SAS® Visual Data Mining and Machine Learning 8.4: Advanced Topics

SAS® Documentation
September 17, 2019

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. *SAS® Visual Data Mining and Machine Learning 8.4: Advanced Topics*. Cary, NC: SAS Institute Inc.

SAS® Visual Data Mining and Machine Learning 8.4: Advanced Topics

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

September 2019

8.4-P1:vdmmladvug

Contents

1

Predictive Modeling

About This Document

This document presents overviews of the following:

- the types of predictive modeling algorithms and the preprocessing tools that accompany them.
- the functionality provided by SAS Visual Data Mining and Machine Learning. In particular, these topics align with the major functionalities that SAS Visual Data Mining and Machine Learning offers: data mining preprocessing and supervised learning. These topics contain details about the nodes themselves, as well as advice on best practices.

The audience is new and intermediate data scientists and analyst-level users using the Model Studio solution. Though the problem-solving approach is academic in nature, this document discusses only algorithms that are available to SAS Visual Data Mining and Machine Learning users.

These SAS Visual Data Mining and Machine Learning capabilities are a subset of the full Model Studio capabilities.

Overview of Data Mining Preprocessing

Effective machine learning models are built on a foundation of well-prepared data. Before cleaning and transforming the data, think about how the data will be used. Consider the problem at hand, the methods that you are using, and whether your data is appropriate in the first place. Shortcuts in data preparation will hamper your models.

Model Studio provides data preparation capabilities for SAS Visual Data Mining and Machine Learning in the form of pipeline nodes. These nodes form a group called Data Mining Preprocessing. You can use these nodes to do the following:

- Data and data role modification
- Dimension reduction
- Unsupervised learning

The table below describes some challenges that you might encounter in preparing your data. It also includes suggestions for how to handle the challenge by using the Preprocessing pipeline nodes in Model Studio.

Note: Some of these challenges can also be handled in the modeling stage, such as using tree-based methods for handling missing data automatically. Those are covered in the Selecting Your Algorithm on page 7.

Topic	Common Challenges	Suggested Best Practice
Data Collection	■ Biased data ■ Incomplete data ■ High-dimensional data ■ Sparsity	■ Take time to understand the business problem and its context ■ Enrich the data ■ Dimension reduction (Feature Extraction, Variable Clustering, and Variable Selection nodes) ■ Change representation of data (Transformations node)
“Untidy” Data	■ Value ranges as columns ■ Multiple variables in the same column ■ Variables in both rows and columns	■ Transform the data with SAS code (Code node)

Outliers	■ Out-of-range numeric values and unknown categorical values in score data	■ Discretization (Transformations node) ■ Winsorizing (Imputation node)
Sparse target variables	■ Low primary event occurrence rate ■ Overwhelming preponderance of zero or missing values in target	■ Proportional oversampling
Variables of disparate magnitudes	■ Misleading variable importance ■ Distance measure imbalance ■ Gradient dominance	■ Standardization (Transformations node)
High-cardinality variables	■ Overfitting ■ Unknown categorical values in holdout data	■ Binning (Transformations node) ■ Replacement (Replacement node)
Missing Data	■ Information loss ■ Bias	■ Binning (Transformations node) ■ Imputation (Imputation node)
Strong multicollinearity	■ Unstable parameter estimates	■ Dimension reduction (Feature Extraction, Variable Clustering, and Variable Selection nodes)

Preprocessing by Modifying Data

Modifying data is a broad preprocessing category. Any operation that alters the data or data roles can be considered as a modification, including dimension reduction techniques. For more information, see Preprocessing by Dimension Reduction on page 4.

Model Studio provides you with several SAS Visual Data Mining and Machine Learning nodes to modify your data:

Anomaly Detection	The **Anomaly Detection** node identifies and excludes anomalies using the support vector data description, or SVDD. Briefly, the SVDD formulation identifies outliers by determining the smallest possible hypersphere (built using support vectors) that encapsulates the training data points. The SVDD then excludes those data points that lie outside the sphere that is built from the training data. Anomaly detection with SVDD is useful for data sets where the

majority of the data belongs to one class and the other class is scarce or missing.

Filtering	The **Filtering** node excludes certain observations, such as rare values and outliers. Filtering extreme values from the training data tends to produce better models because the parameter estimates are more stable.
Imputation	The **Imputation** node replaces missing values in data sets. Simple imputation schemes include replacing a missing value in a particular input variable with the mean or mode of that variable's nonmissing values. For non-normally distributed variables or variables that have a high proportion of missing values, simple imputation might be ineffective. Imputation might also fail to be effective for variables whose missingness is not at random. For ideal results, create missing indicators and use them in the model alongside imputed variables. This practice can result in improved outcomes, even in cases where the variables are normally distributed and have few missing values.
Manage Variables	The **Manage Variables** node enables you to make modifications (such as changing the role of a variable or adding new transformations) to the data while within a Model Studio pipeline. The options available to you are a subset of the options available under the **Data** tab.
Replacement	The **Replacement** node enables you to replace outliers and unknown class levels with specified values. Much like with imputation, simple replacement of outliers and unknown class level is not always effective. Care should be taken to use replacement effectively.
Transformations	The **Transformations** node enables you to alter your data by replacing an input variable with some function of that variable. Transformations have many use cases. Transformations can be used to stabilize variances, remove nonlinearity, and correct non-normality.

Preprocessing by Dimension Reduction

Dimension reduction decreases the number of variables under consideration. In many applications, the raw data has very high dimensional features, and some features are redundant or irrelevant to the task. Reducing the dimensionality helps find the true, latent relationship. Model Studio provides you three nodes in SAS Visual Data Mining and Machine Learning for dimension reduction:

Feature Extraction	The **Feature Extraction** node transforms the existing features (variables) into a lower-dimensional space. Feature extraction in Model Studio is done using various techniques, including principal component analysis (PCA), robust PCA, singular value decomposition (SVD), and autoencoders. This is done by

generating new features that are composites of the existing features. One drawback to feature extraction is that the composite variables are no longer meaningful with respect to the original problem.

