

Paper SAS4283-2020

Introducing the GAMSELECT Procedure for Generalized Additive Model Selection

Michael Lamm and Weijie Cai, SAS Institute Inc., Cary, NC

ABSTRACT

Model selection is an important area in statistical learning. Both SAS/STAT® software and SAS® Visual Statistics software provide a rich set of tools for performing model selection over linear models, generalized linear models, and Cox proportional hazards models. With these tools, you can build parsimonious predictive models by constructing linear or fixed nonlinear effects to describe the dependency structure. But what if the dependency structure is nonlinear and the nonlinearity is unknown? And what if covariates are nonlinearly correlated? The new GAMSELECT procedure, available in SAS Visual Statistics, addresses these questions by using spline terms to approximate the nonlinear dependency, then selecting important variables in appropriate nonlinear transformations by using the boosting method or the shrinkage method. The procedure builds models for response variables in the exponential family, so that you can use it for continuous, count, or binary responses. This paper introduces the GAMSELECT procedure and provides a brief comparison to related SAS® procedures.

INTRODUCTION

The GAMSELECT procedure, which is available in SAS Visual Statistics 8.5 in SAS® Viya®, fits and performs model selection on generalized additive models (GAMs). Generalized additive models are a popular class of models that you can use to build flexible and interpretable predictive models. By using spline terms to estimate functions of the input variables, you can use GAMs to model possibly unknown nonlinear dependencies and use visualizations of univariate and bivariate spline terms to understand the contribution that an individual spline term makes to the model fit.

PROC GAMSELECT supports model selection on GAMs by componentwise functional gradient descent, or boosting, and by the shrinkage method by using penalized likelihood optimization with sparsity-inducing penalties. These model selection methods enable you to build parsimonious predictive models when there might be unknown nonlinear dependence structures or correlated predictors. Moreover, because PROC GAMSELECT is designed specifically to fit and select GAMs, the model selection methods that it implements also support options you can use to control the smoothness of the model fit. This paper provides examples that demonstrate how PROC GAMSELECT enables you to control both the complexity and smoothness of the model fit and how the selection methods supported by PROC GAMSELECT compare to alternative modeling approaches supported by related procedures in SAS software.

The paper is organized as follows. First, the section “[Generalized Additive Models and Model Selection Methods](#)” briefly reviews generalized additive models and provides an overview of the model selection methods that PROC GAMSELECT supports. [Example 1](#) is an extensive example that uses a simulated data set to demonstrate the model selection methods supported by PROC GAMSELECT and how these methods compare to alternative approaches. [Example 2](#) analyzes the home equity data set **Hmeq** to illustrate the analysis of binary response data and the use of data partitioning to assess a model fit.

Generalized Additive Models and Model Selection Methods

Generalized additive models are an extension of generalized linear models and can be used to model distributions in the exponential family. Let (y_i, \mathbf{x}_i) , $i = 1, \dots, n$, be a set of observations where y_i is independently distributed in some exponential family. Generalized additive models and generalized linear models both assume an additive model

$$g(\mu_i) = f_1(x_{i1}) + f_2(x_{i2}) + \dots + f_p(x_{ip})$$

where $\mu_i = E(y_i)$ and $g(\cdot)$ is a link function. Generalized linear models further assume that each component function f_j , $j = 1, \dots, p$, is a linear function of x_{ij} . Generalized additive models relax the linearity assumption and allow for nonlinear smoothing functions f_j .

Maximizing a likelihood function for the data is the basis of many algorithms for fitting generalized additive models and generalized linear models. The GENSELECT procedure in SAS Visual Statistics fits generalized linear models by using maximum likelihood estimation. It supports model selection by using traditional selection methods, such as forward and backward selection, or by using the group LASSO, a penalized likelihood method. The selection methods that PROC GENSELECT supports are not designed specifically for use on generalized additive models.

The GAMMOD procedure in SAS Visual Statistics fits generalized additive models by using penalized likelihood estimation. It uses thin-plate regression splines to construct spline terms, and the penalty that is applied to the likelihood is used to control the roughness of the model fit. The roughness penalty aims to control the complexity between linear and highly nonlinear fits for individual spline terms and does not induce sparsity. Although PROC GAMMOD does not support model selection, it does support model inference.

The GAMSELECT procedure supports methods of fitting and selecting generalized additive models that can control both the roughness and sparsity of the model fit. The models that it fits can consist of smoothing functions, which are estimated by using spline terms, and parametric effects. PROC GAMSELECT supports two methods of fitting and selecting generalized additive models. The boosting method fits and selects a model by using a componentwise implementation of a functional gradient descent algorithm. The shrinkage method uses a penalized likelihood approach with sparsity-inducing penalties. The procedure does not support inference on the selected model. The following sections provide an overview of the boosting and shrinkage model selection methods that PROC GAMSELECT supports.

Boosting

The boosting selection method implements a componentwise version of a functional gradient descent, or boosting, algorithm (Friedman 2001; Bühlmann and Hothorn 2007). The component functions in the algorithm correspond to the parametric effects and spline terms that are specified in the model. The model can consist of only parametric effects, only spline terms, or a mix of parametric effects and spline terms. The boosting method supports univariate and bivariate spline terms.

Table 1 defines some notation that is used here to describe the componentwise boosting algorithm that PROC GAMSELECT implements.

Table 1 Boosting Notation

| Notation | Meaning |
|------------------|---|
| ℓ | Negative log likelihood |
| \hat{f}^m | Estimate for f at m th boosting iteration |
| \hat{f}_j^m | Estimate for j th component function f_j at m th boosting iteration |
| \mathbf{x}_j | Predictors for j th component function f_j |
| γ_j^m | Fit for j th component at m th boosting iteration |
| ν | Step size for boosting algorithm |
| m_{end} | Number of boosting iterations executed |
| M | Maximum number of boosting iterations |
| w_i | Observation weight for i th observation |

The componentwise boosting algorithm is as follows:

1. Initialize \hat{f}^0 and $m = 1$.
2. Repeat the following steps until the maximum number of iterations M has been reached or an early stopping criterion is met:
 - a) Compute $u_i = -\frac{\partial}{\partial f} \ell(y_i, f)|_{\hat{f}^{m-1}(\mathbf{x}_i)}$, $i = 1, \dots, n$.
 - b) For each component function f_j , $j = 1, \dots, p$, obtain an estimated fit $\hat{\gamma}_j^m$ for the data (u_i, \mathbf{x}_{ij}) , $i = 1, \dots, n$.
 - c) Select the component $j^* = \operatorname{argmin}_{j=1, \dots, p} \sum_{i=1}^n w_i (u_i - \hat{\gamma}_j^m(\mathbf{x}_{ij}))^2$.
 - d) Update the component functions by

$$\hat{f}_j^m = \begin{cases} \hat{f}_j^{m-1} + v\gamma_j^m & \text{if } j = j^* \\ \hat{f}_j^{m-1} & \text{if } j \neq j^* \end{cases}$$
 - e) Set $m = m + 1$ and check the stopping criterion.
3. Choose an iteration $m^* \leq m_{\text{end}}$ and set $\hat{f} = \hat{f}^{m^*}$.

By default, the initial function \hat{f}^0 is an intercept-only model fit. The function can also be set to the value of an offset variable that you specify.

For component functions f_j that correspond to parametric effects, the functions γ_j^m are linear models that are fit by solving weighted ordinary least squares problems. Each linear model $\gamma_j^m = \mathbf{x}_{ij} \boldsymbol{\beta}_j$ includes an intercept term. For component functions f_j that correspond to spline terms, the functions γ_j^m are estimated using penalized B-splines (De Boor 1978; Eilers and Marx 1996; Marx and Eilers 2005). By default, a smoothing parameter value is selected so that a univariate spline term has 4 degrees of freedom and a bivariate spline term has 6 degrees of freedom. The small degrees of freedom for spline terms and the small step size used in the update step, which is 0.1 by default, support a fine exploration of the model space. For more information about the construction of spline terms when you are using the boosting selection method, see the documentation of [PROC GAMSELECT](#) (SAS Institute Inc. 2019).

When the boosting iterations finish, there are $m_{\text{end}} + 1$ models \hat{f}^m . By default, the procedure returns the model $\hat{f} = \hat{f}^{m^*}$ that minimizes the average square error (ASE) for the training data. To prevent overfitting of the training data, you can specify an early stopping criterion or change the criterion that is used to choose the final model to either k -fold cross-validation of the ASE or the model ASE for a validation data partition. The effects in the final model are effects that were selected for at least one boosting iteration $m < m^*$. The use of early stopping and cross-validation of the ASE with the boosting selection method is demonstrated in the section [“The Boosting Selection Method in PROC GAMSELECT.”](#)

PROC GAMSELECT Compared with the GRADBOOST Procedure

Both the GAMSELECT and GRADBOOST procedures fit predictive models by using a generic functional gradient descent, or boosting, algorithm (Friedman 2001). However, the procedures differ substantially in how they implement the boosting algorithms and the types of predictive models that they fit.

The GAMSELECT procedure uses a componentwise implementation of the generic functional gradient descent algorithm. The effects in the model can be parametric effects, univariate spline terms, or bivariate spline terms. The parameter estimates for each component are additive across the iterations of the boosting algorithm. As a result, the predictive model that is fit by the boosting method in PROC GAMSELECT is a generalized additive model that consists of the parametric effects and spline terms selected at some boosting iteration.

The boosting algorithm that is implemented by the GRADBOOST procedure fits a decision tree at each iteration. The tree can consist of both interval and nominal predictors. The decision trees that are fit at each iteration are not additive in the estimated parameters. As a result, the predictive model that is fit by PROC GRADBOOST is a tree ensemble. A comparison between models that are fit by the boosting selection method in PROC GAMSELECT and PROC GRADBOOST is provided in the section [“Comparison with PROC GRADBOOST.”](#)

Shrinkage

The shrinkage selection method uses sparsity-inducing norms to select a generalized additive model. Because nonlinear terms in additive models are typically represented by splines and thus are naturally grouped, the L_2 norm that is used in group LASSO (Yuan and Lin 2006) and its generalizations are often used to perform additive model selection. For different approaches, see Lin and Zhang (2006), Cantoni, Flemming, and Ronchetti (2006), Ravikumar et al. (2009), Meier, Van de Geer, and Bühlmann (2009), Marra and Wood (2011), Raskutti, Wainwright, and Yu (2012), Suzuki and Sugiyama (2013), and Chouldechova and Hastie (2015). For a review of this field, see Amato, Antoniadis, and De Feis (2016). The shrinkage method performs selection only for the spline terms in the model, supports only univariate splines, and requires at least one spline term in the model.

