PROJECTION_TYPE","0"`
			`i,n,4,"reference","astorefile"`
			`i,n,5,,`
			Other parameters can be used to customize how the analysis is conducted. For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*.

Stability Monitoring Scoring	Scores events in order to monitor the stability of the underlying model. Detects anomalous behavior of various signals within event data and, using the stored model, generates forecasts of a target signal.	Use stability monitoring scoring for anomaly detection or error prediction based on a statistical model of the stable system. For example, you can use stability monitoring to capture gradual degradation of a turbofan engine long before a	stabilityMonitoring action set
			`smCalib` action
			`...`
			`scoreOutTable="astorefile"`
			Move that *astorefile* to a location where your project can find it.
			You must supply values for ProjectID and ModelID to the Model Reader window when you load the analytic store file.
			For example:
			`i,n,1,"action","load"`
			`i,n,2,"type","astore"`
			`i,n,3,"reference","astorefile"`
			`i,n,4,"ProjectId","1"`
			`i,n,5,"ModelId","1"`
			`i,n,6,,`

Algorithm	Description	General Uses	SAS Procedure or action set to create the analytic store file
		catastrophic failure occurs.	For more information, see *SAS Visual Forecasting: Forecasting Procedures* **Note:** The `timeId` input has the following conditions: ■ It must be equally spaced ■ The input used for scoring must have the same granularity as the one used for training

Support Vector Data Description	A one-class classification technique that can be useful in applications where data about one class is abundant but data about any other class is scarce or missing. You can model one-class data and subsequently use the model anomaly detection.	Use support vector data description (SVDD) to detect anomalies. It can be used for fault monitoring when you have data from reliable equipment such as aircraft turbines and high-end batteries. In this case, most of the data describes the healthy state of the machine.	SVDD action set
			`svddTrain` action
			When you train a model with multiple bandwidths, specify the appropriate parameters in the load request to the Model Reader window. Specify `modelId` to identify the model to score and `bwCol` to include bandwidth columns in the result. For example:
			`i,n,1,"action","load"`
			`i,n,2,"type","astore"`
			`i,n,3,"reference","astorefile"`
			`i,n,4,"modelId",1`
			`i,n,5,"bwCol",1`
			`i,n,6,,`
			For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*.

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.	Use support vector machine for face detection, text and hypertext categorization, handwriting recognition, and classification of images.	SVMACHINE procedure For more information, see *SAS Visual Data Mining and Machine Learning: Procedures*.

SAS Event Stream Processing also supports models that perform multi-task learning. These models are stored in analytic store files built with the MTLEARN procedure or Deep Learning action set. With multi-task learning, you train different models with the same data. For example, you might first train an image detection algorithm to differentiate between humans and street signs. Then, with the same data, you might train another model to parse the street signs (stop, yield, rail crossing, and so on). Finally, you might train a third model to predict how a person is going to react to the signs, again with the same data. Multi-task learning models can use both regression functions and classification functions during training.

Note: SAS Event Stream Processing does not support models in analytic store files created with single quotation marks in the attribute **name** field.

Using Deep Learning Models

You can score streaming data developed in models built with deep neural networks (DNNs). Generally, DNNs are fully connected neural networks that are used for classification or regression tasks. DNNs typically use stochastic gradient descent to train the model.

In addition to the typical DNN, the following variants are supported by SAS Event Stream Processing:

Table 143 *Supported Deep Neural Networks*

Deep Neural Network	Description
Convolutional Neural Networks (CNNs)	A class of DNNs that is widely used for image recognition, classification, and analysis and natural language processing. CNNs use variations of multilayer perceptrons in order to minimize preprocessing. This analytic store supports embedding with margin softmax loss. For more information, see Output Layer in *SAS Visual Data Mining and Machine Learning: Deep Learning Programming Guide*. Here are the CNN models that you can apply to streaming data: ■ Image classification: VGG–16 and deeper variants, ResNet–50 and deeper variants, SeNet, DenseNet, Inception V3, MobileNet, and ShuffleNet ■ Object detection: YOLO, SSD, R-CNN, Faster R-CNN, DETNet ■ Semantic segmentation: UNet ■ Instance segmentation: Mask R-CNN **Note:** SAS Event Stream Processing does not support offline CNN models that are trained using decompressed images. When you load images to a CNN model definition using the `loadImages` action of the images action set, set the DECODE parameter to `FALSE`. Set this parameter before you train and export a model as an analytic store file. **Note:** CNN models accept only the BGR order for images.
Recurrent Neural Networks (RNNs)	A class of DNNs that are specifically designed to handle sequence data, such as speech, text, time series, and so on. RNNs are called recurrent because they perform the same task for every element of a sequence. The output for each element depends on the computations of its preceding elements. The original RNN is simple in architecture, but it can be very hard to train when sequences get long. Two popular variants of RNN are widely used: Long Short-Term Memory (LSTM) and Gate Recurrent Unit (GRU). The deepLearn actions support all three model types (RNN, LSTM, and GRU). SAS Event Stream Processing supports models that perform classification and regression tasks on numeric or text data. You can use fully connected (FC) layers in your RNN model. You can feed more than one of the previous layers to an FC layer in the network. You can

Deep Neural Network	Description
	also mix in CNN layers. For more information, see What's New in the *SAS Visual Data Mining and Machine Learning: Deep Learning Programming Guide*.

Note: When an RNN model uses text input, it requires linguistic binary files. You must set the following environment variable in order to access those binary files:

```
export TKTXTANIO_BINDAT_DIR=/opt/sas/viya/home/SASFoundation/misc/tktg
```

Use the deepLearn action set to generate an analytic store file for your SAS Event Stream Processing project. Use the `buildModel` action and specify `type="DNN" | "CNN" |"RNN"` to create the model. Use the `dlTrain` action to train the model, and create the analytic store file using the `dlExportModel` action.

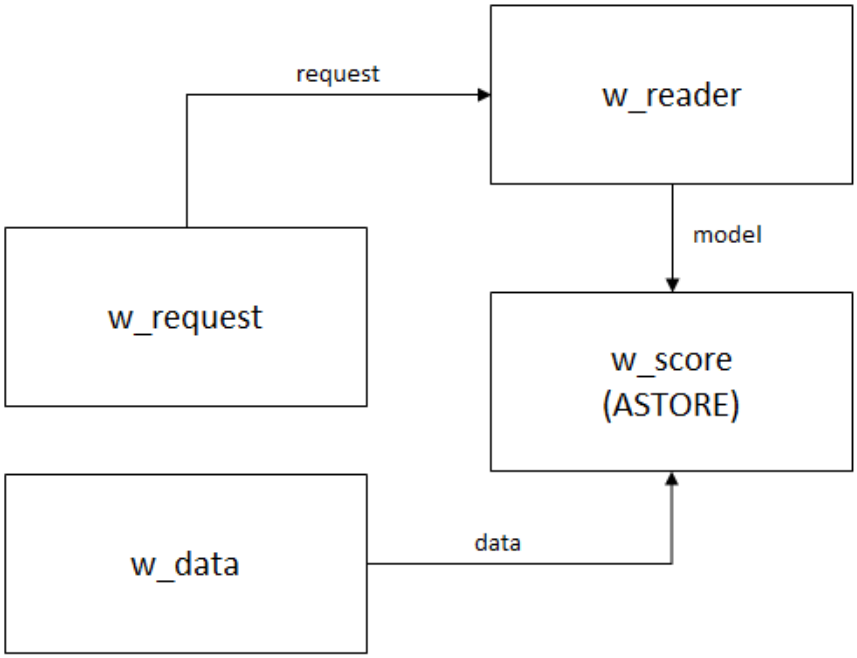
For more information about using the deepLearn action set to build deep learning models, see the *SAS Visual Data Mining and Machine Learning: Deep Learning Programming Guide*. For information about using SAS Visual Analytics to create neural networks, see "Working with Neural Networks" in *SAS Visual Analytics: Working with SAS Visual Data Mining and Machine Learning* .