Variable Clustering — The **Variable Clustering** node divides numeric variables into disjoint clusters and chooses a variable that represents each cluster. Variable clustering removes collinearity, decreases redundancy, and helps reveal the underlying structure of the data set.

Variable Selection — The **Variable Selection** node uses several unsupervised and supervised methods to determine which variables have the most impact on the model. Supervised variable selection techniques include variable selection based on linear models and tree-based models (such as decision tree, forest, and gradient boosting). This tool enables you to specify more than one selection technique, and there are several options for selection criteria. Because there might be disagreements on selected variables when different techniques are used, specifying multiple selection methods helps ensure that important variables are consistently selected. Variables that fail to meet the selection criteria are marked as rejected and not used in successor modeling nodes.

Preprocessing by Unsupervised Learning

When performing unsupervised learning, the machine is presented with unlabeled data. (Unlabeled data has no target.) Unsupervised learning algorithms seek to discover intrinsic patterns that underlie the data, such as a clustering parameter or a redundant parameter (dimension) that can be reduced. For more information, see Preprocessing by Dimension Reduction on page 4.

Model Studio provides the **Clustering** node for processing data using *k*-means clustering.

Clustering — The **Clustering** node groups a set of data examples so that examples in one group (or one cluster) are more similar (according to some criteria) than those in other groups. Clustering is often used to segment a large data set into several groups. Analysis can be performed in each group to help users find intrinsic patterns.

Overview of Supervised Learning

Model Studio provides machine learning capabilities for SAS Visual Data Mining and Machine Learning in the form of nodes. These nodes form a group called Supervised Learning.

Supervised Learning algorithms make predictions based on a set of examples. For example, historical sales can be used to estimate the future prices. With supervised learning, you have an input variable that consists of labeled training data and a desired output variable. You use an algorithm to analyze the training data to learn the function that maps the input to the output. This inferred function maps new, unknown examples by generalizing from the training data to anticipate results in unseen situations. Model Studio supports two types of supervised learning problems:

- **Classification** — When the data is being used to predict a categorical target, supervised learning is called *classification*. This is the case when assigning a label or indicator (for example, labeling an image a dog or a cat). When there are only two labels, this is called *binary classification*. When there are more than two categories, the problems are called *nominal classification*.
- **Regression** — When the data is being used to predict interval targets, the problems become a regression problem.

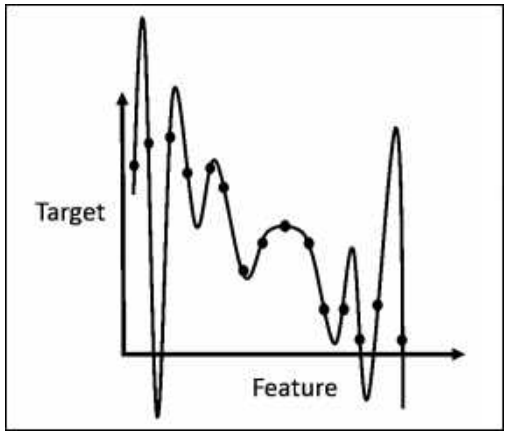
The following table groups the Supervised Learning nodes in Model Studio by possible target type. Nodes that are listed in both columns support both target types.

Interval Target	Nominal and Binary Target
Linear Regression	Logistic Regression
Decision Tree	Decision Tree
Forest	Forest
Gradient Boosting	Gradient Boosting
Neural Network	Neural Network
GLM (Generalized Linear Model)	SVM (only for binary target)
Quantile Regression	Bayesian Network

The following sections contain general information about applying best practices to using the supervised learning nodes.

Overfitting

Machine learning algorithms are very effective at learning a mapping between the features and known target values in your existing data. A model that is complex enough to perfectly fit the existing data might not generalize well when used to score new observations. This is referred to as *overfitting*. If left unattended, the models can overfit and create a 100% accurate mapping, as shown below.



Good models strive to achieve low training error, but it is just as important to achieve low generalization error. The training process needs to account for this compromise and make an honest assessment of the accuracy of the model. Assessing a candidate model on the data that is used to train the model would direct the algorithm to overfit to that training data. Instead of doing this, consider using validation, testing, and holdout data, or a combination of these methods:

Validation Data	Validation data is data that is used to assess the model *during* training for the purpose of selecting variables and adjusting parameters. Validation data sets are instrumental in preventing overfitting. In lieu of a separate validation set (which might not be feasible for smaller data sets), SAS Visual Data Mining and Machine Learning offers *k*-fold cross validation through the Autotuning capability. Whether through a validation set or through *k*-fold cross validation, ensure that the training process assesses the error on data that is not used to train the model.
Test Data	Test data is data that is used *at the end* of model fitting to obtain a final assessment of how the model generalizes to new data. The reason for using test data (instead of validation data) is that validation data plays a role in the model training process. Hence, using validation data might lead to the same biased assessments as using training data. For this reason, a test data set should be used only at the end of the analysis and should not play a role in the model training process.
Holdout Data	Holdout data is a capability that is new to Model Studio. Since test data was used in model comparison, using it again in comparing different pipelines might introduce bias. By setting aside data for the holdout partition, you allow for a further safeguard against generalization error.

Selecting Your Algorithm

When you are presented with a data set, the first thing to consider is how to obtain results, no matter what those results might look like. Users with less experience tend to choose algorithms that are easy to implement and that produce results quickly. This approach is acceptable, if it is the first step of the process. After you

obtain some results and become more familiar with the data, you might spend more time experimenting with more sophisticated algorithms. This might strengthen your understanding of the data, and potentially further improve the results.

Even in this stage, the best algorithms might not be the methods that have achieved the highest reported accuracy. Most algorithms usually require careful tuning and extensive training to obtain the best achievable performance. The following table presents some best practices for selecting SAS Visual Data Mining and Machine Learning supervised learning algorithms.

