Getting Started with the CASL Programming Language 3.1

Requirements

To use SAS Cloud Analytic Services, the client machine must meet the following requirement:

- Use 64-bit Linux

There are additional requirements that are common with other programming languages. For other requirements, see “Programming Basics” in SAS Cloud Analytic Services: System Programming Guide.

Connect and Start a Session

How To

1. If you have not already done so, download the data sets that are used in examples. You can download the data at http://support.sas.com/documentation/onlinedoc/viya/examples.htm. Put them in a directory that is accessible to SAS.

2. Open SAS Studio from the URL in the form of http://hostname:port. Sign in using your user ID and password for your operating system account.
   a. Use the Server Files and Folders section to navigate to the directory that has the two data sets.
   b. Right-click on the directory and select Create ➔ Library. Specify movies as the name.

3. Use the Libraries section to confirm that the new library is listed.

4. Start a CAS session.
Enter the following line in the code editor and click to run the code.

```sas
cas casauto userid=sasdemo host="cloud.example.com" port=5570;
proc cas;
  session casauto;
  session.sessionstatus result=s;
  put s;
run;
```

Click the Code tab. The follow note indicates that you have an active CAS session.

```sas
57 cas casauto userid=sasdemo host="cloud.example.com" port=5570;
NOTE: The session CASAUTO connected successfully to Cloud Analytic Services cloud.example.com using port 5570.
The UUID is 0ed1cc35-c3ec-df49-be6e-68b9193eb8b8. The user is sasdemo and the active caslib is CASUSERHDFS(sasdemo).
NOTE: The SAS option SESSREF was updated with the value CASAUTO.
NOTE: The SAS macro _SESSREF_ was updated with the value CASAUTO.
NOTE: The session is using 139 workers.
58  proc cas;
59  session TestSess;
60  session.sessionstatus result=s;
61  put s;
62  run;
NOTE: Active Session now TestSess.
{state=Connected,number of Connections=1,Timeout=60,ActionStatus=Action is active,Authenticated=Yes,locale=en_US}
```

### About Your Connection and Server

After you connect, a session is started for you. As a documentation convention, a variable that is named S is used to represent the session.

- **S**
  - The name of the variable that is in CASL. It represents the session that is started for you in SAS Cloud Analytic Services.

- **session**
  - The software process that is started on the same hosts as SAS Cloud Analytic Services. When you reference your session through the S variable, statistical computations and actions are run on the server.

As soon as you connect, a good practice is to print information about the connection and session:

```sas
print s;
```

Your results will show different values. In the event that you have a network interruption between CAS and the server, the UUID for the session can be used to reconnect to a session.

To learn the most basic information about the server, you can run the serverStatus action that is part of the builtins action set:

```sas
proc cas;
  builtins.serverStatus;
```
Example: Train Gradient Boosted Trees with k-fold Cross Validation

About the Example

The purpose of this example is to describe how to train gradient boosted trees with k-fold cross validation. A k-fold cross validation process finds the average estimated validation error (misclassification error for nominal targets or average square error for interval targets) for the trained model. During cross validation, all data are divided into k subsets (folds). For each fold, a new model is trained, and then validated using the selected fold. The validation error estimates are then averaged over each set of training and scoring executions to obtain a single value.

Load the Data

Create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server.

```sas
libname mycas mysess; /* #1 */
data mycas.heart; /* #2 */
  set sashelp.heart;
run;
```

1. Define your own session and CAS engine librefs that connect to the CAS server. The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the `CASHOST` and `CASPORT` of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

2. The DATA step creates the data table `mycas.heart`, which consists of 5209 observations that have 17 variables. This DATA step assumes that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

Explore Data Using CAS Actions

This section describes how you can access and manage your data using the tables action set. For more information on the table action set see SAS Cloud Analytic Services: System Programming Guide.

```sas
proc cas;
  tableinfo /table='heart'; /* #1 */
```
The table action set executes the tableInfo action, which shows information about a table.

The table action set executes the columnInfo action, which shows column information.