The shrinkage selection method uses natural cubic splines to estimate the unknown smoothing functions and uses a B-spline basis representation for each natural cubic spline. Based on ideas in Meier, Van de Geer, and Bühlmann (2009), Suzuki and Sugiyama (2013), and Amato, Antoniadis, and De Feis (2016), the shrinkage method that is used in the GAMSELECT procedure performs a grid search of tuning parameters $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ and evaluates different models on the basis of their fits, given any fixed λ value. For each fixed λ , PROC GAMSELECT fits the model by minimizing the penalized negative log-likelihood function

$$-\frac{\ell(\boldsymbol{\beta})\phi_d}{n} + \lambda_1 \sum_j \phi_j \sqrt{\boldsymbol{\beta}'_j \mathbf{K}_j \boldsymbol{\beta}_j} + \lambda_2 \sum_j \psi_j \sqrt{\boldsymbol{\beta}'_j \boldsymbol{\Omega}_j \boldsymbol{\beta}_j} + \lambda_3 \sum_j \boldsymbol{\beta}'_j \boldsymbol{\Omega}_j \boldsymbol{\beta}_j$$

where \mathbf{B}_j is the B-spline basis expansions of \mathbf{x}_j , $\mathbf{K}_j = \mathbf{B}'_j \mathbf{B}_j / n$, and $\boldsymbol{\Omega}_j = \int (f''_j)^2 dx$. The two terms $\lambda_1 \sum_j \phi_j \sqrt{\boldsymbol{\beta}'_j \mathbf{K}_j \boldsymbol{\beta}_j}$ and $\lambda_2 \sum_j \psi_j \sqrt{\boldsymbol{\beta}'_j \boldsymbol{\Omega}_j \boldsymbol{\beta}_j}$ are sparsity-inducing norms, which are generalizations of the L_2 norm. ϕ_d is the dispersion parameter for the appropriate distribution in the exponential family. ϕ_j and ψ_j are two sets of control parameters that you can use to enforce stronger or weaker penalties for each individual spline term. You can use these terms to form sparsity and smoothness penalties in a data-adaptive way (Meier, Van de Geer, and Bühlmann 2009), which is similar to the adaptive LASSO (Zou 2006). The use of this data-adaptive approach is demonstrated in the section [“The Shrinkage Selection Method in PROC GAMSELECT.”](#)

The first term can force $\boldsymbol{\beta}_j = \mathbf{0}$ if λ_1 is sufficiently large, and the second term can force $\boldsymbol{\beta}_j$ to be equivalent to a parametric solution ($\boldsymbol{\beta}'_j \boldsymbol{\Omega}_j \boldsymbol{\beta}_j = \mathbf{0}$) if λ_2 is sufficiently large. The third term, $\lambda_3 \sum_j \boldsymbol{\beta}'_j \boldsymbol{\Omega}_j \boldsymbol{\beta}_j$, serves as an optional penalty (similar to ridge regression) to make the solution smoother.

Example 1: Comparing Model Selection Methods

The example in this section compares different methods of fitting and selecting a generalized additive model. It demonstrates features of the GAMSELECT procedure that you can use to control the complexity of the model fit and compares the estimation methods that are supported by PROC GAMSELECT to related approaches supported by other procedures.

This example is organized as follows. The section [“Simulating the Example Data”](#) introduces the simulated data that are analyzed throughout the example. Next, the section [“PROC GENSELECT Model Fit”](#) provides an initial analysis of the example data by using PROC GENSELECT. To address the challenges that arise in attempting to analyze these data by using the model selection methods supported by PROC GENSELECT, you can use PROC GAMSELECT and the methods that it supports. The section [“The Boosting Selection Method in PROC GAMSELECT”](#) demonstrates the use of the boosting selection method, and the section [“Comparison with PROC GRADBOOST”](#) compares models that are fit by using the different boosting algorithms in the GAMSELECT and GRADBOOST procedures. The final section in this example, [“The Shrinkage Selection Method in PROC GAMSELECT,”](#) demonstrates the use of the shrinkage selection method.

For readability, this paper does not display all the code that is used to generate the output shown throughout the example. All the programs and macros that are used in this example are available from the SAS[®] Global Forum 2020 conference proceedings and online.

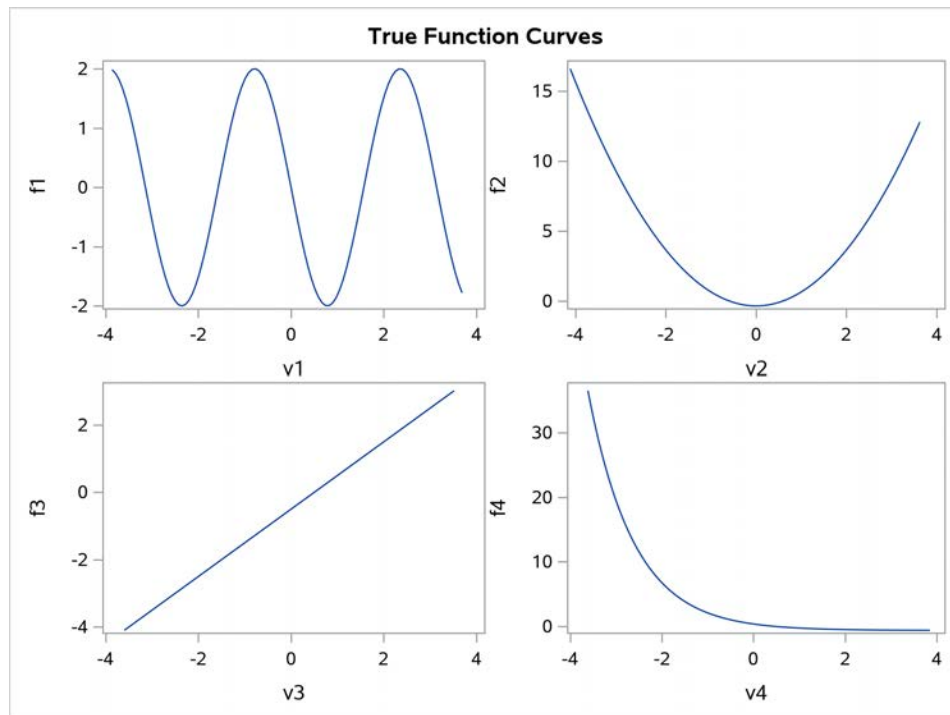
Simulating the Example Data

The GAMSELECT procedure is developed specifically for SAS Viya and requires the input data to be in a SAS® Cloud Analytic Services (CAS) table accessible in your CAS session. The following PROC CAS statements use the `runCode` action in the `dataStep` action set to create the data table `one` in the current CAS engine libref named `mycas`:

```
proc cas;
  dataStep.runCode result=r/
    single='yes' code=
      'data one;
        call streaminit(1);
        array v{200} v1-v200;
        do i=1 to 10000;
          do j=1 to 200;
            v{j}=rand("normal");
          end;
          f1=-2*sin(2*v1);
          f2=v2*v2-1./3;
          f3=v3-0.5;
          f4=exp(-v4)+exp(-1)-1;
          linp=f1+f2+f3+f4;
          y=rand("normal", linp);
        output;
      end;
    run; '
;
run;
quit;
```

The data table `one` is used throughout this example. It consists of 10,000 observations for a continuous response variable (`y`) and 200 continuous variables (`v1-v200`). The model for the mean of response variable `y` includes a sine function of the variable `v1`, a quadratic function of the variable `v2`, an exponential function of the variable `v3`, and a linear function of the variable `v4`. Figure 1 shows the true function curves of the four variables that construct the response.

Figure 1 True Function Curves



Without knowing the data generating process, you could form an initial analysis plan by using some basic exploratory tools. A density histogram of the response variable y (not shown) suggests that it follows a normal distribution, and a scatter matrix (not shown) indicates many possible nuisance predictors and nonlinear relationships between some of the predictors and the response. Because PROC GAMMOD does not support model selection, attempting to fit a model for the response by using PROC GAMMOD with spline terms for all 200 variables, $v1-v200$, would not produce a parsimonious model. The following sections compare different approaches for analyzing these data.

PROC GENSELECT Model Fit

The following statements fit and select a model by using the stepwise selection method in PROC GENSELECT:

```
proc genselect data=mycas.one;
  model y = v1-v200;
  selection method = stepwise;
run;
```

This approach ignores the nonlinear relationship between the predictors and the response, and the selected model results in a poor fit. The average square error for the selected model is about 6.5. The selected model includes only three of the four true effects, one noise effect, and the intercept (Figure 2).

Figure 2 Selected Effects for a Linear Model

The GENSELECT Procedure

Selection Details

| Selection Summary | | | |
|------------------------------|----------------|-------------------|-------------|
| Step | Effect Entered | Number Effects In | SBC |
| 0 | Intercept | 1 | 51875.4386 |
| 1 | v4 | 2 | 49266.8214 |
| 2 | v3 | 3 | 47717.0574 |
| 3 | v1 | 4 | 47228.5801 |
| 4 | v52 | 5 | 47218.3537* |
| * Optimal Value Of Criterion | | | |

To model nonlinear terms in PROC GENSELECT, you can use the EFFECT statement to construct regression splines. The following statements repeat the model selection process with regression splines constructed for all 200 input variables, $v1-v200$:

```
proc genselect data=mycas.one;
  effect spl=spline(v1-v200/separate);
  model y = spl;
  selection method = stepwise;
run;
```

With the regression splines, the model fit is much improved, with an average square error of about 1.4. The output from PROC GENSELECT (Figure 3) shows that the five effects in the selected model are the intercept and the four regression splines constructed from the variables $v1-v4$.

