Paper SAS4502-2020

How to Explain Your Black-Box Models in SAS® Viya®

Funda Güneş, Ricky Tharrington, Ralph Abbey, and Xin Hunt, SAS Institute Inc.

ABSTRACT

SAS® Visual Data Mining and Machine Learning in SAS® Viya® offers a number of algorithms for training powerful predictive models, such as gradient boosting, forest, and deep learning models. Although these models are powerful, they are often too complex for people to understand by directly inspecting the model parameters. The "black-box" nature of these models limits their use in highly regulated industries such as banking, insurance, and health care. This paper introduces various model-agnostic interpretability techniques available in SAS Viya that enable you to explain and understand machine learning models. Methods include partial dependency (PD) plots, independent conditional expectation (ICE) plots, local interpretable model-agnostic explanations (LIME), and Shapley values. This paper introduces these methods and demonstrates their use in two scenarios: a business-centered modeling task and a health-care modeling task. Also shown are the two different interfaces to these methods in SAS Viya: Model Studio and the SAS Viya programming interface.

INTRODUCTION

Modern machine learning algorithms can make accurate predictions by modeling the complex relationship between inputs and outputs. The algorithms build predictive models by learning from the training data, and then make predictions on new observations. Although machine learning algorithms can learn complex relationships, the models that are produced can be equally complex, making it difficult to understand the association between the inputs and outputs. Because of their complexity, many machine learning models are seen as black-box models, producing predictions without explaining why and how they are made.

One example of a black-box machine learning model is a simple neural network model with one or two hidden layers. Even though you can write out the equations that link every input in the model to every output, you might not be able to grasp the meaning of the connections simply by examining the equations. This has less to do with the shortcomings of the models, and more to do with the shortcomings of human cognition. Often, the higher the predictive accuracy of a model, the harder it is to interpret its inner workings. This is where interpretability techniques come into play by providing a lens through which you can view these complex models.

Model interpretability can meet different needs for different users, such as regulators, executives, data scientists, and domain experts. Regulators need model interpretability to make sure the model makes predictions for the right reasons. For example, if an individual's loan application is rejected, the loan agency needs to confirm that this decision does not violate any laws that protect certain groups of people. Executives need to understand black-box models so that they can logically justify the decisions they make. Lastly, data scientists (with the help of domain experts) need model interpretability to be able to detect biases that exist in training data, extract new knowledge that is originally hidden in the data, and debug models when they produce wrong and unexpected predictions.

In recent years, interpretable machine learning has been the topic of much research. The methods for interpreting models can be categorized into two groups: training inherently interpretable models and providing post-hoc explanations for complex machine learning

models. Inherently interpretable models (also called explainable models) incorporate interpretability directly into the model structure, and thus are self-explanatory. One type of commonly used inherently interpretable models is the generalized linear model (GLM), which includes linear and logistic regression. The coefficient estimates of GLMs directly reflect feature contributions; hence, these models can be explained through these coefficients. More recently introduced examples of inherently interpretable models achieve interpretability by forcing the models to use fewer features for prediction or by enabling features to have monotonic relationships with the prediction (Ustun and Rudin 2015). Another example of inherently interpretable models is generalized additive models with pairwise interactions (GA2M). These models enable you to understand the contribution of features through their additive components (Caruana et al. 2015). These constraints can make complex models simpler and increase the model's comprehensibility to users. However, imposing these constraints can also decrease the predictive ability of the model when compared to an unrestricted model.

This paper focuses on exploring post-hoc interpretability methods that are used to explain trained supervised machine learning models such as boosted trees, forests, and neural networks. These model-agnostic techniques explain predictions of these models by treating the models as black boxes and then generating explanations without inspecting the internal model parameters.

In general, model-agnostic interpretability techniques enable fully complex models to be interpreted either globally or locally. Global interpretability provides explanations about the general behavior of the model over the entire population. For example, global interpretability might explain which variables played an important role in the construction of the model or describe the impact of each feature on the overall prediction of the model. In contrast, local interpretability provides explanations for a specified prediction of the model.

