Demo Steps: Getting Started with CAS and the R API

A financial services company offers a home equity line of credit to its clients. The company extended several thousand lines of credit in the past, and many of these accepted applicants (approximately 20%) defaulted on their loans. By using geographic, demographic, and financial variables, the company wants to build a model to predict whether an applicant might default.

After analyzing the data, the company selected a subset of 12 predictor (or input) variables to model whether each applicant defaulted. The response (or target) variable (**BAD**) indicates whether an applicant defaulted on the home equity line of credit. These variables, with their model role, measurement level, and description, are shown in the following table from the **HMEQ** data set:

Name	Model Role	Measurement Level	Description
BAD	Target	Binary	1 = applicant defaulted on loan or delinquent, 0 = applicant paid loan
CLAGE	Input	Interval	Age of oldest credit line in month
CLNO	Input	Interval	Number of credit lines
DEBTINC	Input	Interval	Debt to income ratio
DELINQ	Input	Interval	Number of delinquent credit lines
DEROG	Input	Interval	Number of derogatory reports
JOB	Input	Nominal	Occupational categories
LOAN	Input	Interval	Amount of loan request
MORTDUE	Input	Interval	Amount due on existing mortgage
NINQ	Input	Interval	Number of recent credit inquiries
REASON	Input	Binary	DebtCon = debt consolidation, HomeImp = home improvement
VALUE	Input	Interval	Value of current property
YOJ	Input	Interval	Years at present job

1. From Jupyter Lab, select **File Browser** > **Home** > **Courses** > **EVMLOPRC** > **Notebooks** and select the **R_Machine_Learning_Demo.ipynb** notebook.

2. Load the necessary packages.
   ```
       library(swat)
       library(ggplot2)
       library(reshape2)
       library(xgboost)
       library(caret)
       library(repr)
       options(repr.plot.width=5.5, repr.plot.height=5.5)
   ```
   - **SWAT** is an R package that enables you to interface with SAS Cloud Analytics Services (CAS), the in-memory server that is the centerpiece of the SAS Viya platform. Using the SWAT package, you can write an R program that connects to a CAS server, analyze large in-memory data sets, and then work with the results of the data analysis using familiar data-wrangling techniques in R.
   - **GGPLOT2** is an enhanced graphics language for the creation complex plots.
   - **RESHAPE2** provides functionality to restructure and aggregate data.
   - **XGBOOST** stands for extreme gradient boosting. This package enables users to fit a scalable form of gradient boosting models.
   - **CARET** provides functionality for classification and regression training. You use it to dummy-code variables in R.
   - **REPR** provides functionality to alter the size of graphics in Jupyter notebooks.
   - **OPTIONS** uses the **REPR** package to change the size of graphics.

3. Connect to CAS using the CASHOST, CASPORT, and SAS_VIYA_TOKEN arguments from the operating system environment.
   ```
       conn = CAS(Sys.getenv("CASHOST"), as.numeric(Sys.getenv("CASPORT")), password=Sys.getenv("SAS_VIYA_TOKEN"))
   ```
4. List the most recent CAS sessions.
   ```
       cas.session.listSessions(conn)
   ```
5. Change the CAS time-out for the session to 12 hours.
   ```
   mytime = 60*60*12
   cas.session.timeout(conn,time=mytime)
   cas.session.sessionStatus(conn)
   ```
6. Load the **hmeq.csv** data set onto the CAS server from the **/shared/home/YOUR_EMAIL/Courses/EVMLOPRC/DATA/** location and create a data table object called **castbl**. Also, create a variable to reference the in-memory data table, **HMEQ**, in subsequent code.
   ```
   castbl = cas.read.csv(conn, file=
             paste0(Sys.getenv("HOME"),"/Courses/EVMLOPRC/DATA/hmeq.csv"), casOut = list(name="hmeq", replace=TRUE))
   indata = 'hmeq'
   ```
   The cas.read.csv function is a wrapper for the read.csv function in R. However, the first argument is an instance of a CAS object that represents the CAS session. This object is used in all subsequent actions.

7. View how the data is distributed on the server.
   ```
   cas.table.tableDetails(conn,
       level="node",
       caslib="casuser(student)",
       name=indata
   )
   ```
   The data was loaded onto 16 active blocks.

```
cas.table.tableDetails(conn,
    level="block",
    caslib="casuser(student)",
    name=indata
)
```

The Rows column represents the number of observations that were loaded onto each individual node.

8. Plot the number of observations on each node. Use the Rows column as a vector and pass it to the plot function.

   ```
   node_data = cas.table.tableDetails(conn,
       level = "block",
       caslib = "casuser(student)",
       name = indata
   )$TableDetails$Rows

   node_data
   sum(node_data)

   plot(node_data, main="Distributed Data", xlab="Node", ylab=
   'Number of Observations', col="blue", type='b', lwd=2, pch=19)
   ```

   The variable **node_data** represents the vector of the number of observations on each node, and in total, it equals 5960 observations. CAS decides how the data is partitioned on each node. Notice that there are a different number of observations on each node.