Algorithm Type	Target Type	Suggested Usage	Suggested Scale	Interpretability	Automatic Hyperparameter Tuning Capability	Common Concerns
Regression (Linear, Logistic, GLM, Quantile)	■ Linear regression, Quantile regression, and GLM for interval target ■ Logistic regression for nominal and binary target	■ Modeling linear or linearly separable phenomena ■ Manually specifying nonlinear and explicit interaction ■ LASSO regression includes a regularization term for linear and logistic regression to deal with multicollinearity and overfitting issues	Small to large data sets	High	No	■ Missing values ■ Outliers ■ Standardization ■ Parameter tuning

SVM	■ Binary	■ Modeling linear or linearly separable phenomena by using linear kernels or polynomial kernels up to degree three	Small to large data sets	Low	Yes	■ Missing values ■ Overfitting ■ Outliers ■ Standardization ■ Parameter tuning

Tree-based Modeling (Decision Tree, Forest, Gradient Boosting)	■ Interval ■ Binary ■ Nominal	■ Modeling nonlinear and nonlinear separable phenomena in large data sets ■ Interactions considered automatically, but implicitly ■ Missing values and outliers in input variables handled automatically in many implementations ■ Tree ensembles (forests, gradient boosting) can increase prediction accuracy and decrease overfitting, but also decrease scalability and interpretability	Medium to large data sets	Moderate	Yes	■ Instability with small training sets ■ Gradient boosting can be unstable with noise or outliers ■ Overfitting ■ Parameter tuning

Neural Network	■ Interval ■ Binary ■ Nominal	■ Modeling nonlinear and nonlinearly separable phenomena ■ All interactions considered in fully connected, multilayer topologies	Medium to large data sets	Low	Yes	■ Missing values ■ Overfitting ■ Outliers ■ Standardization ■ Parameter tuning
Bayesian Network	■ Binary ■ Nominal	■ Modeling linearly separable phenomena in large data sets ■ Well suited for extremely large data sets where complex methods are intractable	Small to extremely large data sets	Moderate	No	■ Linear independence assumption ■ Infrequent categorical levels

Assessment Tools

Assessment tools are provided in the results of supervised learning nodes in Model Studio to evaluate the efficacy of your models. For a supervised learning node that has a nominal target, Lift Reports and ROC Reports are produced. For a supervised learning node that has a nominal or binary target, a quantile binning of the predictions is performed. Then, summary statistics of the response variable for each bin are computed. Assessment tools also include fit statistics such as the following:

- Average square error
- Mean square logarithmic error
- Mean absolute error
- Mean consequential error
- Multiclass log loss

For more information, see the PROC ASSESS documentation in *SAS Visual Statistics 8.3: Procedures Guide*.

Assessment Plots for Interval Target

- Actual and Predicted by Depth
- Predicted by Actual

Assessment Plots for Nominal and Binary Target

- **Lift Reports**
  - Cumulative Lift
  - Lift
  - Gain
  - Captured Response Percentage
  - Cumulative Captured Response Percentage
  - Response Percentage
  - Cumulative Response Percentage
- **ROC Reports**
  - ROC
  - F1 Score
  - Accuracy
- **Event Classification** Report

Assessment Measures

For a complete list of the assessment measures available, see the Overview of Model Comparison in the Model Studio reference documentation.

Model Interpretability Plots

Overview

As machine learning models become more sophisticated, the ability to quickly and accurately interpret these models can diminish. SAS Visual Data and Machine Learning provides three plots that help users interpret model results. The Local Interpretable Model-agnostic Explanation (LIME), Individual Conditional Expectation (ICE), and Partial Dependence (PD) plots help you improve your model by providing

a better understanding of the results. Each of these plots work for all models and are used to compare results across many different models.

Local Interpretable Model-agnostic Explanations

A LIME plot creates a localized linear regression model around a particular observation based on a perturbed sample set of data. That is, near the observation of interest, a sample set of data is created. This data set is based on the distribution of the original input data. The sample set is scored by the original model and sample observations are weighted based on proximity to the observation of interest. Next, variable selection is performed using the LASSO technique. Finally, a linear regression is created to explain the relationship between the perturbed input data and the perturbed target variable. The final result is an easily interpreted linear regression model that is valid near the observation of interest.

Partial Dependence

A PD plot depicts the functional relationship between the model inputs and the model's predictions. A PD plot shows how the model's predictions partially depend on the values of the input variables of interest. To create a one-way PD plot, identify the plot variable and the complementary variables. Next, create a replicate of the training data for each unique value of the plot variable. In each replicate, the plot variable is replaced by the current unique value. Finally, score each replicate with your model and compute the average predicted value within each replicate. The final result is a view of how the prediction changes with respect to the plot variable.

Individual Conditional Expectation

An ICE plot presents a disaggregation of the PD plot to reveal interactions and differences at the observation level. The ICE plot is generated by choosing a plot variable and replicating each observation for every unique value of the plot variable. Then, each replicate is scored. SAS Visual Data Mining and Machine Learning creates a segmented ICE plot. A segmented ICE plot is created from a cluster of observations instead of on individual observations.

The two most useful features to observe when evaluating an ICE plot are intersecting slopes and significant differences between each cluster's plot. Intersecting slopes indicate that there is an interaction between the plot variable and one or more complementary variables. Significant differences between each cluster's plot indicate group effects. That is, the differences in the plot indicate that there are significant differences between the clusters.

Handling Rare Events

In data mining, predictive models are often used to detect rare classes. For example, an application to detect credit card fraud might involve a data set containing 100,000 credit card transactions, of which only 100 are fraudulent. Due to noise, it is possible that no transaction will have a posterior probability over 0.5 of being fraudulent. Hence, simply classifying cases according to posterior probability will yield no transactions classified as fraudulent.

When you are collecting the original data, it is always good to over-sample rare classes if possible. If the sample size is fixed, a balanced sample (that is, a sample with equal sizes for each class) will usually produce more accurate predictions than an unbalanced split. For example, if you can sample any 100,000 customers, it would be much better to have 50,000 responders and 50,000 nonresponders than to have 5,000 responders and 95,000 nonresponders.

Unfortunately, balanced sampling is often impractical. Model Studio accounts for this problem by oversampling the rare case observations and adding a posterior probability adjustment for priors in the score code. To do this:

1. When creating a project, in the New Project Settings window, select **Enable event-based sampling**. Alternatively, if the project has already been created, you can select **Enable event-based sampling** via the Project Settings window.
2. In the Project Settings window, set the event and non-event percentage. The two values must sum to 100.
3. Set up your pipeline. After running the pipeline, examine the score code. The score code contains a section titled **Adjust Posterior Probabilities**. This code block modifies the posterior probability by multiplying it by the ratio of the actual probability to the event-based sampling values specified previously.