Note: It is recommended that you score deep learning network analytic store files with graphics processing units (GPUs) that run on computer systems running Linux. For information about GPU support, see the *SAS Event Stream Processing on Linux: Deployment Guide*.

You can use the SAS Deep Learning Python (DLPy) package to build deep learning models with image, text, and audio data. DLPy is a high-level Python library that provides Keras APIs. You can use DLPy with an ONNX image classification model of image data, enabling the use of any GPU hardware. For an example, see Create DLPy Image Classification Model and Export to ONNX.

Example: Using a Random Forest Model in an Analytic Store File

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' insert-only='true' index='pi_EMPTY'>
          <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' model-type='astore'/>
```

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 analytic store file. The type of offline model is specified as astore, and the name of the analytic store file is referenced (forest_iris_astore.

Note: Make sure that the analytic store 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 model-type='astore'>
            <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**.

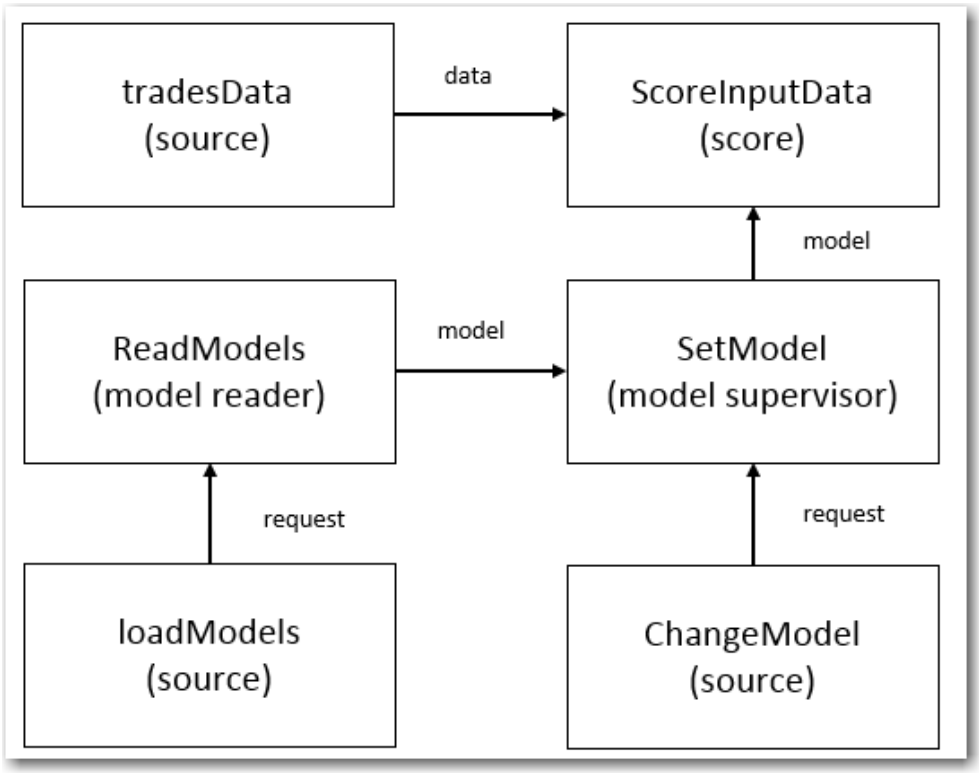
The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<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 models. 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 analytic store file into the engine.

2 The **`type`** of model to load is analytic store.

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

4 This event contains an empty `req_key`

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

6 The **`type`** of model to load is analytic store.

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

8 This event contains an empty `req_key`.

The models in the two analytic store 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,,
```