The simple action set executes the summary action, which generates descriptive statistics of numeric variables such as the sample mean, sample variance, sample size, sum of squares, and so on. For more information about the simple action set, see “Summary” in SAS Cloud Analytic Services: Analytics Programming Guide

Results: Table Information for Heart

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NLS Encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Duplicated Rows</th>
<th>View</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEART</td>
<td>5209</td>
<td>17</td>
<td>utf-8</td>
<td>09Sep2016 15:18:00</td>
<td>09Sep2016 15:18:00</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Results: Column Information for HEART

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Id</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td></td>
<td>1</td>
<td>char</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DeathCause</td>
<td>Cause of Death</td>
<td>2</td>
<td>char</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>AgeCHDdiag</td>
<td>Age CHD Diagnosed</td>
<td>3</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Sex</td>
<td></td>
<td>4</td>
<td>char</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>AgeAtStart</td>
<td>Age at Start</td>
<td>5</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Height</td>
<td></td>
<td>6</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Weight</td>
<td></td>
<td>7</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Diastolic</td>
<td></td>
<td>8</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Systolic</td>
<td></td>
<td>9</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>MRW</td>
<td>Metropolitan Relative Weight</td>
<td>10</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Smoking</td>
<td></td>
<td>11</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>AgeAtDeath</td>
<td>Age at Death</td>
<td>12</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Cholesterol</td>
<td></td>
<td>13</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Chol_Status</td>
<td>Cholesterol Status</td>
<td>14</td>
<td>char</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>BP_Status</td>
<td>Blood Pressure Status</td>
<td>15</td>
<td>char</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Weight_Status</td>
<td>Weight Status</td>
<td>16</td>
<td>char</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Smoking_Status</td>
<td>Smoking Status</td>
<td>17</td>
<td>char</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>
Generate Folds for Cross Validation

Use CASL to partition the table and specify the number of partition folds for cross validation. For this example, we are generating ten folds. A column named \_fold\_ is produced, and the only way to get repeatable folds is to copy the new table and add the \_fold\_ column.

```cas
partition / table={name='heart',
compvars={'\_fold\_'},
comppgm='call streaminit(__rankid*1000);\_fold\_=floor(rand("UNIFORM")*10);'}
outtable={name='new_heart_with_fold', replace=True};
run;
```

**Log Output: Partitioned Table Log Output**

```plaintext
{caslib=CASUSERHDFS(casuser),tableName=NEW_HEART_WITH_FOLD,rowsTransferred=27,shuffleWaitTime=0.0000758171,minShuffleWaitTime=0,maxShuffleWaitTime=1.9073486E-6,averageShuffleWaitTime=4.5672957E-7}
```

Verify the \_Fold\_ Column

In order to verify the new \_fold\_ column, perform these simple statistics.

```cas
summary/ table='new_heart_with_fold' inputs={'\_fold\_'};
run;
simple.distinct / table={name='new_heart_with_fold', inputs='\_fold\_'};
run;
freq/ table={name='new_heart_with_fold', inputs={'\_fold\_'}};
run;
columninfo result=r /table={name='new_heart_with_fold'};
run;
```

**Results: Simple Distinct Results**

```
Distinct Counts for NEW_HEART_WITH_FOLD

<table>
<thead>
<tr>
<th>Column</th>
<th>Number of Distinct Values</th>
<th>Number of Missing Values</th>
<th>Truncated</th>
</tr>
</thead>
<tbody>
<tr>
<td>_fold_</td>
<td>10</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>
```
Remove the _Fold_ Column

Remove the _fold_ column since it is not our analysis variable.

\[
n\text{Vars} = \text{dim}(r[\text{‘columninfo’}]) - 1;
\]

Create an Input Variable List

Create a variable list where you define each variable. Each variable is defined by an expression, character, or numeric value. For this example, we use CASL to create our variable list. The syntax for the variable list follows the Assignment statement syntax. For more information see “ASSIGNMENT Statement” in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

\[
i = 4;
\]
\[
j = 2;
\]
\[
xx = \{r[\text{‘columninfo’}][1,1]; /* #1 */
\]
\[
do\text{ while } (i < n\text{Vars}); /* #2 */
\]
\[
xx = xx + r[\text{‘columninfo’}][i,1];
\]
\[
i = i + 1;
\]
\[
j = j + 1;
\]
\[
end;
\]
\[
print xx;
\]
\[
run;
\]

1 The target xx uses the result variable from the previously run columninfo, and \( r \) as its expression in addition to searching for the value in 1, 1.