Scoring Your Models

Model Studio creates SAS language score code for the purpose of scoring new data. You can run this code in your production systems to make business decisions for each record of new data.

There are two types of score code that Model Studio nodes can create: DATA step or analytic store. To generate score code for an entire pipeline, the score code for each node producing score code is appended together into a single DATA step. When the nodes in a pipeline produce multiple analytic stores, or one or more analytic stores and DATA step score code, an EP score code file is created. EP score code represents the score code produced by these pipelines. To run this code outside Model Studio, see Running Your Score Code From Analytic Store Models on page 16 .

The following table demonstrates which Model Studio nodes produce score code, as well as the types of code they produce.

Node Name	Type of Score Code
Anomaly Detection	Analytic store
Bayesian Network	Analytic store
Clustering	DATA step
Decision Tree	DATA step
Ensemble	DATA step (if all models in the ensemble produce DATA step), otherwise one or more analytic stores and the EP score code to combine the models' score code
Feature Extraction	DATA step
Filtering	DATA step
Forest	Analytic store
GLM	DATA step
Gradient Boosting	Analytic store
Imputation	DATA step
Linear Regression	DATA step
Logistic Regression	DATA step
Neural Network	DATA step for networks with less than 6 layers, analytic store for networks with 6 or more layers
Quantile Regression	DATA step
Replacement	DATA step
SVM	Analytic store
Text Mining	Analytic store
Transformations	DATA step

Running Your Score Code from Analytic Store Models

Certain models in SAS Visual Data Mining and Machine Learning produce EP score code. If your model generates EP score code, complete the following steps to score new data:

1. Download the score code from your champion model. Save this ZIP file onto the server that contains the SAS Viya installation. The ZIP file will contain a SAS program called dmcas_epscorecode.sas. For more information about this process, see Export Models for Production in the *SAS Visual Data Mining and Machine Learning: User's Guide*.

2. Sign on to SAS Studio. Start a CAS session that connects to the same server used by Model Studio. Your SAS administrator can provide you with the name of the CAS server. For example, the code below connects to the CAS server on port 5570:

```
/* start a CAS session and assign the libnames */
options cashost="<myCASserver>" casport=5570;
cas mysess;

cas;
caslib _all_ assign;
```

3. Use PROC CASUTIL to load the SASHDAT file into your SAS computing environment.

```
/* load the analytic store table for scoring */
proc casutil;
    load casdata="example_data_for_scoring_ast.sashdat" inCASlib="models"
    casOut="exampleCAStable" outCASlib=casuser replace;
quit;
```

Note: To determine the name of the analytic store for the CASDATA= parameter, open dmcas_epscorecode.sas. In the top comments section, you should see a name ending with _ast. This is the analytic store filename. Because Linux is case sensitive, you must convert any letters in the filename to uppercase.

4. Use PROC ASTORE to run the EP score code against the data.

```
proc astore;
    score data=casuser.exampledata
          rstore=casuser.exampleCAStable
          epcode="</u/userName/>example_epscorecode.sas"
          out=casuser.exampledataout;
quit;
```

Here, example_epscorecode.sas denotes the EP score code downloaded from your model. The location specified in the EPCODE parameter must point to a location where you have Write access.

2

Autotuning Machine Learning Models

Overview of Autotuning

To create a good statistical model, many choices have to be made when deciding on algorithms and their parameters. The usual approach is to apply trial-and-error methods to find the optimal algorithms for the problem at hand. Often, a data scientist chooses algorithms based on practical experience and personal preferences. This is reasonable, because usually there is no unique and relevant solution to create a machine learning model. Many algorithms have been developed to automate manual and tedious steps of the machine learning pipeline. Still, it requires a lot of time and effort to build a machine learning model with trustworthy results.

A large portion of this manual work relates to finding the optimal set of hyperparameters for a chosen modeling algorithm. *Hyperparameters* are the parameters that define the model applied to a data set for automated information extraction.

For example, when data scientists build a machine learning model to predict which customers are good credit risks, they must decide about the following during the training process:

- which modeling approaches to test
- which data to choose to train the model
- which data to test the results
- how to tune the parameters of the chosen model
- how to validate the results

All these choices affect the outcome of the model building exercise, and eventually the final model selected. Since this model is used to decide which customers get

credit, it is vital that there is high confidence in the model to make trustworthy decisions.

A large portion of the model building process is taken up by experiments to identify the optimal set of parameters for the model algorithm. As algorithms get more complex (neural networks to deep neural networks, decision trees to forests and gradient boosting), the amount of time required to identify these parameters grows.

There are several ways to support the data scientist in this cumbersome work of tuning machine learning model parameters. These approaches are called *hyperparameter optimization*.

In general, there are three different types: parameter sweep, random search, and parameter optimization.

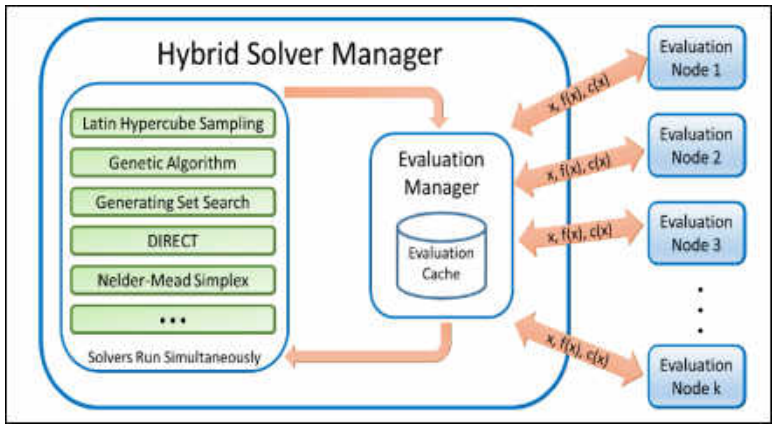
Parameter sweep:
: This is an exhaustive search through a predefined set of parameter values. The data scientist selects the candidates of values for each parameter to tune, trains a model with each possible combination, and selects the best-performing model. Here, the outcome very much depends on the experience and selection of the data scientist.