9. Use the tableInfo action from the table action set to view the in-memory data tables on the CAS server. Also, use the class function on the castbl object and the head function applied to castbl.

   ```
   cas.table.tableInfo(conn)
   ```

   Only one data set, HMEQ, is currently on the server. No data tables persist from earlier sessions.

   ```
   class(castbl)
   class(head(castbl))
   ```

   This example represents three locations of data or results. The **hmeq** name is specific to the in-memory data table on the server. To use its original name in subsequent actions, you must specify the connection object for the CAS session (**conn**) and the table name (**hmeq**). The **castbl** object is a CASTable object reference for the in-memory **hmeq** data set. It can be used similarly to an R data frame and can be passed to subsequent R SWAT functions and CAS actions. The casDataFrame is available on both the server and client. A copy is brought to the client.

10. Find the number of functions that are available in the SWAT package and list the first few. Then use the Help function to view documentation for the as.casTable SWAT function.

    ```
    funcs = lsf.str("package:swat")
    length(funcs)
    head(funcs)
    help("as.casTable")
    ```

    The Help function loads documentation for the listed function.

11. Using SWAT functionality, explore the **HMEQ** data set.

    ```
    dim(castbl)
    names(castbl)
    colMeans(castbl)
    mean(castbl$BAD)
    summary(castbl)
    ```

12. Recall that the SWAT package simply uses CAS actions. Compare the head function to the fetch action and the nrow function to the recordCount action.

    ```
    head(castbl)
    cas.table.fetch(conn, table=indata, to=6)
    ```

    Notice that the head function is applied to the castbl object and the fetch action specifies the server data set name **HMEQ**.

    ```
    nrow(castbl)
    cas.table.recordCount(conn, table=indata)
    ```

13. Like open source languages, CAS requires you to load additional functionality onto the server. List the available action sets and go to the online documentation.

    ```
    listActionSets(conn)
    ```

    Note: Documentation about both the action sets and actions can be found on the following page: http://go.documentation.sas.com/?cdcId=pgmcdc&cdcVersion=8.11&docsetId=allprodsactions&docsetTarget=actionSetsByName.htm&locale=en.
    Select the link and then scroll down to select the **table** action set. Then select the **fetch** action.

    The documentation provides the syntax requirements and description of the syntax, examples of how to use the action, and details about the action.

    Note: The documentation provides the syntax for the CASL, Lua, Python, and R software languages. The documentation is also helpful to translate the action from one language to another.

14. Using CAS actions instead of SWAT functionality, explore the **HMEQ** data set.

    ```
    loadActionSet(conn, 'simple')
    ```

    The loadActionSet action loads the functionality onto the current session only. It is removed when the CAS session ends. Loading action sets prints the actions and a description of the actions. The online documentation provides much more detail for each action, and there is no Help functionality within the session for actions.
    For more detail about the individual actions and syntax requirements for each action, consult the online documentation.

    ```
    cas.simple.correlation(conn,
        table = indata,
        inputs = c("LOAN","VALUE","MORTDUE")
    )
    ```

    The correlation action provides both correlations between the listed inputs and simple summary statistics.

    ```
    cas.simple.distinct(conn,
        table = indata,
        inputs = names(castbl)
    )
    ```

    Notice that instead of listing variable names as the inputs, you can use the names function and pass the action a vector of variable names.

```
cas.simple.freq(conn,
    table = indata,
    inputs = c("BAD","JOB","REASON")
)
```

Notice that the first level of the variables **JOB** and **REASON** represents the missing level. There are no missing values for the target **BAD**.

```
cas.simple.crossTab(conn,
    table = indata,
    row = "BAD", col = "JOB"
)
loadActionSet(conn, 'cardinality')
cas.cardinality.summarize(conn,
    table = indata,
    cardinality = list(name='card', replace=TRUE)
)
cas.table.fetch(conn, table='card', to=5)
cas.table.recordCount(conn, table='card')
```

Because the **card** data set has no object reference, it must be analyzed using CAS actions. The **card** data set also includes skewness, kurtosis, and other summary information about the variables. The cardinality action set is useful before you decide which variables to transform or separate based on the level type, class, or interval.