Figure 3 Selected Effects with Regression Splines

The GENSELECT Procedure

Selection Details

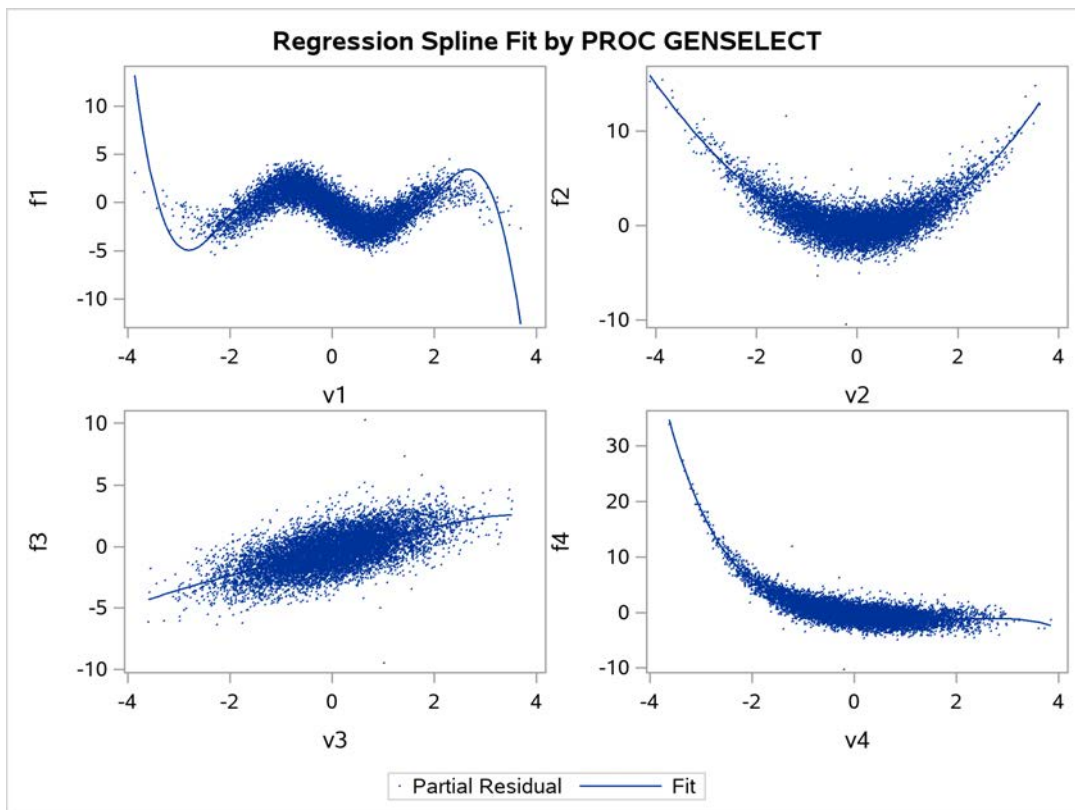
| Selection Summary | | | | |
|-------------------|----------------|----------------|-------------------|-------------|
| Step | Effect Entered | Effect Removed | Number Effects In | SBC |
| 0 | Intercept | | 1 | 51875.4386 |
| 1 | spl_v4 | | 2 | 47699.2599 |
| 2 | spl_v2 | | 3 | 43126.1957 |
| 3 | spl_v1 | | 4 | 38349.1835 |
| 4 | spl_v3 | | 5 | 33134.1582 |
| 5 | spl_v159 | | 6 | 32003.0750 |
| 6 | | spl_v159 | 5 | 31961.3017* |

* Optimal Value Of Criterion

For regression modeling of data that have nonlinear structures, it is often necessary to check partial residuals and partial predictions to determine whether the included model effects sufficiently explain the nonlinear structure. Figure 4 displays the partial predictions overlaid with fitted partial prediction curves for the selected model.

In Figure 4, you observe that the fitted curves explain the pattern in the partial residuals reasonably well, except for the first variable, v1. The fitted curve seems to overfit the partial residuals, especially in the tail areas. This overfitting suggests that the default regression spline construction is not sufficient for modeling the nonlinear relationship between v1 and y.

Figure 4 PROC GENSELECT Fit



This problem has a few possible solutions. One solution is to construct multiple spline bases at different scales of knot intervals in order to approximate the nonlinear structure at finer scales and plot the partial residuals for the new models. In general, you have to keep repeating these steps in order to obtain a model that explains the nonlinear structure well. For data that have complicated dependency structures, this might take many iterations, and the model fitting algorithms that PROC GENSELECT supports are not tailored to this problem.

These challenges are addressed by PROC GAMSELECT. The methods that it supports to fit and select models use penalized spline terms that are less prone to exhibit extreme boundary behaviors and use algorithms tailored to this application.

The Boosting Selection Method in PROC GAMSELECT

This section continues the analysis of building an additive model for the data table **one** that is created at the beginning of this example and demonstrates how you can use the boosting selection method in PROC GAMSELECT.

The boosting selection method supports models that consist of both parametric effects and spline terms. In the MODEL statement, you can use the PARAM(*effects*) option to specify parametric effects that are constructed from the input data. You can specify multiple effects in one PARAM option and use this option multiple times.

In the MODEL statement, you use the SPLINE(*variable*) option to specify a nonparametric spline term that is constructed from the *variable* in the input data. This syntax for specifying a spline term is consistent with the syntax that you use to specify spline terms in PROC GAMMOD in SAS Visual Statistics and PROC GAMPL in SAS/STAT. Unlike the PARAM(*effects*) option, which can be used to specify multiple effects, each SPLINE(*variable*) option constructs only one spline term from the specified *variable*. You can specify multiple SPLINE(*variable*) options. The boosting method supports univariate spline terms that are constructed from a single continuous input variable or bivariate spline terms that are constructed using the SPLINE(*variable1 variable2*) option. The spline terms are estimated by using penalized B-splines. In the SPLINE(*variable*) option, you can specify further spline-options to control the spline construction. For more information about options to control the construction of spline terms, see the documentation of PROC GAMSELECT (SAS Institute Inc. 2019).

The following PROC GAMSELECT statements use the boosting selection method to fit a model for the variable **y** without specifying any options to control the model fit. To simplify the model specification, the macro function **SplinePrefixList** is used to specify spline terms for the variables **v1–v200** with the default construction:

```
%macro SplinePrefixList(prefix,n);
  %do i = 1 %to &n;
    spline(&prefix.&i)
  %end;
%mend;

proc gamselect data=mycas.one plots=all;
  model y = %SplinePrefixList(v,200);
  selection method=boosting;
run;
```

The “Iteration Summary” table (Figure 5) shows that the boosting algorithm executed 500 iterations and selected the model that corresponds to the final boosting iteration.

Figure 5 Iteration Summary for Default Model
The GAMSELECT Procedure

| Iteration Summary | |
|----------------------------|-----|
| Iterations Executed | 500 |
| Selected Iteration | 500 |
| Number of Selected Effects | 8 |

The selected model includes eight effects, not counting the intercept. The “Selected Effects” table (Figure 6) shows the effects in the selected model, the iteration at which an effect first entered the model, and the number of times (that is, the number of iterations for which) the effect was selected. The four true effects that were constructed from

the variables **v1–v4** were selected for update at most of the boosting iterations. The four noise effects in the model entered at iterations later in the boosting algorithm and were selected for a small number of iterations.

Figure 6 Selected Effects for Default Model

| Selected Effects | | |
|------------------|-----------------|----------------|
| Effect | Entry Iteration | Times Selected |
| Spline(v4) | 1 | 187 |
| Spline(v2) | 5 | 67 |
| Spline(v1) | 9 | 204 |
| Spline(v3) | 18 | 35 |
| Spline(v15) | 443 | 3 |
| Spline(v51) | 464 | 2 |
| Spline(v148) | 472 | 1 |
| Spline(v98) | 493 | 1 |

Figure 7 shows the fit statistics for the selected model; in this case, there is only one—the ASE.

Figure 7 Fit Statistics for Default Model

| Fit Statistics | |
|----------------------|---------|
| Average Square Error | 1.03910 |

The smoothing component plots for the selected effects in Figure 8 show well-reconstructed curves for the four spline terms that are constructed from the variables **v1–v4**. The curves that are constructed from the noise terms show small contributions.

Figure 8 Smoothing Component Panel for Default Model

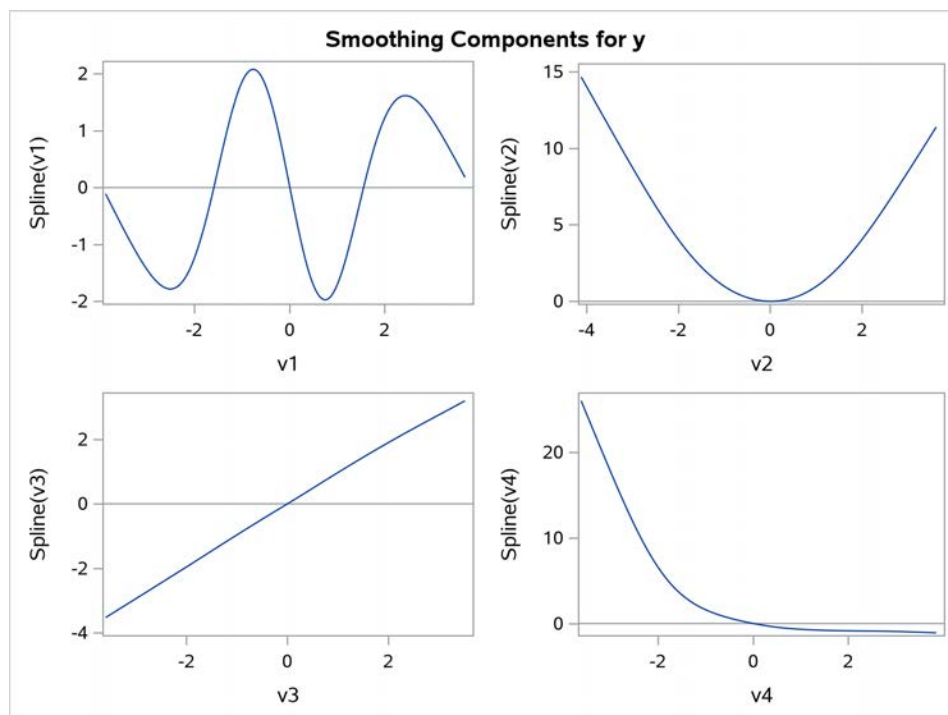
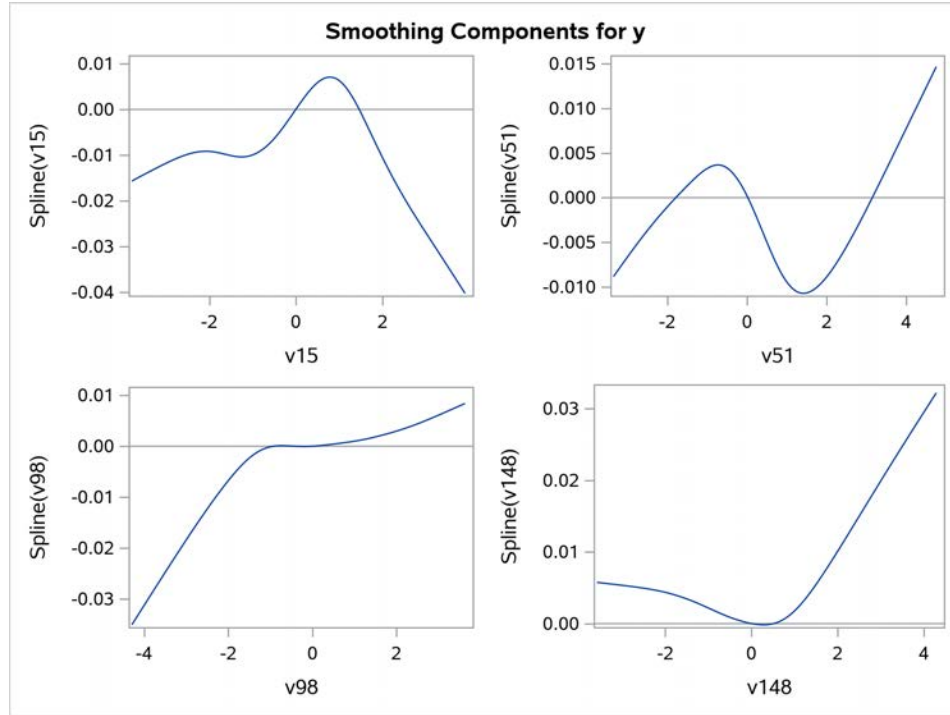