Random search:
: This is a search through a set of randomly selected sets of values for the model parameters. This can provide a less biased approach to finding an optimal set of parameters for the selected model. Since this is a random search, it is possible to miss the optimal set unless a sufficient number of experiments are conducted, which can be expensive.

Parameter optimization:
: This is the approach that applies modern optimization techniques to find the optimal solution. It is the best way to find the most appropriate set of parameters for any predictive model, and any business problem, in the least expensive way.

SAS has conducted research in the area of hyperparameter tuning. In SAS products, these capabilities are referred to as autotuning. Model Studio provides autotuning capabilities to SAS Visual Data Mining and Machine Learning users. This offering provides a hyperparameter autotuning capability that is built on local search optimization (LSO) in SAS software.

LSO in SAS is a hybrid, derivative-free optimization framework that operates in the SAS Viya parallel and distributed computing environment to overcome the challenges and computational expense of hyperparameter optimization. It consists of an extendable suite of search methods. Evaluations of different model configurations are distributed across multiple evaluation worker nodes in a compute grid. These nodes are coordinated in a feedback loop that supplies data from all concurrent running search methods.



The autotuning capability in SAS Visual Data Mining and Machine Learning takes advantage of the LSO framework to provide a flexible and effective hybrid search strategy. The default search strategy begins with a Latin hypercube sample (LHS), which provides a more uniform sample of the hyperparameter space than a grid or random search provides. The best samples from the LHS are then used to seed a genetic algorithm (GA), which crosses and mutates the best samples in an iterative process. This generates a new population of model configurations for each iteration. Importantly, SAS Viya can evaluate the LHS samples in parallel, and the GA population at each iteration can also be evaluated in parallel. Alternate search methods include a single LHS, a purely random sample, a grid search, and a Bayesian search method.

In SAS Visual Data Mining and Machine Learning, autotuning is used by the following nodes:

- **Bayesian Network**
- **Decision Tree**
- **Forest**
- **Gradient Boosting**
- **Neural Network**
- **SVM**

The parameters that can be tuned are given in the table below:

Bayesian Network				
	Parameter	Minimum Value	Maximum Value	Notes
	Network Structure	not applicable	not applicable	Network Structure is evaluated from a specified list.
	Maximum Parents	1	16	Integers
	Parenting Method	not applicable	not applicable	Parenting Method is evaluated from a specified list.
	Number of Bins	2	20	Integers

Decision Tree				
	Parameter	Minimum Value	Maximum Value	Notes
	Maximum Depth	1	19	Integers
	Interval Input Bins	20	200	Integers
	Grow Criterion	not applicable	not applicable	Grow criteria are evaluated from a specified list.
Forest				
	Parameter	Minimum Value	Maximum Value	Notes
	Maximum Depth	1	150	Integers
	Number of Trees	1	1000	Integers
	In-bag Sample Proportion	0 (not inclusive)	1	Real values
	Number of Inputs per Split	1	Number of inputs.	Integers
Gradient Boosting				
	Parameter	Minimum Value	Maximum Value	Notes
	L1 Regularization	0	not applicable	Real values
	L2 Regularization	0	not applicable	Real values
	Learning Rate	0 (not inclusive)	1	Real values
	Number of Inputs per Split	1	Number of inputs.	Real values
	Number of Iterations	1	10000	Integers
	Subsample Rate	0 (not inclusive)	1	Real values
Neural Network				
	Parameter	Minimum Value	Maximum Value	Notes
	Number of Hidden Layers	0	5	Integers
	Number of Neurons in Each Hidden Layer	1	1000	Integers

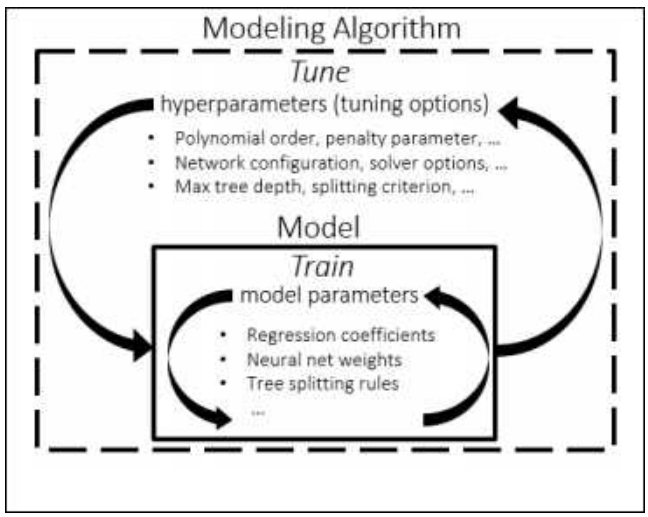
	L1 Weight Decay	0	not applicable	Real values
	L2 Weight Decay	0	not applicable	Real values
	Learning Rate	0 (not inclusive)	not applicable	Real values
	Annealing Rate	0 (not inclusive)	not applicable	Real values
SVM				
	Parameter	Minimum Value	Maximum Value	Notes
	Penalty	.000001	100000	Real values
	Polynomial Degree	1	3	Integers

Parameters and Hyperparameters

Before performing autotuning, you must determine which hyperparameters are to be tuned. Settings for these hyperparameters can significantly influence the resulting accuracy of the predictive models, and there are no clear defaults that work well for different data sets. The machine learning algorithms themselves also have many hyperparameters. For example, for a neural network, this includes (but is not limited to) values such as the following:

- Number of hidden layers
- Number of neurons in each hidden layer
- Distribution used for the initial weights

These values govern the quality of the resulting model, and the ideal values vary widely from data set to data set.