2 The DO WHILE statement executes statements in a DO loop repetitively as long as the condition is true. For more information see “Using a DO WHILE Statement” in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

Log Output: Variable List Concatenated

\[
\{\text{Status, Sex, AgeAtStart, Height, Weight, Diastolic, Systolic, MRW, Smoking, AgeAtDeath, Cholesterol, Chol_Status, BP_Status, Weight_Status, Smoking_Status}\}
\]
Train a Decision Tree Using a Left-Out Fold

Train the decision tree by splitting the sub-sampled data, then splitting each resulting segment, and so on recursively until some constraint is met.

```plaintext
function OneFoldTree(nFold, iFold); /*1*/
   /* generate _fold_where */
   foldwhere1 = "_fold_ NE " || (String)iFold;
   foldwhere2 = "_fold_ EQ " || (String)iFold;
   mymodel = "gbt_" || (String)iFold; /*2*/
   decisiontree.gbtreetrain /*3*/
      table={name="new_heart_with_fold", where=foldwhere1}
         inputs=xx
         target="AgeAtDeath"
         casout={name=mymodel, replace=1}
         maxbranch=2
         maxlevel=8
         leafsize=60
         ntree=100
         binorder=1
         nbins=100
         seed=1234
         learningRate=0.1
         subsamplerate=0.7
         m=64;
   decisionTree.gbtreescore result = r/ table={name='new_heart_with_fold',
         where=foldwhere2} model={name=mymodel}; /*4*/
   print r;
   myscoredata = "gbtscore_" || (String)iFold; /*5*/
   saveresult r replace dataset=myscoredata; /*6*/
   /* return prediction error; MSE or misclassification rate */
   return (r["ScoreInfo"]{3,2}); /*7*/
end func;
```

1 The FUNCTION statement creates a new function that can be called in an expression. In this example, the function is named `OneFoldTree` and has two arguments named `nFold` and `iFold`. For more information about the FUNCTION statement syntax, see "FUNCTION Statement" in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

2 Generate a model name.

3 Create a model without iFold, and score on the holdout iFold using the train model. The action set decision tree executes the action gbtreetrain that trains a gradient boosting tree. This function can be easily expanded to train other models such as a neural network with cross validation. For more information about gbtreetrain action syntax, see "Train gradient boosting tree" in SAS Cloud Analytic Services: Analytics Programming Guide.

4 Score iFold-th data with a trained model. The action set decision tree executes the action gbtreescore that scores a table using a gradient boosting tree model. For more information about gbtreescore action syntax, see "Score a table using gradient boosting tree" in SAS Cloud Analytic Services: Analytics Programming Guide.

5 Generate a score data set name. In this example the score data set name is `myscoredata`. The data set name is evaluated as "gbtscore_" || (String)iFold. For more information about the Assignment statement syntax, see "ASSIGNMENT Statement" in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.
The SAVERESULT statement creates a SAS data set from the results of an ACTION. For more information about the SAVERESULT statement syntax, see “SAVERESULT Statement” in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

The RETURN statement returns a value from the current function. For more information about the RETURN statement syntax, see “RETURN Statement” in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