Figure 8 continued



By default, PROC GAMSELECT selects the model \hat{f}^{m^*} that minimizes the ASE for the training data from the set of models that correspond to each boosting iteration. This criterion for selecting the final model might result in overfitting of the training data. To limit overfitting of the training data with the boosting selection method, you can use an alternative criterion to select the final model, use an early stopping criterion, or combine the two approaches. You can change the selection criterion either to the ASE for a validation data partition or to the k -fold cross-validation of the ASE. For early stopping of the selection process, you can specify a number of consecutive iterations at which the performance as measured by the selection criterion must deteriorate on an absolute or relative scale.

To demonstrate the use of early stopping, the model selection process is repeated using k -fold cross-validation of the ASE as the selection criterion. PROC GAMSELECT can perform cross-validation either by randomly assigning observations to folds or by using an input variable to partition the data into folds. Note that the random sampling of observations for cross-validation depends on the initial random number seed and how the data are distributed across machines and threads. Therefore, even if you specify the same initial seed value every time, results might differ between runs of PROC GAMSELECT. The following DATA step adds a variable **cvFold** that can be used to partition the data into folds for cross-validation:

```
data mycas.one / single=yes;
  set mycas.one;
  call streaminit(1848);
  cvFold = rand("table",0.2,0.2,0.2,0.2,0.2);
run;
```

The following PROC GAMSELECT statements perform model selection by using k -fold cross-validation to select the final model with early stopping on the basis of a relative change in the cross-validation ASE for the same list of spline terms specified in the **SplinePrefixList** macro function:

```
proc gamselect data=mycas.one plots=all;
  model y = %SplinePrefixList(v,200);
  selection method=boosting(choose=CV index=cvFold
    stopHorizon=10 stopTol = 0.0005);
run;
```

Figure 9 shows that with cross-validation of the ASE as the selection criterion and early stopping added to the selection process, the process is terminated early, at the 300th boosting iteration, and the selected model includes only the four true effects.

Figure 9 Iteration Summary and Selected Effects Using Cross-Validation

The GAMSELECT Procedure

| Iteration Summary | |
|----------------------------|-----|
| Iterations Executed | 300 |
| Selected Iteration | 300 |
| Number of Selected Effects | 4 |

| Selected Effects | | |
|------------------|-----------------|----------------|
| Effect | Entry Iteration | Times Selected |
| Spline(v4) | 1 | 94 |
| Spline(v2) | 5 | 45 |
| Spline(v1) | 9 | 134 |
| Spline(v3) | 18 | 28 |

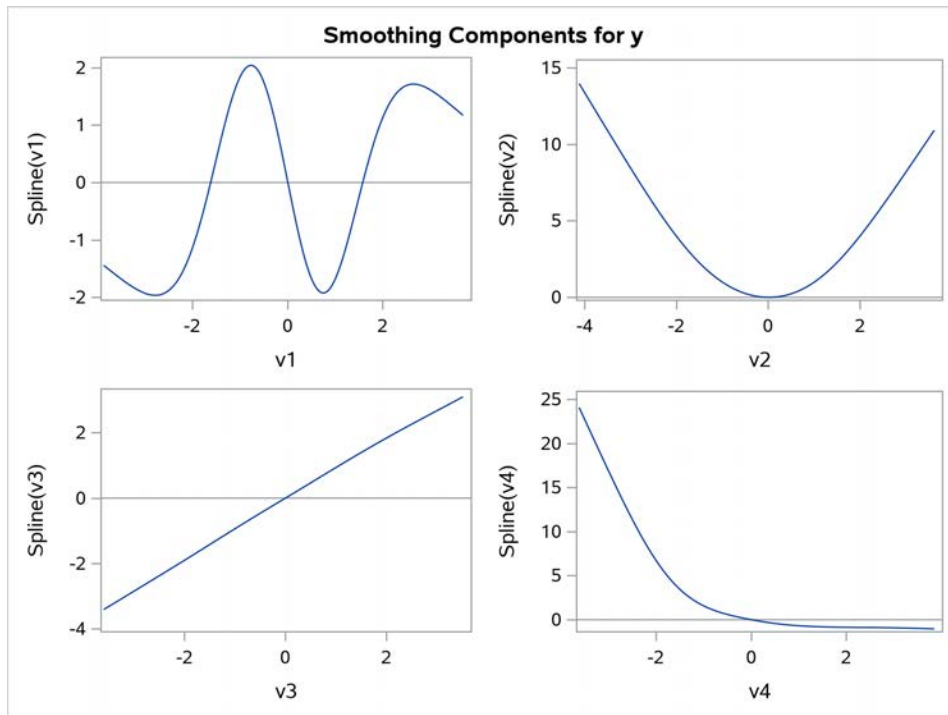
The “Fit Statistics” table in [Figure 10](#) shows the ASE for the training data and the k -fold cross-validation of the ASE. The two ASE values are similar and are slightly larger than the ASE value for the default model ([Figure 7](#)).

Figure 10 Fit Statistics Using Cross-Validation

| Fit Statistics | |
|----------------------|---------|
| Average Square Error | 1.07588 |
| Cross Validation ASE | 1.09528 |

If you compare the component plots for the model selected using cross-validation and early stopping ([Figure 11](#)) to plots for the model selected using the default settings ([Figure 8](#)), you see that the early stopping model shows a poorer reconstruction in the tails of **Spline(v1)**, with a fit that appears more linear in the tail regions.

Figure 11 Smoothing Component Panel Using Cross-Validation



This difference suggests that the fit might benefit from increasing the degrees of freedom for the penalized B-spline fit at each boosting iteration. Increasing the degrees of freedom for the penalized B-spline for **Spline(v1)** fit at each iteration will lead to a decrease in the smoothing parameter. The following PROC GAMSELECT statements fit a reduced model by using spline terms for the variables **v1–v4** with 10 degrees of freedom for the **Spline(v1)** estimate fit at each boosting iteration, and the model is selected by using cross-validation and early stopping:

```
proc gamselect data=mycas.one plots=all;
  model y = spline(v1 / df = 10) spline(v2) spline(v3) spline(v4);
  selection method=boosting(choose=CV index=cvFold
    stopHorizon=10 stopTol = 0.0005);
run;
```

Figure 12 shows that the selection process terminates at the 219th boosting iteration and selects a model with an ASE closer to the ASE of the default model.

Figure 12 Iteration Summary and Fit Statistics for Reduced Model

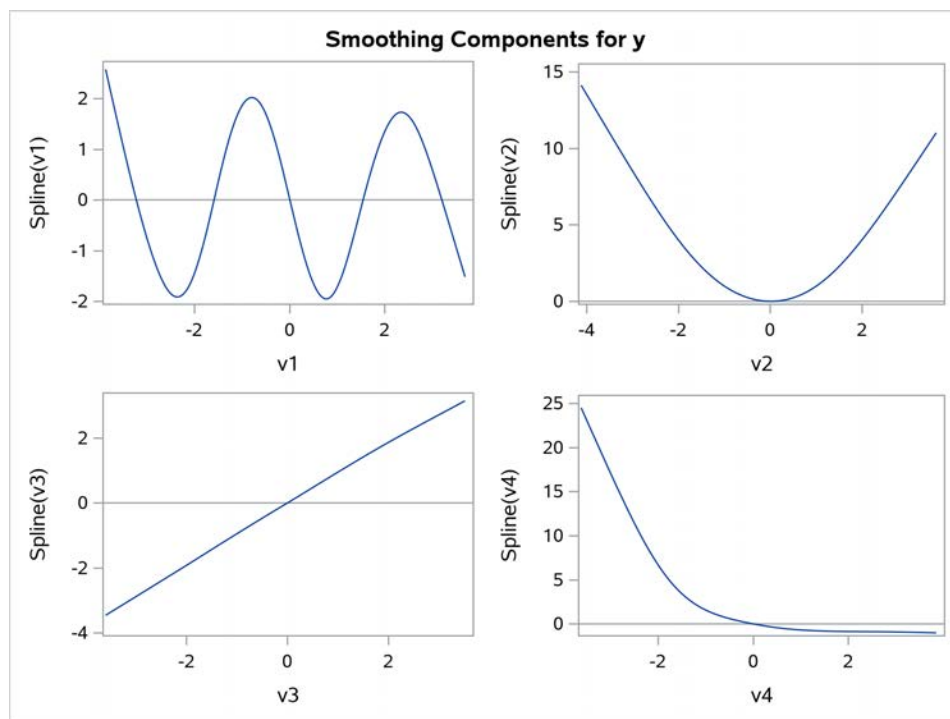
The GAMSELECT Procedure

| Iteration Summary | |
|-----------------------------------|-----|
| Iterations Executed | 219 |
| Selected Iteration | 219 |
| Number of Selected Effects | 4 |

| Fit Statistics | |
|-----------------------------|---------|
| Average Square Error | 1.04808 |
| Cross Validation ASE | 1.06386 |

The smoothing component plots for the model (Figure 13) show well-reconstructed curves for all terms.

Figure 13 Smoothing Component Panel for Reduced Model



Comparison with PROC GRADBOOST

This section compares the model fit by using the boosting selection method in PROC GAMSELECT that is discussed in the section “[The Boosting Selection Method in PROC GAMSELECT](#)” to a model fit by using PROC GRADBOOST. The GRADBOOST procedure in SAS[®] Visual Data Mining and Machine Learning software also fits models by using a generic functional gradient descent, or boosting, algorithm. However, the types of models that are fit by PROC GAMSELECT and PROC GRADBOOST differ substantially. The boosting selection method in PROC GAMSELECT is used to fit and select generalized additive models, whereas PROC GRADBOOST implements a boosting algorithm to build a tree ensemble.