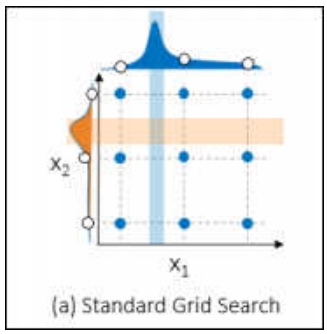


Search Options

Once you have chosen which hyperparameters to autotune, you must select the search method. The following search options are available:

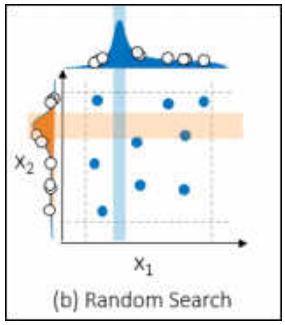
- Grid Search: a typical approach to exploring alternative model configurations is by using a grid search. In a grid search, each hyperparameter of interest is discretized into a desired set of values to be studied. Models are trained and assessed for all combinations of values across all the hyperparameters (thus forming a multi-dimensional “grid”).

  Though simple and straightforward to carry out, a grid search is computationally costly, with expense that grows exponentially with the number of hyperparameters and number of discrete values in each. Thus, for the grid search to be feasible, the grid must be quite coarse, and might fail to identify an improved model configuration. The following figure shows hypothetical distributions of two hyperparameters X_1 and X_2 with respect to a training objective. As seen, there is difficulty in finding a good combination with a coarse standard grid search.

- Random Search: a simple alternative to performing a grid search is to train and assess candidate models using a random search, that is, random combinations of hyperparameter values. Because some of the hyperparameters might actually have little effect on the model for certain data sets, it is prudent to avoid wasting the effort to evaluate all combinations. This is especially important for higher-dimensional hyperparameter spaces. Random combinations enable you to explore more values of each hyperparameter at the same cost.

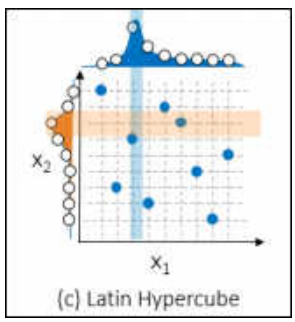
  The figure below depicts a potential random distribution with the same budget of evaluations (nine points, as in the grid search), highlighting the potential to find better hyperparameter values. Still, the effectiveness of a purely random search is subject to the size and uniformity of the sample; candidate combinations can be concentrated in regions that completely leave out the most effective values of one or more of the hyperparameters.



(b) Random Search

- Latin Hypercube Sampling: a similar but more structured approach is to use a random Latin hypercube sample, or LHS. The Latin hypercube sample is a combinatorial object that selects values in a uniform way across each hyperparameter but random in combinations. This criterion ensures that points are approximately equidistant from each other in order to fill space efficiently. This sampling allows for coverage across the entire range of the hyperparameter and is more likely to find good values of each hyperparameter. Good values for each hyperparameter can then be used to identify good combinations.

  In the figure below, note that no X_1 or X_2 value is repeated in sampling; equivalently, note that no horizontal or vertical line goes through more than one point.



(c) Latin Hypercube

- Genetic Algorithm: the Genetic Algorithm (GA) initially uses Latin Hypercube Sampling. The best samples from the LHS are then used to seed a GA, which generates a new population of alternative configurations at each iteration. The GA is the default search option.
- Bayesian: the Bayesian search method builds a kriging surrogate model to approximate the objective value and uses this surrogate model to generate new alternative configurations at each iteration. The kriging model is continuously updated during the search process.

Validation Options

Another aspect of hyperparameter tuning involves cross validation. For small data sets, a single validation partition might leave insufficient data for validation in addition to training. Keeping the training and validation data representative can be a challenge. For this reason, cross validation is typically recommended for model validation. With cross validation, the data are partitioned into *k* approximately equal subsets called *folds*. Training and scoring happens *k* times, training on all *k-1* folds except the holdout fold, and then scoring on that remaining fold. The cross

validation error is then given as the average of all the errors obtained from each validation fold.

This process can produce a better representation of error across the entire data set, because all observations are used for training and scoring. With this cross validation process, the trade-off is increased time.

References

Koch, Patrick, et al. 2017. “Automated Hyperparameter Tuning for Effective Machine Learning.” *Proceedings of the SAS Global Forum 2017 Conference.* Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings17/SAS0514-2017.pdf.

Schubert, Sascha. “Autotuning: How machine learning helps optimize itself.” Available https://blogs.sas.com/content/sascom/2017/05/01/autotuning-machine-learning-helps-optimize/. Last modified May 1, 2017. Accessed on October 4, 2017.

Koch, Patrick, et al. 2018. “Managing the Expense of Hyperparameter Autotuning.” *Proceedings of the SAS Global Forum 2018 Conference.* Cary, NC: SAS Institute Inc. Available https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1941-2018.pdf

3

Administration

Promotions and Upgrades within SAS Viya

Promotions Considerations

A *promotion* is the process of making resources that exist in one environment present, available, and usable in another environment. The promotion process consists of exporting the resources from the source environment and then importing the resources to the target environment. For more information, see Promotion: Overview.

Consider the following information before performing a promotion:

- The owner of a project that is being promoted must sign in to the target environment before any projects can be imported. If you are a project owner, it is recommended that you promote your own individual projects.
- Before you promote a project, you must promote the input data for that project to the target environment.
- A user-created pipeline or node template must be promoted separately before you can use it in a new project. If a user-created pipeline or node template was used in a promoted project, that user-created content is available on the target environment.

- If a project contains a pipeline that was created by SAS Visual Analytics using the **Create Pipeline** interface, that pipeline is not promoted. All other pipelines in the project are promoted.

  You can re-create the pipeline by promoting the SAS Visual Analytics report that was used to create the pipeline. You must follow the steps outlined in the SAS Viya Administration Guide to promote the report. If your model contains an analytic store, pay attention to the information in Details: Reports That Contain ASTORE Tables so that you can use this report.

