## Contents

**Chapter 1 / Predictive Modeling**  
Overview .......................................................... 1  
Overview of Data Mining Preprocessing ...................... 1  
Preprocessing by Modifying Data .............................. 3  
Preprocessing by Dimension Reduction ...................... 3  
Preprocessing by Unsupervised Learning .................... 4  
Overview of Supervised Learning ............................ 4  
Overfitting ......................................................... 5  
Selecting Your Algorithm ....................................... 6  
Assessment Tools .................................................. 8  
Handling Rare Events ........................................... 9  
Scoring Your Models ............................................. 9  
Running Your Score Code from ASTORE Models .......... 11  

**Chapter 2 / Autotuning Machine Learning Models**  
Overview of Autotuning .......................................... 13  
Parameters and Hyperparameters .............................. 16  
Search Options .................................................... 16  
Validation Options ............................................... 18  
References .......................................................... 18
Overview

This document is an overview of the different types of predictive modeling algorithms and the preprocessing tools that accompany them. The intended audience for this documentation 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.

In this document, you will find an overview of 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.

Overview of Data Mining Preprocessing

Effective machine learning models are built on a foundation of well-prepared data. Before cleaning and transforming the data, you must think about how the data will be used. You must 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:

- Modify your data and data roles
- 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 will be covered in the Selecting Your Algorithm on page 6 section.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Common Challenges</th>
<th>Suggested Best Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Collection</td>
<td>Biased Data</td>
<td>Take time to understand the business problem and its context</td>
</tr>
<tr>
<td></td>
<td>Incomplete data</td>
<td>Enrich the data</td>
</tr>
<tr>
<td></td>
<td>High-dimensional data</td>
<td>Dimension reduction (Feature Extraction, Variable Clustering, and Variable Selection nodes)</td>
</tr>
<tr>
<td></td>
<td>Sparsity</td>
<td>Change representation of data (Transformations node)</td>
</tr>
<tr>
<td>&quot;Untidy&quot; Data</td>
<td>Value ranges as columns</td>
<td>Transform the data with SAS code (Code node)</td>
</tr>
<tr>
<td></td>
<td>Multiple variables in the same column</td>
<td>Discretization (Transformations node)</td>
</tr>
<tr>
<td></td>
<td>Variables in both rows and columns</td>
<td>Winsorizing (Imputation node)</td>
</tr>
<tr>
<td>Outliers</td>
<td>Out-of-range numeric values and unknown categorical values in score data</td>
<td>Discretization (Transformations node)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Winsorizing (Imputation node)</td>
</tr>
<tr>
<td>Sparse target variables</td>
<td>Low primary event occurrence rate</td>
<td>Proportional oversampling</td>
</tr>
<tr>
<td></td>
<td>Overwhelming preponderance of zero or missing values in target</td>
<td></td>
</tr>
<tr>
<td>Variables of disparate magnitudes</td>
<td>Misleading variable importance</td>
<td>Standardization (Transformations node)</td>
</tr>
<tr>
<td></td>
<td>Distance measure imbalance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gradient dominance</td>
<td></td>
</tr>
<tr>
<td>High-cardinality variables</td>
<td>Overfitting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unknown categorical values in holdout data</td>
<td></td>
</tr>
<tr>
<td>Missing Data</td>
<td>Information loss</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bias</td>
<td></td>
</tr>
</tbody>
</table>
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 about dimension reduction techniques, see Preprocessing by Dimension Reduction on page 3.

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 of 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. Since there can be disagreements on selected variables when different techniques are used, this functionality enables you to select variables that 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 or a redundant parameter (dimension) that can be reduced. For more information about dimension reduction, see Preprocessing by Dimension Reduction on page 3.

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 are 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 predicting interval targets, the problems become a regression problem.
The following table groups the Supervised Learning nodes in Model Studio by possible target type. Nodes listed in both columns support both target types.

<table>
<thead>
<tr>
<th>Interval Target</th>
<th>Nominal and Binary Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>Logistic Regression</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>Decision Tree</td>
</tr>
<tr>
<td>Forest</td>
<td>Forest</td>
</tr>
<tr>
<td>Gradient Boosting</td>
<td>Gradient Boosting</td>
</tr>
<tr>
<td>Neural Network</td>
<td>Neural Network</td>
</tr>
<tr>
<td>GLM (Generalized Linear Model)</td>
<td>SVM (only for binary target)</td>
</tr>
<tr>
<td></td>
<td>Bayesian Network</td>
</tr>
</tbody>
</table>

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/or holdout data:

**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 can obtain 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.

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Target Type</th>
<th>Suggested Usage</th>
<th>Suggested Scale</th>
<th>Interpretability</th>
<th>Automatic Hyperparameter Tuning Capability</th>
<th>Common Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression (Linear, Logistic, GLM)</td>
<td>Linear regression and GLM for interval target</td>
<td>Modeling linear or linearly separable phenomena</td>
<td>Small to large data sets</td>
<td>High</td>
<td>No</td>
<td>Missing values</td>
</tr>
<tr>
<td></td>
<td>Logistic regression for nominal and binary target</td>
<td>Manually specifying nonlinear and explicit interaction</td>
<td></td>
<td></td>
<td></td>
<td>Outliers</td>
</tr>
<tr>
<td></td>
<td>LASSO regression includes a regularization term for linear and logistic regression to deal with multicollinearity and overfitting issues</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Standardization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Parameter tuning</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Type</td>
<td>Modeling</td>
<td>Data Size</td>
<td>Complexity</td>
<td>Missing</td>
<td>Instability</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SVM</td>
<td>Binary</td>
<td>Modeling linear or linearly separable phenomena by using linear kernels or polynomial kernels up to degree three</td>
<td>Small to large data sets</td>
<td>Low</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Tree-based Modeling (Decision Tree, Forest, Gradient Boosting)</td>
<td>Interval, Binary, Nominal</td>
<td>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</td>
<td>Medium to large data sets</td>
<td>Moderate</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
### Neural Network
- Interval: Nonlinear
- Binary: Nonlinearly separable phenomena
- Nominal: 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: Linearly separable phenomena in large data sets
- Nominal: 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 8.2: Procedures Guide.