The following statements use the GRADBOOST procedure to fit a tree ensemble by using the default settings and use cross-validation to evaluate the model fit:

```
proc gradboost data=mycas.one seed=1234;  
  target y;  
  input v1-v200;  
  savestate rstore=mycas.gbTreeMod;  
  crossvalidation;  
run;
```

Figure 14 displays the “Fit Statistics” table for the model. The number of trees in the model corresponds to the number of boosting iterations.

Figure 14 Fit Statistics for Default PROC GRADBOOST Model

| Fit Statistics | |
|-----------------|-------------------------------|
| Number of Trees | Training Average Square Error |
| 1 | 9.38 |
| 2 | 8.44 |
| 3 | 7.66 |
| 4 | 6.98 |
| 5 | 6.44 |
| 6 | 5.94 |
| . | . |
| . | . |
| . | . |
| . | . |
| 94 | 1.15 |
| 95 | 1.14 |
| 96 | 1.14 |
| 97 | 1.13 |
| 98 | 1.13 |
| 99 | 1.13 |
| 100 | 1.12 |

Figure 15 displays the “Cross-Validation Fit Statistics” table for the model. The ASE for the training data of 1.12 (Figure 14) is lower than the cross-validation of the ASE of 1.83 (Figure 15). Note that the CROSSVALIDATION statement in PROC GRADBOOST does not support the partitioning of the data into folds on the basis of an input variable, and the folds that PROC GRADBOOST uses do not correspond to the folds that PROC GAMSELECT uses in the section “[The Boosting Selection Method in PROC GAMSELECT](#).”

Figure 15 Cross-Validation Fit Statistics for Default PROC GRADBOOST Model

The GRADBOOST Procedure

| Cross-Validation Fit Statistics | | | | | | | | |
|---------------------------------|------------------------|--------------------|----------|--------------|----------------|-----------|---------------------------|-----------|
| Fold | Number of Observations | Squared Error | | | Absolute Error | | Squared Logarithmic Error | |
| | | Divisor of Average | Average | Root Average | Mean | Root Mean | Mean | Root Mean |
| Fold 1 | 2000 | 2000 | 1.914462 | 1.383641 | 0.978425 | 0.989154 | 0.479221 | 0.692258 |
| Fold 2 | 2000 | 2000 | 2.096067 | 1.447780 | 0.951607 | 0.975504 | 0.663859 | 0.814776 |
| Fold 3 | 2000 | 2000 | 1.761975 | 1.327394 | 0.963282 | 0.981469 | 0.507466 | 0.712366 |
| Fold 4 | 2000 | 2000 | 1.833504 | 1.354070 | 0.930407 | 0.964576 | 0.600733 | 0.775069 |
| Fold 5 | 2000 | 2000 | 1.529184 | 1.236602 | 0.935761 | 0.967347 | 0.421566 | 0.649281 |
| Average | 2000 | 2000 | 1.827039 | 1.349897 | 0.951896 | 0.975610 | 0.534569 | 0.728750 |

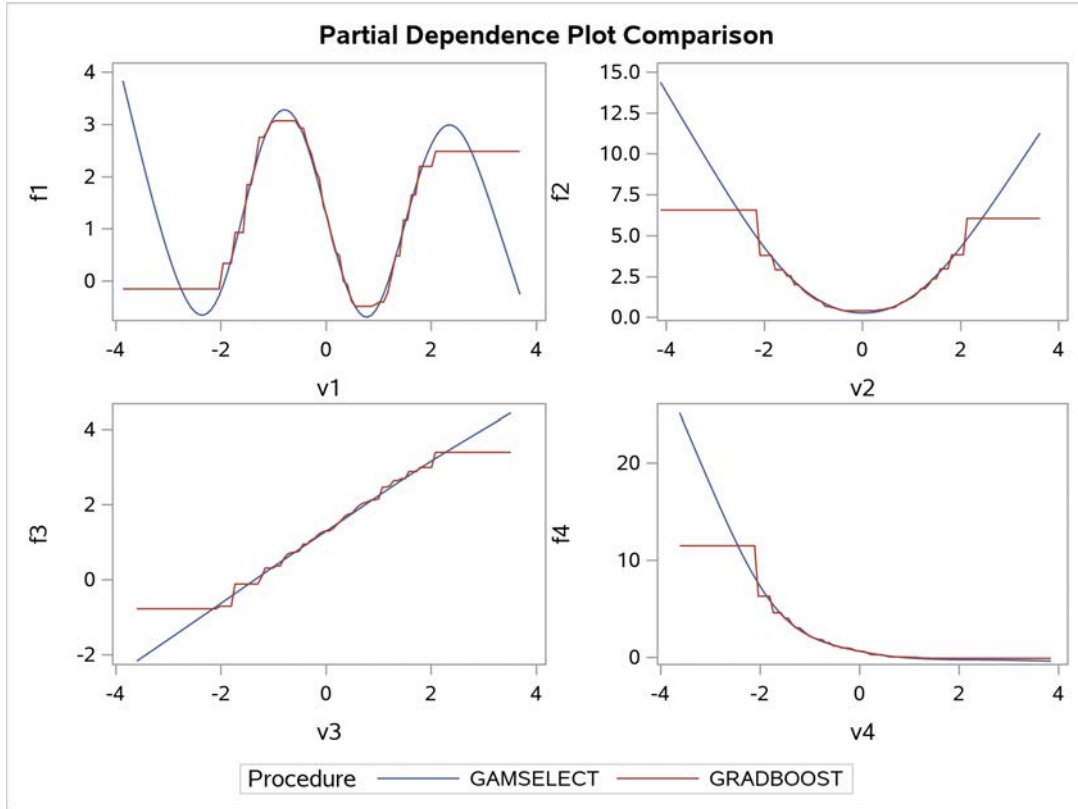
The GRADBOOST procedure does not build parsimonious predictive models. It does produce measures of variable importance that you can use to assess the importance of a variable to the model prediction. The “Variable Importance” table (Figure 16) ranks the variables in the model according to the residual sum of squares importance measure. This metric identifies the variables **v1–v4** as the variables most important to the model predictions.

Figure 16 Variable Importance for Default PROC GRADBOOST Model

| Variable Importance | | | |
|---------------------|------------|--------------------|---------------------|
| Variable | Importance | Std Dev Importance | Relative Importance |
| v4 | 1111.34 | 1817.28 | 1.0000 |
| v2 | 558.74 | 318.39 | 0.5028 |
| v1 | 535.44 | 556.46 | 0.4818 |
| v3 | 277.73 | 339.07 | 0.2499 |
| v79 | 13.2417 | 150.41 | 0.0119 |
| v65 | 7.7189 | 127.86 | 0.0069 |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| v57 | 0.1380 | 0 | 0.0001 |
| v136 | 0.1348 | 0 | 0.0001 |
| v191 | 0.1145 | 0 | 0.0001 |
| v7 | 0.08875 | 0 | 0.0001 |
| v27 | 0.07487 | 0 | 0.0001 |
| v66 | 0.06919 | 0 | 0.0001 |

The SAVESTATE statement is used to create an analytic store for the model that you can use to score new observations. The analytic store is saved in the data table **gbTreeMod** in the current CAS engine libref **mycas**. You can use the analytic store with the **partialDependence** action in the **explainModel** action set, part of SAS Visual Data Mining and Machine Learning, to compute partial dependence functions that can be used to create partial dependence plots. Figure 17 compares the partial dependence plots for the variables **v1–v4** for the model fit by PROC GRADBOOST and the final model fit by PROC GAMSELECT in the section “The Boosting Selection Method in PROC GAMSELECT.”

Figure 17 Default PROC GRADBOOST Model



The default model that is fit by PROC GRADBOOST shows poor reconstructions in the tails of all four curves. Note that for these example data, the variables **v1–v200** were simulated from standard normal distributions, and the poor performance of the tree-based ensemble occurs in regions with limited data. One way that you can attempt to improve the fit of the tree ensemble prediction is to increase the number of bins into which the interval predictor variables are divided. By default, PROC GRADBOOST uses 50 bins. The following code refits the model by using PROC GRADBOOST with 500 bins:

```
proc gradboost data=mycas.one seed=1234 numBin=500;  
  target y;  
  input v1-v200;  
  savestate rstore=mycas.gbTreeMod2;  
  crossvalidation;  
run;
```

Increasing the number of bins to this level leads to a substantial increase in the time required to train the model and also improves the model fit. Figure 18 shows a cross-validation of the ASE of 1.25 with the increased bin count.

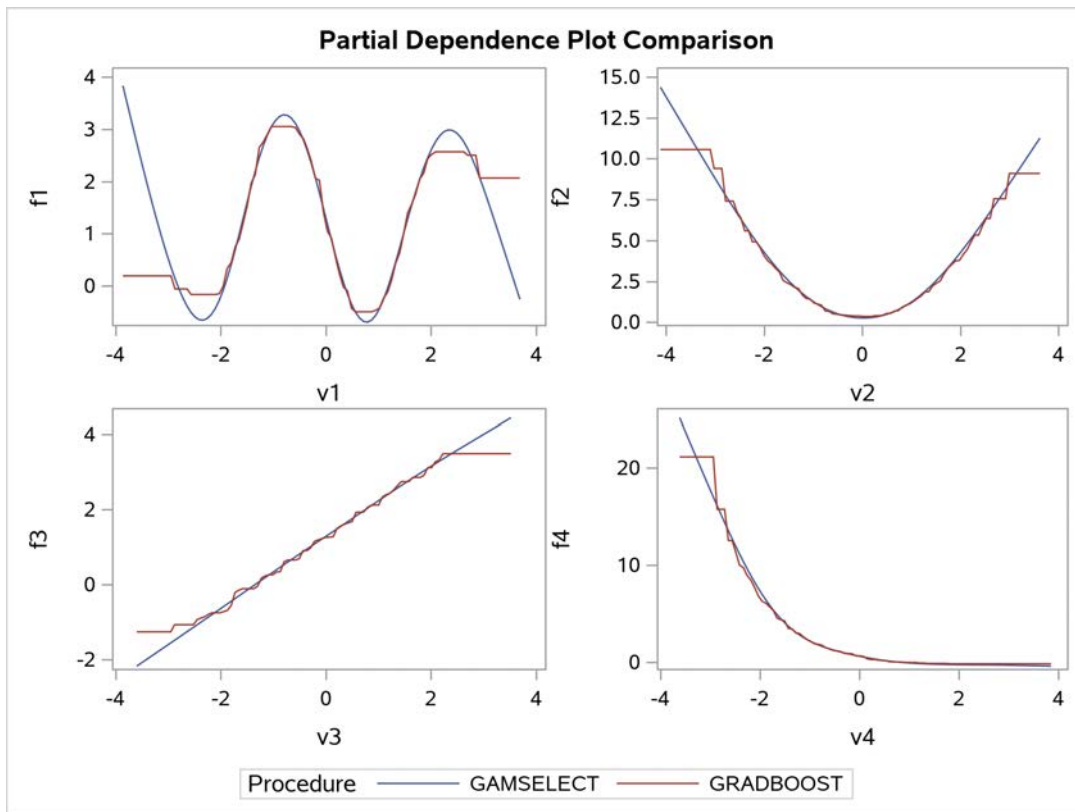
Figure 18 Cross-Validation Fit Statistics for Increased Bin Count