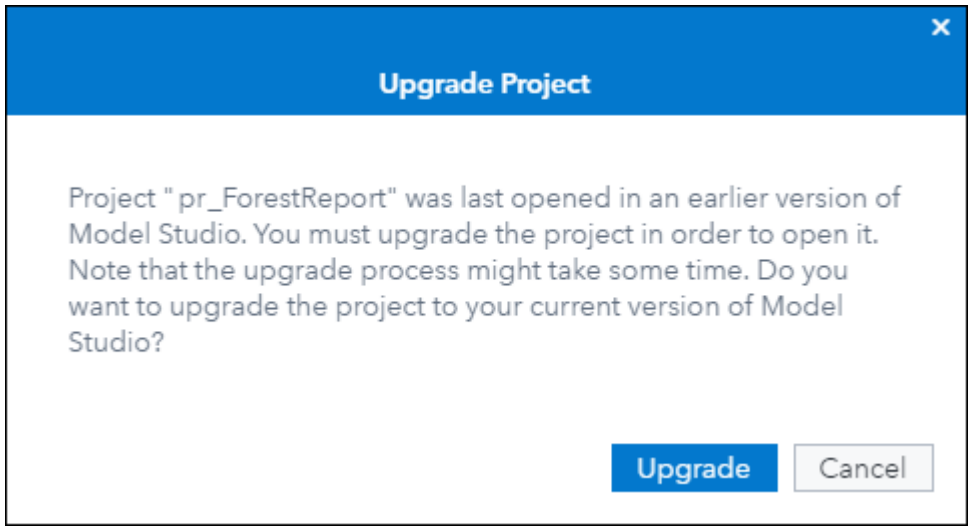
  After the SAS Visual Analytics report and the Model Studio project are promoted, you can use the **Create Pipeline** interface to insert the SAS Visual Analytics model into the promoted Model Studio project.
- In Model Studio 8.2, you can import score code from the **Pipeline Comparison** tab. That score code is not promoted.
- You must rerun all nodes and pipelines in a promoted project before the results are available on the target environment.

Upgrade Considerations

An *upgrade* to Model Studio adds significant feature changes or improvements to the product.

Consider the following information before performing an upgrade:

- If you are upgrading Model Studio within the same version of SAS Viya, see "Adding SAS Viya Software to a Deployment and Upgrading Products in SAS Viya 3.4" in *SAS Viya for Linux: Deployment Guide* for more information.
- If you are upgrading Model Studio in addition to upgrading SAS Viya, see "Upgrading to SAS Viya 3.4 from Earlier Versions of SAS Viya" in *SAS Viya for Linux: Deployment Guide* for more information.
- After all the steps have been completed in the *SAS Viya for Linux: Deployment Guide* and Model Studio or SAS Viya has been upgraded, users can upgrade their individual projects. To upgrade a project:
  - Sign in to Model Studio. The icon in the lower left corner of the project tile indicates that the project has not been upgraded.
  - Open the project that you want to upgrade, and click the **Upgrade** button in the Upgrade Project window.



- When a shared project is upgraded, it becomes a private project. After you upgrade a project, you must re-share it. It is recommended that you take note of all your shared projects, and with whom they are shared, before upgrading.
- If you are the project owner, you must upgrade the projects that you created. SAS Administrators cannot upgrade projects that are created by other users.
- Before you upgrade a project, you must load the input data for that project to the target environment.
- After your project is upgraded and you run your pipelines, the models in the project are no longer registered. You must re-register and re-publish your models.

Upgrading Model Studio 8.2 Projects with User-Defined Partitioning

After you upgrade a project that contains a user-defined partition variable, you cannot run any pipelines in that project until the data advisor generates a partition level report. If you attempt to run your pipeline, the following error message is generated: `The partition variable levels have not been set for the variable named: _partInd_`. In Model Studio 8.2, the partition variable was defined by a column that was named _partInd_. You could not otherwise specify a partition variable.

If you have already upgraded a project that contains a user-defined partition variable, do the following:

1. Open the project and view the **Data** tab.
2. Select any variable with the role **Input** and change that variable's role to **Rejected**. This action triggers the data advisor to generate the partition level report.
3. Undo the action in the previous step. This ensures that your chosen variable is still assigned the role **Input**.
4. If your project was derived from SAS Visual Analytics, change the variable role to **Rejected** for any variables that you identified in step 8 of "Upgrading Model

Studio 8.2 Projects Derived from SAS Visual Analytics Reports with Partitioning" on page 30.

You can confirm that the report was generated by browsing the **Project logs** and locating the **Log for Setting Variable Partition Levels**. After forcing the data advisor to generate the partition level report, you should be able to run the pipelines in your project.

Upgrading Model Studio 8.2 Projects Derived from SAS Visual Analytics Reports with Partitioning

If your project contains a partition variable that was derived in SAS Visual Analytics, additional actions are necessary before Model Studio can be upgraded. When SAS Visual Analytics creates a partitioned data source, it builds the table in memory and associates it with the CASUSER library. This table is never persisted and is rebuilt each time that SAS Visual Analytics starts a new session. This causes problems in Model Studio 8.4 because the **Data** tab depends on the source table for a number of operations.

A SAS Administrator must follow the instructions in "Identify Model Studio Projects Derived from SAS Visual Analytics Reports with Partitioning" in *SAS Viya for Linux: Deployment Guide* to determine which projects contain a partition variable that was derived in SAS Visual Analytics.

The project owner must now complete the following steps for each project identified:

Important: These steps must be completed before SAS Viya and Model Studio are upgraded.

1. Open the project and click the **Pipelines** tab. Select any pipeline.
2. Right-click the **Data** node and select **Add below** ⇨ **Miscellaneous** ⇨ **Save Data**.
3. Select the **Save Data** node. In the options panel, click **Browse** to select an **Output library** (for example, the Public library).
4. In the options panel, specify a **Table name**. The name that you specify must be 32 characters or fewer.
5. In the options panel, select **Replace existing table** and **Promote table**.
6. Run the **Save Data** node.
7. After the run completes, select the **Data** tab.
8. Identify any variables that are assigned the role **Rejected**. Here are some variables that might be assigned the role **Rejected**:
   - the original target variable from SAS Visual Analytics, if it had been converted to a new variable type
   - _PARTITION_
   - _va_d__PARTITION__ONES

9 Click the **Change data source** icon in the upper right corner of the window. You might need to click **Refresh** to see the newly created table.