This paper first explains various global and local post-hoc interpretability techniques, and then applies these techniques to explain two previously trained models. The first example explains a gradient boosting model that was trained on financial data that includes loan records. This example is shown in SAS Model Studio, a highly flexible point-and-click web interface for constructing machine learning pipelines. The second example explains a forest model that was trained on health-care data for predicting the malignancy of potential breast cancer biopsies. This example uses the programmatic interface in SAS Viya, which offers more user control in requesting explanations and is therefore more suitable for advanced users.

GLOBAL INTERPRETABILITY

Post-hoc global interpretability aims to provide a global understanding about what is learned by the pretrained model over the entire population and presents this learned knowledge in a way that humans can understand. Variable importance tables and partial dependence (PD) plots are two commonly used global interpretability techniques. This section also covers individual conditional expectation (ICE). Although ICE is a local explanation technique, it is related to PD.

VARIABLE IMPORTANCE

Variable importance tables indicate the statistical contribution of each feature to the underlying model. There are various ways of calculating model-agnostic feature importance. One method includes fitting a global surrogate decision tree model to the black-box model predictions and using the variable importance table that is produced by this simple decision tree model. Another commonly used approach is permutation-based feature importance as described in Altmann et al. (2010). This approach measures the decrease in model

predictive performance when a single feature is randomly shuffled. This technique can be computationally very expensive if the number of predictors is very large, because it requires training a new model (on the perturbed data) for each feature. If the pretrained model is tree-based (decision tree, gradient boosting, or forest), you can also use the model-specific variable importance table that is generated during the model construction. Generation of these tree-based model variable importance tables is often based on the number of times a feature is used to split data.

PARTIAL DEPENDENCE AND INDIVIDUAL CONDITIONAL EXPECTATION

Both PD and ICE provide explanations that are based on data perturbation, where the contribution of each feature is determined by measuring how a model's prediction changes when the feature is altered. Partial dependence (PD) plots depict the relationship between a feature and the average prediction of the pretrained model (Friedman 2001).

The following steps briefly describe how the partial dependence of a feature is calculated:

1. Select a single feature.
2. Select a sample of observations (rows).
3. For each row in your sample, replicate the row a number of times equal to the number of unique values for the selected feature (or the number of binned values for an interval variable), and replace the feature's value with its unique values. Do this for each row in your sample.
4. Score all the replicated data.
5. Average the model predictions that are produced by the replicated data for each unique value of the selected feature.
6. Generate a bar plot (or a line plot for an interval variable) that shows the unique values of the feature on the X axis and the average prediction on the Y axis.
7. Repeat steps 1–6 for each selected feature.

PD plots focus on the average effect of a feature for the entire data, whereas ICE plots focus on the effect of a feature for a single observation (Goldstein et al. 2014). By examining various ICE plots, you gain insight into how the same feature can have a different effect for different individuals or observations in the data.

The steps for creating an ICE plot for a feature are similar to the steps for creating PD plots except that ICE plots are done only for a single observation, and hence the Step 3 is done for a single row, and the averaging step listed in Step 5 is skipped. For more information about how PD and ICE plots are generated in SAS Viya, see Wright (2018).

LOCAL INTERPRETABILITY

This section describes local interpretability methods for explaining individual predictions. In general, these techniques assume that machine learning predictions in the neighborhood of a particular instance can be approximated by a white-box interpretable model such as a regularized linear regression model (LASSO). This local model does not have to work well globally, but it must approximate the behavior of the pretrained model in a small local region around the instance of interest. Then the parameters of the white-box model can be used to explain the prediction of the pretrained model.

For more information about the SAS Viya implementation of the techniques in this section, see the chapter "Explain Model Action Set" in *SAS Visual Data Mining and Machine Learning 8.5: Programming Guide*.

LIME

LIME (local interpretable model-agnostic explanations) explains the predictions of any model by building a white-box local surrogate model (Ribeiro, Singh, and Guestrin 2016). The method first samples the feature space in the neighborhood of an individual observation with respect to a training data set. Then, a sparse linear regression model, such as LASSO, is trained on this generated sample, using the predictions that are produced by the pretrained model as a target. This surrogate model approximates the behavior of the black-box model locally, but it is much easier to explain by examining the regression coefficients.

When performing the LIME method, SAS Viya constructs a local surrogate regression model by doing the following:

1. For each input variable, generates synthetic data that are centered on the observed value. This is done independently for each input variable by using the standard deviation from the overall training data set.
2. Scores the synthetic data by using the pretrained black-box model you want to explain.
3. Fits a weighted LASSO regression model to the synthetic data, where the target is the pretrained model score obtained in step 2.