15. Create an object reference to the in-memory **card** data set to apply the head and dim SWAT functions. You cannot use SWAT functionality on an in-memory data table without creating a reference.

```
card = defCasTable(conn, tablename = "card")
head(card)
dim(card)
```

16. Using the srs action from the sampling action set, visualize the data locally by first creating a subset of the **HMEQ** data set. Then use the ggplot R function to bring the sample to the client and create a panel of histograms of the numeric variables. Notice that, when you bring data to the client, you must be careful not to use too much of the client's RAM space. Therefore, it is advisable to first sample the data set before you bring it to the client.

```
loadActionSet(conn, 'sampling')
cas.sampling.srs(conn,
    table   = indata,
    samppct = 50,
    seed = 12345,
    partind = FALSE,
    output  = list(casOut = list(name = 'mysam', replace = T),
copyVars = 'ALL')
)
```

Selected arguments:

Argument	Description
table	specifies the in-memory input data table. The table is used in combination with the CAS session connection object.
samppct	specifies the sample percentage to be used for sampling or partitioning.
seed	specifies the integer to use to start the pseudorandom number generator.
partind	when set to *true*, generates a partition indicator column in the output table.
output	creates, on the server, a table that contains the sample output or partition output.
casOut	specifies the settings for an output table.
name	specifies the in-memory table name to save the action results.
replace	when set to ***true***, replaces the table with the results of the action.
copyVars	specifies a list of one or more variables to be copied from the input table to the output table. The keyword ALL copies all variables.

You sampled 50% of the original data and put it in the **mysam** data set on the server.

```
# Create connection object
mysam = defCasTable(conn, table='mysam')

# Bring data locally
df = to.casDataFrame(mysam)

# Use melt to help with data formatting
df = melt(df[sapply(df, is.numeric)], id.vars=NULL)

# Plot data with ggplot
ggplot(df, aes(x = value)) +  facet_wrap(~variable,scales =
   'free_x') + geom_histogram(fill = 'blue', bins = 25)
```

Next, you used the defCasTable function to create a reference to the mysam data table. The to.casDataFrame function downloads the in-memory table to the client and stores it in a local data frame. The melt function from the reshape2 package converts the wide data frame to a skinny data frame. This enables the ggplot function to create a panel of the variables instead of single graphics.

17. Check the variables for missing values by using the distinct action from the simple action set and create a table of the number of missing observations for each variable. Then locally plot the percentage of missing value for each observation.

```
tbl = cas.simple.distinct(castbl)$Distinct[,c('Column', 'NMiss')]
class(tbl)
tbl
```

The results of the distinct action are copied to the client. The class of the tbl results object is a data frame. This means that all functionality that is applied to the object can be native open source syntax.

```
'data.frame'
tbl$PctMiss = tbl$NMiss/nrow(castbl)
ggplot(tbl, aes(Column, PctMiss)) + geom_col(fill = 'blue') +
   ggtitle('Pct Missing Values') + theme(plot.title =
   element_text(hjust = 0.5), axis.text.x = element_text(angle = 90))
```

An additional column is added to the data frame. This column represents the missing percentage of each variable and then it is passed to ggplot. Again, the tbl object is local, so you can use local functions such as ggplot.

18. Use the impute action from the dataPreprocess actions set to impute missing values with the median for continuous variables and the mode for nominal variables.

```
cas.dataPreprocess.impute(conn,
    table = indata,
    methodContinuous = 'MEDIAN',
    methodNominal    = 'MODE',
    inputs           = colnames(castbl)[-1],
    copyAllVars      = TRUE,
    casOut           = list(name = indata, replace = TRUE)
)
```

Selected arguments:

Argument	Description
table	specifies the in-memory input data table. The table is used in combination with the CAS session connection object.
methodContinuous	specifies the imputation technique for interval variables.
methodNominal	specifies the imputation technique for nominal variables.
inputs	specifies the variables to use for the analysis.
copyAllVars	when set to *true*, specifies that all variables from the input table are copied to the output table.
casOut	specifies the settings for an output table.

The **HMEQ** data table now has 25 columns (the original 13 variables and now an additional 12 variables for the imputed inputs) by setting the copyAllVars argument equal to *TRUE*. Notice, from the ResultVar column, that all the new imputed variables begin with the IMP_ prefix followed by the original variable name. Setting the argument to *FALSE* would remove the original data and keep only the imputed variables.

19. Create variable shortcuts for the target, inputs, and nominal variables to avoid the need to enter variable names in future code.

```
colinfo = cas.table.columnInfo(conn, table=indata)$ColumnInfo
colinfo
```

The columnInfo action from the table action set provides the names of the variables in the data set as well as the variable type. Use the type variable to create separate sets of variables to avoid the need to enter the names repeatedly in subsequent code.

```
# Target variable is the first variable
target = colinfo$Column[1]

# Get all variables
inputs = colinfo$Column[-1]
nominals = c(target, subset(colinfo, Type == 'varchar')$Column)

# Get only imputed variables
inputs = grep('IMP_', inputs, value = T)
nominals = c(target, grep('IMP_', nominals, value = T))

# Print
target
inputs
nominals
```

Alternatively, you can manually code the target, inputs, and nominals.

```
target = 'BAD'
nominals = c('BAD','IMP_JOB','IMP_REASON')
inputs = c('IMP_CLAGE','IMP_CLNO','IMP_DEBTINC','IMP_DELINQ',
   'IMP_DEROG','IMP_LOAN','IMP_MORTDUE','IMP_NINQ','IMP_VALUE',
   'IMP_YOJ','IMP_JOB','IMP_REASON')
```