Partial Results: Decision Tree Action

The SAS System

Results from decisionTree.gbt.treeTrain

<table>
<thead>
<tr>
<th>Gradient Boosting Tree for NEW_HEART_WITH_FOLD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trees</td>
<td>100.000000</td>
</tr>
<tr>
<td>Distribution</td>
<td>1.000000</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.100000</td>
</tr>
<tr>
<td>Subsampling Rate</td>
<td>0.700000</td>
</tr>
<tr>
<td>Number of Selected Variables (M)</td>
<td>14.000000</td>
</tr>
<tr>
<td>Number of Bins</td>
<td>100.000000</td>
</tr>
<tr>
<td>Number of Variables</td>
<td>14.000000</td>
</tr>
<tr>
<td>Max Number of Tree Nodes</td>
<td>33.000000</td>
</tr>
<tr>
<td>Min Number of Tree Nodes</td>
<td>17.000000</td>
</tr>
<tr>
<td>Max Number of Branches</td>
<td>2.000000</td>
</tr>
<tr>
<td>Min Number of Branches</td>
<td>2.000000</td>
</tr>
<tr>
<td>Max Number of Levels</td>
<td>8.000000</td>
</tr>
<tr>
<td>Min Number of Levels</td>
<td>0.000000</td>
</tr>
<tr>
<td>Max Number of Leaves</td>
<td>17.000000</td>
</tr>
<tr>
<td>Min Number of Leaves</td>
<td>9.000000</td>
</tr>
<tr>
<td>Maximum Size of Leaves</td>
<td>466.000000</td>
</tr>
<tr>
<td>Minimum Size of Leaves</td>
<td>60.000000</td>
</tr>
<tr>
<td>Random Number Seed</td>
<td>1234.000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output CAS Tables</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS Library</td>
<td>gbt_0</td>
</tr>
<tr>
<td>CASUSERHDFS</td>
<td>2730</td>
</tr>
</tbody>
</table>

Run Each Model In Its Own Session

You can run each model in its own session by creating a function for each model.

```plaintext
function KFoldCV(nFold); /* #1 */
do i=1 to nFold; /* #2 */
   myerror[i] = OneFoldTree(nFold, i);
end;
return (myerror); /* #3 */
end func;
```

The FUNCTION statement creates a new function that can be called in an expression. In this example, the function is named KFoldCV and has one argument named nFold.
The DO statement, Iterative iterates over the list that starts at the value of 1 to the value of nFold. For more information about the DO Iterative syntax, see “DO Statement, Iterative Statement” in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

The RETURN statement returns a value from the current function. For more information about the RETURN statement syntax, see “RETURN Statement” in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

Train k (nFold) Models

During cross validation, all data are divided into k subsets (folds). For each fold, a new model is trained then validated using a selected fold. In this example, we have ten folds that we are going to train against the selected fold (nFold=1).

```sas
nFold = 10;
ModelError = KFoldCV(nFold);
run;
```

Log Output: Trained Models

```
NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
58.744875114

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
50.51471611

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
71.30714715

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
59.846180435

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
63.013744439

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
59.198893492

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
60.686242454

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
59.041830014

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
62.895729306

NOTE: The data set work.myscoredata has 3 observations and 2 variables.
NOTE: The data set work.myscoredata1 has 100 observations and 6 variables.
57.160065856

/*print error into log*/
print ModelError;
run;

Output Log: Model Error for Each Fold

{58.744875114, 50.51471611, 71.307144715, 59.846180435, 63.013744439, 59.198893492, 60.686242454, 59.041830014, 62.895729306, 57.160065856}

Compute Average Error Rate from Cross Validation

This section describes how to compute the average error rate or the misclassification error or average square of k-fold cross validation.

1. Compute the average error rate from cross validation. The mean squared error (MSE) of an estimator measures the average of the squares of the deviations.

2. The DO statement, Iterative iterates over the list that starts at the value of 1 to the value of nFold. For more information about the DO statement, Iterative syntax see "DO Statement, Iterative Statement" in SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference.

Log Output: Conversion of String to Number

NOTE: String '58.744875114' convert to number.
NOTE: String '50.51471611' convert to number.
NOTE: String '71.307144715' convert to number.
NOTE: String '59.846180435' convert to number.
NOTE: String '63.013744439' convert to number.
NOTE: String '59.198893492' convert to number.
NOTE: String '60.686242454' convert to number.
NOTE: String '59.041830014' convert to number.
NOTE: String '62.895729306' convert to number.
NOTE: String '57.160065856' convert to number.

mse = mse/nFold; /*1*/
rmse = sqrt(mse);
print "mse=", mse, " rmse=", rmse;
run;

1. Print the average mean squared error (MSE) and root mean squared error (RMSE) to the log.

Log Output: Average MSE and RMSE

mse=60.240942194 rmse=7.7615038616