### Assessment Plots for Interval Target
- Actual and Predicted by Depth
- Predicted by Actual

### Assessment Plots for Nominal and Binary Target
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, complete the following steps:

1. When creating a project, in the New Project window, select 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 of Model Studio, see Running Your Score Code From ASTORE Models on page 11.

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

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Type of Score Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomaly Detection</td>
<td>Analytic store</td>
</tr>
<tr>
<td>Bayesian Network</td>
<td>Analytic store</td>
</tr>
<tr>
<td>Clustering</td>
<td>DATA step</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>DATA step</td>
</tr>
<tr>
<td>Ensemble</td>
<td>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</td>
</tr>
<tr>
<td>Feature Extraction</td>
<td>DATA step</td>
</tr>
<tr>
<td>Filtering</td>
<td>DATA step</td>
</tr>
<tr>
<td>Forest</td>
<td>Analytic store</td>
</tr>
<tr>
<td>GLM</td>
<td>DATA step</td>
</tr>
<tr>
<td>Gradient Boosting</td>
<td>Analytic store</td>
</tr>
<tr>
<td>Imputation</td>
<td>DATA step</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>DATA step</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>DATA step</td>
</tr>
<tr>
<td>Neural Network</td>
<td>DATA step for networks with less than 6 layers, analytic store for networks with 6 or more layers</td>
</tr>
<tr>
<td>Replacement</td>
<td>DATA step</td>
</tr>
<tr>
<td>SVM</td>
<td>Analytic store</td>
</tr>
<tr>
<td>Text Mining</td>
<td>Analytic store</td>
</tr>
<tr>
<td>Transformations</td>
<td>DATA step</td>
</tr>
</tbody>
</table>
Running Your Score Code from ASTORE 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 code 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;
```

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

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

To determine the name of the ASTORE for the CASDATA= parameter, open dmcas_epscorecode.sas. In the top comments section, you should see a name ending with _ast. This is the ASTORE filename. Because Linux is case sensitive, you will need to 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.exmpledataout;
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.
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 will choose 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, a data scientist decides to build a machine learning model to predict which customer is a good credit risk. They must make many decisions during the training process. They need to choose the following:

- which modeling approaches to test
- which data they 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 will impact the outcome of the model building exercise, and eventually the final model selected. Since this model will be used to decide which customers will 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 pre-defined 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. With modern computers, 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.

SAS LSO 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, and an experimental Bayesian search method.

In SAS Visual Data Mining and Machine Learning, autotuning is used by the following nodes:
- Decision Tree
- Forest
- Gradient Boosting
- Neural Network
## SVM

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

<table>
<thead>
<tr>
<th>Forest</th>
<th>Parameter</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maximum Depth</td>
<td>1</td>
<td>150</td>
<td>Integers</td>
</tr>
<tr>
<td></td>
<td>Number of Trees</td>
<td>1</td>
<td>1000</td>
<td>Integers</td>
</tr>
<tr>
<td></td>
<td>In-bag Sample Proportion</td>
<td>0 (not inclusive)</td>
<td>1</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>Number of Inputs per Split</td>
<td>1</td>
<td>Number of inputs.</td>
<td>Integers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gradient Boosting</th>
<th>Parameter</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L1 Regularization</td>
<td>0</td>
<td>not applicable</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>L2 Regularization</td>
<td>0</td>
<td>not applicable</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>Learning Rate</td>
<td>0 (not inclusive)</td>
<td>1</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>Number of Inputs per Split</td>
<td>1</td>
<td>Number of inputs.</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>Number of Iterations</td>
<td>1</td>
<td>10000</td>
<td>Integers</td>
</tr>
<tr>
<td></td>
<td>Subsample Rate</td>
<td>0 (not inclusive)</td>
<td>1</td>
<td>Real values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Neural Network</th>
<th>Parameter</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of Hidden Layers</td>
<td>0</td>
<td>5</td>
<td>Integers</td>
</tr>
<tr>
<td></td>
<td>Number of Neurons in Each Hidden Layer</td>
<td>1</td>
<td>1000</td>
<td>Integers</td>
</tr>
<tr>
<td></td>
<td>L1 Weight Decay</td>
<td>0</td>
<td>not applicable</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>L2 Weight Decay</td>
<td>0</td>
<td>not applicable</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>Learning Rate</td>
<td>0 (not inclusive)</td>
<td>not applicable</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>Annealing Rate</td>
<td>0 (not inclusive)</td>
<td>not applicable</td>
<td>Real values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SVM</th>
<th>Parameter</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Penalty</td>
<td>.000001</td>
<td>100000</td>
<td>Real values</td>
</tr>
<tr>
<td></td>
<td>Polynomial Degree</td>
<td>1</td>
<td>3</td>
<td>Integers</td>
</tr>
</tbody>
</table>

<p>| Decision Tree       |                                |               |               |           |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Depth</td>
<td>1</td>
<td>19</td>
<td>Integers</td>
</tr>
<tr>
<td>Interval Input Bins</td>
<td>20</td>
<td>200</td>
<td>Integers</td>
</tr>
<tr>
<td>Grow Criterion</td>
<td>not applicable</td>
<td>not applicable</td>
<td>Grow criteria are evaluated from a specified list.</td>
</tr>
</tbody>
</table>

**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.

![Modeling Algorithm](image)

**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.

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.
**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**