10 Select the table that you created with the **Save Data** node and click **OK**.

11 Remove the **Save Data** node from your pipeline.

The project is now retrained on the table and you are ready to upgrade. To run your pipelines when the upgrade is complete, you must complete the steps listed in Upgrading Model Studio 8.2 Projects with User-Defined Partitioning on page 29.

If the partitioned data source from SAS Visual Analytics was not persisted before the upgrade occurred, additional steps are necessary to find and import the data source into your project. Several of these steps must be completed by a SAS Administrator.

First, determine the name of the project caslib as noted below. These steps can be completed by either the project owner or a SAS Administrator:

1 Open the project. You are prompted to upgrade the project the first time that you open it. An error message is generated: `The requested resource was not found. Check the application's error logs or contact your system administrator for assistance.`

2 Click the ⚙ icon in the upper right corner of the window, under the user name button, and click **Project logs**.

3 In the Available Logs window, select the **Log for the Project Data Advisor**, and click **Open**.

4 In the first few lines of the log, search for the value **dm_projectId**. An example is 577d49c6-a95d-4efa-88de-3f298d9d3ab6. In this example, the project caslib is named datamining-577d49c6-a95d-4efa-88de-3f298d9d3ab6.

After determining the name of the project caslib, copy the project partitioned table to a caslib that can be viewed by the project owner. These steps must be completed by a SAS Administrator:

1 The standard location of the machine where CAS runs and where all caslibs of the SAS Visual Data Mining and Machine Learning projects live is **`/opt/sas/viya/config/data/cas/default/projects`**. The project caslib name that you previously determined is a subdirectory in this location.

2 The partitioned table is a SASHDAT file that has a name with the prefix DM_. An example is DM_56GQP8AITE1WQE9J3P1YV9BQU.sashdat.

3 Copy the file to another caslib directory that the project owner can view. An example is the public directory: **`/opt/sas/viya/config/data/cas/default/public`**. It is recommended that you rename the file to something more meaningful.

After the partitioned table is moved to an accessible location, the table must be imported as the data source table of the project. These steps must be completed by the project owner:

1 Open the project. An error message is generated: `The requested resource was not found. Check the application's error logs or contact your system administrator for assistance.`

2 Make a note of the assigned variable roles on the **Data** tab.

3 Click the icon in the upper left corner of the window.

4 Click the icon in the upper left corner of the Data sources pane.

5 Click the **Data Sources** tab and navigate to the library where the partitioned table is saved.

6 Select the table and then click the icon in the upper right corner of the window to load the table.

7 Click **OK** to complete the project retrain. The **Data** tab now displays the metadata of the partitioned table.

8 Set the variable role of _dmIndex_ to **Rejected**.

9 Modify the variable role of any other variable that does not match the variable role that you noted in step 2. You can now run your pipelines.

Promoting from Model Studio 8.2 to Model Studio 8.3 and Later

You can promote projects, pipeline templates, and node templates from Model Studio 8.2 to Model Studio 8.3 and later. Before promoting your content from Model Studio 8.2, consider the following information:

- Before promoting from Model Studio 8.2, you must first apply the latest software update on the Model Studio 8.2 server. See SAS Note 62339 to obtain this hot fix.
- To promote a project or template from Model Studio 8.2 to Model Studio 8.3 and later, you must follow the CLI instructions found in Promotion within SAS Viya: Instructions.

Promoting from Model Studio 8.3 to Model Studio 8.3 and Later

You can promote projects, pipeline templates, and node templates from Model Studio 8.3 to Model Studio 8.3 and later. Before promoting your content from Model Studio 8.3, consider the following information:

- If you need to quickly promote a single project within the same version of Model Studio, use the instructions in Importing and Exporting a Project to export the project from the source environment and import the project on the target environment.
- To promote a project or template from Model Studio 8.3 to Model Studio 8.3 or later, you can follow either the CLI instructions or the Wizard instructions found in Promotion within SAS Viya: Instructions. In either case, all pipelines in the promoted projects must be rerun on the target system.

Note: If you derive a custom template from another template in the exchange, both templates are required to be on the source system in order to successfully import or export a project.

Configuration Properties

Overview

This section covers configuration settings that are specific to SAS Visual Data Mining and Machine Learning. For more information about SAS Viya configuration properties, see SAS Viya Administration: Configuration Properties.

Job Polling Properties

The following job polling properties regulate the number of times that a CAS polling call can be made and the length of time that a CAS session is kept open.

max.job.polling.attempts
: The set of properties for the Max Polling Job Attempts service.

  maxJobPollingAttempts
  : The number of requests that can be made.

JOB_SLEEP_INTERVAL
: The length of time in milliseconds between calls to the CAS session. The value of this property multiplied by the value of maxJobPollingAttempts determines the length of time that the CAS session remains open.

Concurrency Properties

The following properties regulate the number of nodes that can run simultaneously and the number of concurrent consumers.

sas.analytics.flows.maximumConcurrentNodeExecution
: The maximum number of nodes that are allowed to run simultaneously. Changing this value requires a restart of the analyticsFlows service. The default value is 5.

  This value is applied per execution provider and per tenant. Therefore, if there are 3 tenants and this property is set to 5, no more than 15 nodes are allowed to execute concurrently per execution provider.

sas.analytics.gateway.job.segment.execution.concurrentConsumers
: The initial number of consumers that are used in each instance of the analyticsGateway service. This is a global setting, not per tenant or per execution provider.

sas.analytics.gateway.job.segment.execution.maxConcurrentConsumers
: The maximum number of consumers that are used in each instance of the analyticsGateway service. This is a global setting, not per tenant or per execution provider.

Disabling Automatic Data Duplication

Model Studio copies the data source when the first **Data** node is run. This can cause performance issues and can cause you to run out of disk space. The amount of space that is required depends on the number of saved projects and on the size of the data source. To prevent Model Studio from automatically creating copies of your data, ensure that the following conditions are met:

1. A Key variable exists in your data. This can be either a variable named _INDEX_ or a variable that is assigned the role Key.
2. A Partition variable exists in your data. This can be either a variable named _PARTIND_ or a variable that is assigned the role Partition.
3. The data must be persistent on the disk.