The GRADBOOST Procedure

| Cross-Validation Fit Statistics | | | | | | | | |
|---------------------------------|------------------------|--------------------|----------|--------------|----------------|-----------|---------------------------|-----------|
| Fold | Number of Observations | Squared Error | | | Absolute Error | | Squared Logarithmic Error | |
| | | Divisor of Average | Average | Root Average | Mean | Root Mean | Mean | Root Mean |
| Fold 1 | 2000 | 2000 | 1.341170 | 1.158089 | 0.898162 | 0.947714 | 0.488456 | 0.698897 |
| Fold 2 | 2000 | 2000 | 1.357408 | 1.165078 | 0.878380 | 0.937219 | 0.571462 | 0.755951 |
| Fold 3 | 2000 | 2000 | 1.175271 | 1.084099 | 0.850240 | 0.922085 | 0.451409 | 0.671870 |
| Fold 4 | 2000 | 2000 | 1.173475 | 1.083271 | 0.836542 | 0.914627 | 0.603089 | 0.776588 |
| Fold 5 | 2000 | 2000 | 1.196262 | 1.093738 | 0.871857 | 0.933733 | 0.471227 | 0.686460 |
| Average | 2000 | 2000 | 1.248717 | 1.116855 | 0.867036 | 0.931075 | 0.517129 | 0.717953 |

Figure 19 shows the partial dependence plots for the variables v_1 – v_4 for the refitted model. The increased bin count leads to much-improved reconstructions in the tails for the variables v_2 – v_4 , though a poor reconstruction remains in the tails for the variable v_1 .

Figure 19 PROC GRADBOOST Model with Increased Bin Count



The Shrinkage Selection Method in PROC GAMSELECT

This section continues the analysis of the data table **one**, which was created at the beginning of this example, and demonstrates features of the shrinkage selection method in PROC GAMSELECT.

The following statements use the shrinkage selection method to fit and select an additive model by using the default settings:

```
proc gamselect data=mycas.one seed=123 plots=all;
  model y = %SplinePrefixList(v, 200);
  selection method=shrinkage;
run;
```

By default, the shrinkage selection method fits and evaluates models at different values of sparsity-inducing penalty parameters. The convergence status table (Figure 20) appears in the output as a note that indicates the convergence status of the models. In this example, all the model fits successfully converged within the specified number of iterations. Other possible statuses for the model fits would be that the selected model converged but some penalty parameter fits did not, that the selected model reached the maximum iteration limit, or that the selected model did not converge. The table includes a nonprinting numeric variable, **Status**, that you can use to programmatically assess the convergence status.

Figure 20 Convergence Status for Default Model

The GAMSELECT Procedure

Selected model converged. All regularization parameters succeeded.

The “Iteration Summary” table (Figure 21) provides more information about the selected model and the convergence of model fits that were tried during the selection process. For this example, PROC GAMSELECT evaluated 200 models with different combinations of penalty parameter values, and all 200 model fits converged successfully. The selected model includes six effects, counting the intercept, and converged in 12 reweighted iterations. The five spline terms in the selected model include the four spline terms for the true effects and one noise spline term for the variable **v15**.

Figure 21 Iteration Summary and Selected Effects for Default Model

| Iteration Summary | |
|--|-----|
| Number of Penalty Parameter Set Tries | 200 |
| Number of Penalty Parameter Set Successes | 200 |
| Number of Reweighted Iterations for Selected Model | 12 |
| Number of Selected Effects | 6 |

Selected Effects Intercept Spline(v1) Spline(v2) Spline(v3) Spline(v4) Spline(v15)

The “Shrinkage Regularization Parameter Summary” table (Figure 22) shows the upper bounds for the regularization penalty values evaluated and the penalty parameter values for the selected model. The sparsity, smoothness, and generalized ridge penalties correspond to λ_1 , λ_2 , and λ_3 , respectively, using the notation in the section “Shrinkage.”

Figure 22 Regularization Parameter Summary for Default Model

| Shrinkage Regularization Parameter Summary | | |
|--|----------|----------|
| Penalty | Maximum | Chosen |
| Sparsity | 2.097253 | 0.030225 |
| Smoothness | 0.5503 | 0.001075 |
| Generalized Ridge | 0 | 0 |

The penalty values for each spline term in the selected model are displayed in the “Spline Regularization” table (Figure 23). Using the notation of the “Shrinkage” section, the spline sparsity, smoothness, and generalized ridge penalty values for spline j correspond to $\lambda_1 \phi_j \sqrt{\beta_j' \mathbf{K}_j \beta_j}$, $\lambda_2 \psi_j \sqrt{\beta_j' \mathbf{\Omega}_j \beta_j}$, and $\lambda_3 \beta_j' \mathbf{\Omega}_j \beta_j$, respectively. By default, the ϕ_j and ψ_j values are equal to 1 for each spline term.

Figure 23 Spline Regularization for Default Model

| Spline Regularization | | | |
|-----------------------|------------------|--------------------|---------------------------|
| Component | Sparsity Penalty | Smoothness Penalty | Generalized Ridge Penalty |
| Spline(v1) | 0.0422 | 0.0143 | 0 |
| Spline(v2) | 0.0436 | 0.00521 | 0 |
| Spline(v3) | 0.0289 | 0.000061 | 0 |
| Spline(v4) | 0.0620 | 0.0120 | 0 |
| Spline(v15) | 0.000088 | 0.000031 | 0 |

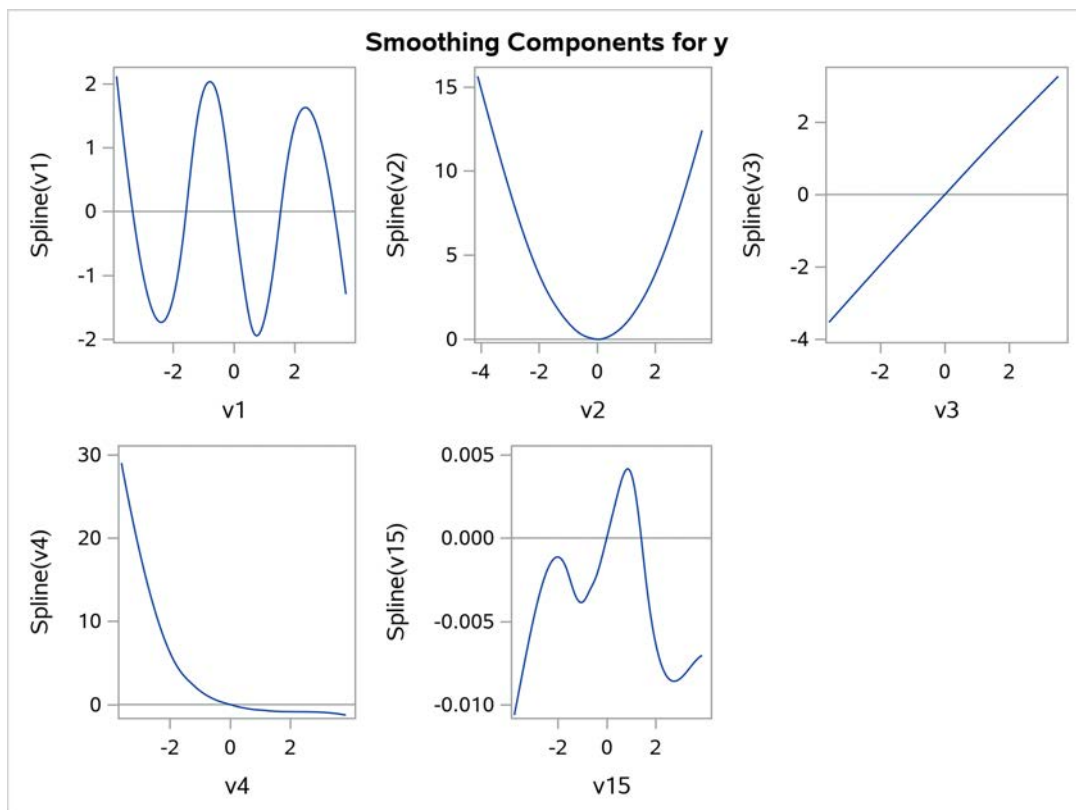
Figure 24 displays the fit statistics for the selected model; in this case, there is only one—the ASE.

Figure 24 Fit Statistics for Default Model

| Fit Statistics | |
|----------------------|---------|
| Average Square Error | 1.01459 |

The smoothing component plots for the selected effects in Figure 25 show well-reconstructed curves for the four spline terms that are constructed from the variables **v1–v4**. The relatively small vertical scale of **Spline(v15)** compared to the other terms indicates a small contribution from this effect.

Figure 25 Smoothing Component Panel for Default Model



To further refine the model fit by the shrinkage selection method, you can apply a data-adaptive procedure for forming the sparsity and smoothness penalties (Meier, Van de Geer, and Bühlmann 2009). This approach, similar to the adaptive LASSO (Zou 2006), involves specifying weights for the individual spline terms on the basis of previously computed sparsity and smoothness penalty values. You can compute the adaptive weights for the j th spline term as

$$\phi_j = \frac{1}{\sqrt{\hat{\beta}'_j \mathbf{K}_j \hat{\beta}_j}}$$

$$\psi_j = \frac{1}{\sqrt{\hat{\beta}'_j \mathbf{\Omega}_j \hat{\beta}_j}}$$

The following program uses the WEIGHT1 and WEIGHT2 spline options to specify sparsity and smoothness penalty weights:

```
proc gamselect data=mycas.one seed=123 plots=all;
  model y=spline(v1 / weight1=0.716 weight2=0.075)
        spline(v2 / weight1=0.693 weight2=0.206)
        spline(v3 / weight1=1.046 weight2=17.62)
        spline(v4 / weight1=0.488 weight2=0.090)
        spline(v15/ weight1=343.5 weight2=34.68);
  selection method=shrinkage;
run;
```

Figure 26 displays the selected effects and ASE for the adaptive model.

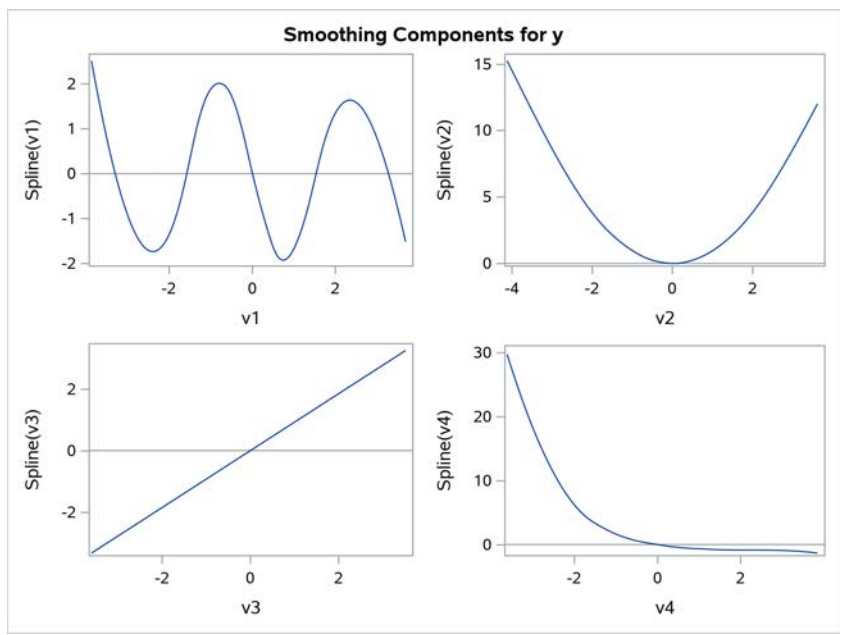
Figure 26 Selected Effects and Fit Statistics for Adaptive Model

The GAMSELECT Procedure

| | |
|-----------------------------|---|
| Selected Effects | Intercept Spline(v1) Spline(v2) Spline(v3) Spline(v4) |
| Fit Statistics | |
| Average Square Error | 1.01867 |

The adaptive model fit selects a model that consists of only the intercept and the four true spline terms. The smoothing component plots for the selected model (Figure 27) also display well-reconstructed curves for all four terms.

Figure 27 Smoothing Component Panel for Adaptive Model



Example 2: Testing Model Fit with Partitioned Data

This section demonstrates the use of partitioned data to evaluate a model fit. The example uses the home equity data set **Hmeq**, which is in the **Sampsio** library that SAS provides. Table 2 describes the variables in **Hmeq**.

Table 2 Variables in **Hmeq**

| Variable | Description |
|----------------|---|
| Bad | 1 = applicant defaulted on the loan or is seriously delinquent 0 = applicant paid off the loan |
| CLAge | Age of oldest credit line in months |
| CLNo | Number of credit lines |
| DebtInc | Debt-to-income ratio |
| Delinq | Number of delinquent credit lines |
| Derog | Number of major derogatory reports |
| Job | Occupational category |
| Loan | Requested loan amount |
| MortDue | Amount due on mortgage |
| nInq | Number of recent credit inquiries |
| Reason | 'DebtCon' = debt consolidation 'HomeImp' = home improvement |
| Value | Value of property |
| YoJ | Years at present job |

The following statements load the data set **Hmeq** onto the CAS server:

```
data mycas.hmeq;
  set sampsio.hmeq;
  if cmiss(of _all_) then delete;
  if CLAge > 1000 then delete;
  part = ranbin(1,1,0.2);
run;
```

In addition to creating the data table **mycas.Hmeq**, the statements remove observations that contain missing values; remove one observation with a reported credit line over 1,000 months old; and add a variable, **Part**, which will be used to partition the data table. Note that deleting the observations that contain missing values removes a large number of observations from the data table. Moreover, the proportion of applicants that default is substantially higher in the set of observations that are removed than in the set of complete case observations that remain in the data table. This difference between the remaining complete case observations that are used to fit and select the generalized additive model and the deleted observations might have important implications, depending on the intended use of the model.

The following statements fit and select a model by using the five parametric effects, seven univariate spline terms, and three bivariate spline terms that are specified in the MODEL statement. The three bivariate spline terms are included to model interactions between the value of the property and debts related to the property.

```
proc gamselect data=mycas.hmeq plots=all;
  class Job Reason ;
  model Bad(event='1') = Param(Job Reason nInq Derog Delinq)
    spline(CLAge) spline(CLNo) spline(DebtInc)
    spline(Loan) spline(Mortdue) spline(Value)
    spline(YoJ) spline(MortDue Value) spline(MortDue Loan)
    spline(Loan Value)
  / dist=binary allobs;
  selection method=boosting(stopHorizon=10 stopTol=0.0005 stepSize=0.2);
  output out=mycas.hmeqPred copyvars=( _all_ ) pred = p role=rInd;
  partition role=part(test='1');
run;
```

The variable **Part** that is specified in the **ROLE=** option in the **PARTITION** statement is used to assign observations to different roles. Observations that have the level of one for the variable **Part** are assigned to the test data role; all other observations are assigned to the training data role. The **ALLOBS** option in the **MODEL** statement specifies that all observations be used to construct the spline basis functions. By default, only observations that have the training data role are used to determine the knot values that are used to construct the spline basis functions. Observations with the validation or test role that have spline variable values outside the range of values that is used to construct the spline basis functions are omitted from the analysis, and a message noting the omission is printed in the log.

In the **SELECTION** statement, the **STOPHORIZON=** and **STOPTOL=** options request the use of early stopping on the basis of a relative change in the model selection criterion. Because no validation data partition is specified for this example, the model selection criterion is the average square error for the training data. For this example, a step size of 0.2 is specified instead of the default value of 0.1.

The “Response Profile” table (Figure 28) displays the number of observations that have the training and test role for each level of the response variable **Bad**. The **EVENT='1'** response option in the **MODEL** statement specifies 1, which indicates the applicant defaulting or the loan being in serious delinquency, as the event category.

Figure 28 Response Profile Table
The GAMSELECT Procedure

| Response Profile | | | | | |
|------------------|-----|-----------------|--------------------|----------------------|----------------|
| Ordered Value | BAD | Total Frequency | Training Frequency | Validation Frequency | Test Frequency |
| 1 | 0 | 3064 | 2451 | 0 | 613 |
| 2 | 1 | 299 | 247 | 0 | 52 |

Probability modeled is BAD = 1.

The “Iteration Summary” table (Figure 29) shows that the boosting selection process terminates early and selects the model that corresponds to the 273rd iteration. The selected model includes the six effects, not counting the intercept, that are displayed in the “Selected Effects” table (Figure 29).

Figure 29 Iteration Summary and Selected Effects

| Iteration Summary | |
|----------------------------|-----|
| Iterations Executed | 273 |
| Selected Iteration | 273 |
| Number of Selected Effects | 6 |

| Selected Effects | | |
|-----------------------|-----------------|----------------|
| Effect | Entry Iteration | Times Selected |
| Spline(DEBTINC) | 1 | 103 |
| DELINQ | 16 | 44 |
| DEROG | 21 | 34 |
| Spline(CLAGE) | 84 | 41 |
| Spline(LOAN VALUE) | 94 | 35 |
| Spline(MORTDUE VALUE) | 193 | 17 |

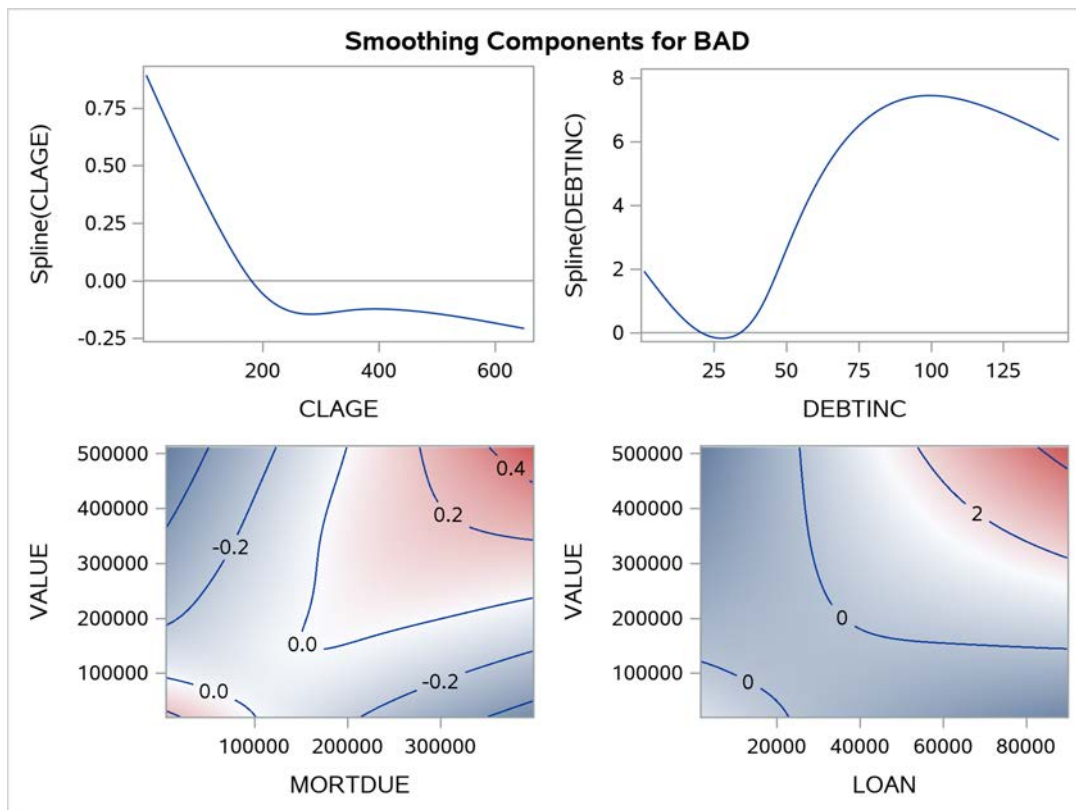
The model includes two parametric effects, two univariate spline terms, and two bivariate spline terms. The parameter estimates for the parametric effects are displayed in the “Parameter Estimates” table (Figure 30). The intercept estimate that is reported in the table is determined by the initial intercept-only model fit that initializes the boosting algorithm and the intercept estimate that is associated with the parametric effects at each iteration at which they were selected. The variables **Derog** and **Delinq** are both treated as continuous effects, and the sign of the parameter estimates indicates a higher predicted probability of a loan defaulting or becoming serious delinquent as the applicant’s number of major derogatory reports and number of delinquent credit lines increase.

Figure 30 Parameter Estimates for Parametric Effects

| Parameter Estimates | |
|---------------------|-----------|
| Parameter | Estimate |
| Intercept | -2.563425 |
| DEROG | 0.488251 |
| DELINQ | 0.454980 |

The smoothing component plots for the selected spline terms are displayed in Figure 31. The curves for the univariate spline terms **Spline(CLAge)** and **Spline(DebtInc)** show what you might expect to see. The probability of a default or seriously delinquent loan initially decreases as the age of an applicant’s oldest line of credit increases before leveling off, and the probability is higher for applicants who have high debt-to-income ratios. The contour plots for the bivariate spline terms **Spline(MortDue Value)** and **Spline(Loan Value)** indicate higher predicted probabilities of a default for applicants who have larger outstanding amounts due for more expensive properties and applicants who request larger loan amounts for more expensive properties.

Figure 31 Smoothing Component Panel



The “Fit Statistics” table for this model (Figure 32) shows the average square error and misclassification rate for the training and test data.

Figure 32 Fit Statistics

| Fit Statistics | |
|--------------------------------|---------|
| ASE (Train) | 0.05861 |
| ASE (Test) | 0.05113 |
| Misclassification Rate (Train) | 0.07079 |
| Misclassification Rate (Test) | 0.05714 |

To further examine the model fit, you can use the output data table **hmeqPred** that is requested by the OUTPUT statement and created in the current CAS engine libref named **mycas**. In the OUTPUT statement, the COPY-VARS=(**_ALL_**) option copies all variables in the input data table to the output data table, and the PRED= and ROLE= keywords name two variables to add to the output table. The variable **p** contains the predicted probability of an event based on the model fit, and the variable **rInd** is a numeric variable that indicates the role played by an observation in the model fitting process.

The following code creates the indicator variable **I_Bad** on the basis of the predicted event probabilities and uses PROC FREQTAB, a procedure in SAS Visual Statistics, to compute crosstabulation tables:

```
data mycas.hmeqPred;
  set mycas.hmeqPred;
  if p >= 0.5 then I_Bad = 1;
  else if p < 0.5 and p ne . then I_Bad = 0;
  else I_Bad = .;
run;

proc freqtab data=mycas.hmeqPred;
  table rInd * I_Bad * Bad / nopercents;
run;
```

Figure 33 compares the predicted and observed outcomes for observations in the training data role, **rInd=1**, and test data role, **rInd=3**. All the misclassified observations are loan applications for which the event of a default or highly delinquent loan occurred.

Figure 33 Predicted versus Observed Outcome by Data Role for PROC GAMSELECT

| Frequency Row Pct Col Pct | Table 1 of I_Bad by BAD | | | | Frequency Row Pct Col Pct | Table 2 of I_Bad by BAD | | | |
|---------------------------------|-------------------------|--------|-------|--------------|---------------------------------|-------------------------|-------|--|--|
| | Controlling for rInd=1 | | | | | Controlling for rInd=3 | | | |
| | I_Bad | BAD | | | | I_Bad | BAD | | |
| 0 | | 1 | Total | 0 | 1 | | Total | | |
| 0 | 2451 | 191 | 2642 | 0 | 613 | 38 | 651 | | |
| | 92.77 | 7.23 | | | 94.16 | 5.84 | | | |
| | 100.00 | 77.33 | | | 100.00 | 73.08 | | | |
| 1 | 0 | 56 | 56 | 1 | 0 | 14 | 14 | | |
| | 0.00 | 100.00 | | | 0.00 | 100.00 | | | |
| | 0.00 | 22.67 | | | 0.00 | 26.92 | | | |
| Total | 2451 | 247 | 2698 | Total | 613 | 52 | 665 | | |

For comparison, the following statements use PROC SVMACHINE in SAS Visual Data Mining and Machine Learning to fit an alternative model by using support vector classification and a polynomial kernel of degree two:

```
proc svmachine data=mycas.hmeq;
  input reason job derog delinq ninq / level=nominal;
  input loan mortdue value yoj clage clno debtinc / level=interval;
  kernel polynomial/degree=2;
  target bad / desc;
  partition role=part(test='1');
run;
```

The misclassification matrix for the model that is fit by PROC SVMACHINE (Figure 34) shows similar performance on the test data to that of the model fit by PROC GAMSELECT. Although not shown here, the GRADBOOST procedure can be used to fit a tree ensemble that produces a misclassification rate of about 0.045 for the test data, and the LOGSELECT procedure can be used to fit and select a model that produces a misclassification rate of about 0.069 for the test data.

Figure 34 SVM Model Misclassification Matrix

The SVMACHINE Procedure

| Misclassification Matrix | | | | | | |
|--------------------------|---------------------|------|-------|--------------------|-----|-------|
| Observed | Training Prediction | | | Testing Prediction | | |
| | 1 | 0 | Total | 1 | 0 | Total |
| 1 | 100 | 147 | 247 | 13 | 39 | 52 |
| 0 | 3 | 2448 | 2451 | 1 | 612 | 613 |
| Total | 103 | 2595 | 2698 | 14 | 651 | 665 |

Note that for this example, using PROC GAMSELECT to create the crosstabulation tables by using the variable **Part**, which is specified in the PARTITION statement, or the variable **rlnd**, which is created in the OUTPUT statement, produces the same results. For some data and models this might not be the case, and there might be observations for which the levels of the variable listed in the ROLE= option in the PARTITION statement and the variable created by the ROLE= option in the OUTPUT statement do not correspond. For example, because observations that have a missing value for one of the independent variables are not used to fit the model, they are assigned the role of not used for the role variable that is created by the OUTPUT statement. This role assignment might not correspond to the level of the variable that is specified in the PARTITION statement.

Additionally, the values in the crosstabulation tables that are computed from the output data table might differ from the values shown in the “Fit Statistics” table that PROC GAMSELECT produces. The “Fit Statistics” table reports values that are computed using only the observations with nonmissing values for all variables listed in the MODEL statement and values that are within the knot range used to construct the spline basis functions. The OUTPUT statement computes predicted probabilities for any observation that can be scored on the basis of the selected model and therefore might include additional observations that are not used in the model fitting process.

SUMMARY

The GAMSELECT procedure adds to the rich set of tools for performing model selection available in SAS Visual Statistics by supporting model selection methods designed specifically for use on generalized additive models. Generalized additive models can be used to fit flexible predictive models by using spline terms to model nonlinear or unknown dependence structures. You can use the boosting and shrinkage model selection methods in PROC GAMSELECT to fit parsimonious generalized additive models while also controlling the smoothness of the model fit. Moreover, you can use visualizations of the univariate and bivariate spline terms in the selected model to understand the contribution that the selected effects make to the model predictions. The examples in this paper demonstrate how you can use the estimation supported by PROC GAMSELECT, how you can benefit from using model selection methods designed specifically for generalized additive models, and how the model selection methods that are supported by PROC GAMSELECT compare to alternative modeling approaches.

REFERENCES

- Amato, U., Antoniadis, A., and De Feis, I. (2016). “Additive Model Selection.” *Statistical Methods and Applications* 25:519–564.
- Bühlmann, P., and Hothorn, T. (2007). “Boosting Algorithms: Regularization, Prediction and Model Fitting.” *Statistical Science* 22:477–505. <https://doi.org/10.1214/07-STS242>.
- Cantoni, E., Flemming, J. M., and Ronchetti, E. (2006). “Variable Selection in Additive Models by Non-negative Garrote.” *Statistical Modelling* 11:237–252.

- Chouldechova, A., and Hastie, T. (2015). “Generalized Additive Model Selection.” Technical paper. Stanford University.
- De Boor, C. (1978). *A Practical Guide to Splines*. New York: Springer-Verlag.
- Eilers, P. H. C., and Marx, B. D. (1996). “Flexible Smoothing with *B*-Splines and Penalties.” *Statistical Science* 11:89–121. With discussion.
- Friedman, J. H. (2001). “Greedy Function Approximation: A Gradient Boosting Machine.” *Annals of Statistics* 29:1189–1232.
- Lin, Y., and Zhang, H. H. (2006). “Component Selection and Smoothing in Multivariate Nonparametric Regression.” *Annals of Statistics* 34:2272–2297.
- Marra, G., and Wood, S. N. (2011). “Practical Variable Selection for Generalized Additive Models.” *Computational Statistics and Data Analysis* 55:2372–2387.
- Marx, B. D., and Eilers, P. H. C. (2005). “Multidimensional Penalized Signal Regression.” *Technometrics* 47:13–22. <https://doi.org/10.1198/004017004000000626>.
- Meier, L., Van de Geer, S., and Bühlmann, P. (2009). “High-Dimensional Additive Modeling.” *Annals of Statistics* 37:3779–3821.
- Raskutti, G., Wainwright, M. J., and Yu, B. (2012). “Minimax-Optimal Rates for Sparse Additive Models over Kernel Classes via Convex Programming.” *Journal of Machine Learning Research* 13:389–427.
- Ravikumar, P., Lafferty, J., Liu, H., and Wasserman, L. (2009). “Sparse Additive Models.” *Journal of the Royal Statistical Society, Series B* 71:1009–1030.
- SAS Institute Inc. (2019). *SAS Visual Statistics 8.5: Procedures*. Cary, NC: SAS Institute Inc. <https://go.documentation.sas.com/?docsetId=casstat&docsetTarget=titlepage.htm&docsetVersion=8.5&locale=en>.
- Suzuki, T., and Sugiyama, M. (2013). “Fast Learning Rate of Multiple Kernel Learning: Trade-Off between Sparsity and Smoothness.” *Annals of Statistics* 41:1381–1405.
- Yuan, M., and Lin, L. (2006). “Model Selection and Estimation in Regression with Grouped Variables.” *Journal of the Royal Statistical Society, Series B* 68:49–67.
- Zou, H. (2006). “The Adaptive Lasso and Its Oracle Properties.” *Journal of the American Statistical Association* 101:1418–1429.

ACKNOWLEDGMENTS

The authors are grateful to Ed Huddleston for his valuable editorial assistance.

| | |
|----------------------|--------------------|
| Michael Lamm | Weijie Cai |
| SAS Institute Inc. | SAS Institute Inc. |
| SAS Campus Drive | SAS Campus Drive |
| Cary, NC 27513 | Cary, NC 27513 |
| Michael.Lamm@sas.com | Weijie.Cai@sas.com |

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.