   For each row in the synthetic data, the weight of the LASSO regression is the distance to the observation of interest (whose model prediction you want to explain), as shown by the following equation for the selected observation q, synthetic row x_i, and scaling factor σ:

   $$weight = e^{\frac{||q-x_i||}{\sigma^2}}$$

Shapley Values

Like LIME, the Shapley values explain individual predictions (Kononenko 2010). Different from LIME coefficients, Shapley values for feature contributions do not directly come from a local regression model. In regression models, the coefficients represent the effect of a feature assuming all the other features are already in the model. It is well-known that the values of the regression coefficients highly depend on the collinearity of the feature of interest with the other features that are included in the model. To eliminate this bias, Shapley values calculate feature contributions by averaging across all permutations of the features joining the model. This enables Shapley values to control for variable interactions.

The Shapley values are additive, meaning that you can attribute a portion of the model's predictive value to each of the observation's input variables. For example, if you have a model that is built with three input variables, then you can write the predicted value as summation of the corresponding Shapley values plus the average predicted value across the input data set. Note that even though Shapley values are additive, they are not ordered.

Because of computational complexities, there are multiple methods for computing approximations to Shapley values. SAS Viya offers the Kernel SHAP and HyperSHAP methods.

Kernel SHAP

The kernel SHAP method implementation in SAS Viya is based on the steps presented in Lundberg and Lee (2017). When performing this method, Viya does the following:

1. Generates synthetic data that are based on the distributions of the input data set.
2. Scores the data by using the machine learning model you want to explain.
3. Transforms both interval and nominal inputs into binary inputs, with a value of 1 if the

synthetic data row is close to the observation of interest, and 0 otherwise.

4. Computes a weighted regression to the model prediction by using the SHAP kernel, where the weights are calculated by the following equation, where M is the number of variables and $|z|$ is the number of 1's in the row of the transformed data:

$$weight = \frac{(M-1)}{(M\ choose\ |z|)*|z|*(M-|z|)},$$

HyperSHAP

The HyperSHAP method estimates the conditional expectation values without fitting a regression model. It reduces the number of necessary evaluations by controlling the degrees of variable interactions that are considered in the calculation. When computing the HyperSHAP values, SAS Viya does the following:

1. Accepts a parameter that specifies the depth of the approximation.
2. For all subsets of variables whose number of selected variables is less than or equal to the depth, and for all subsets of variables whose number of selected variables is greater than or equal to the total number of variables minus the depth: generates a copy of the training set where the variables' values are set equal to the values of the variable in the observation of interest.
3. Scores the new observations.
4. Averages the prediction for each data set copy.
5. Computes a weighted aggregation of the average predictions.

One benefit of the HyperSHAP method is that it enables you to choose the level of approximation to the Shapley values—a tradeoff between fidelity and timing.

Note that in SAS Viya 3.5, you can access HyperSHAP only through the programmatic user interfaces.

INTERPRETABILITY IN THE MODEL STUDIO USER INTERFACE

SAS Viya is a cloud-enabled, in-memory analytics platform that supports fast processing of huge amounts of data and complex analytics across a large distributed grid. SAS Visual Data Mining and Machine Learning running on SAS Viya solves complex analytical problems with a comprehensive visual interface that handles all tasks in the analytics life cycle, including data wrangling, exploration, feature engineering in addition to modern statistical, data mining, and machine learning techniques. All of this is done in a single, scalable, in-memory processing environment. You can access these tasks through SAS® Drive, which is a web-based single point of access.

Included in SAS Visual Data Mining and Machine Learning, the SAS Model Studio application is web-based visual interface that enables users to build predictive modeling pipelines. A pipeline is a structured flow of analytical nodes, each of which performs a single data mining or predictive modeling task. You access Model Studio by selecting Build Models from the Analytics Life Cycle menu in SAS Drive, as shown in Figure 1.

Figure 1. Access Model Studio by Selecting Build Models

When working in Model Studio, you construct pipelines by adding nodes through the point-and-click web interface. Figure 2 shows a simple Model Studio pipeline that performs missing data imputation, selects variables, constructs two logistic regression models and a decision tree model, and compares their predictive performances.

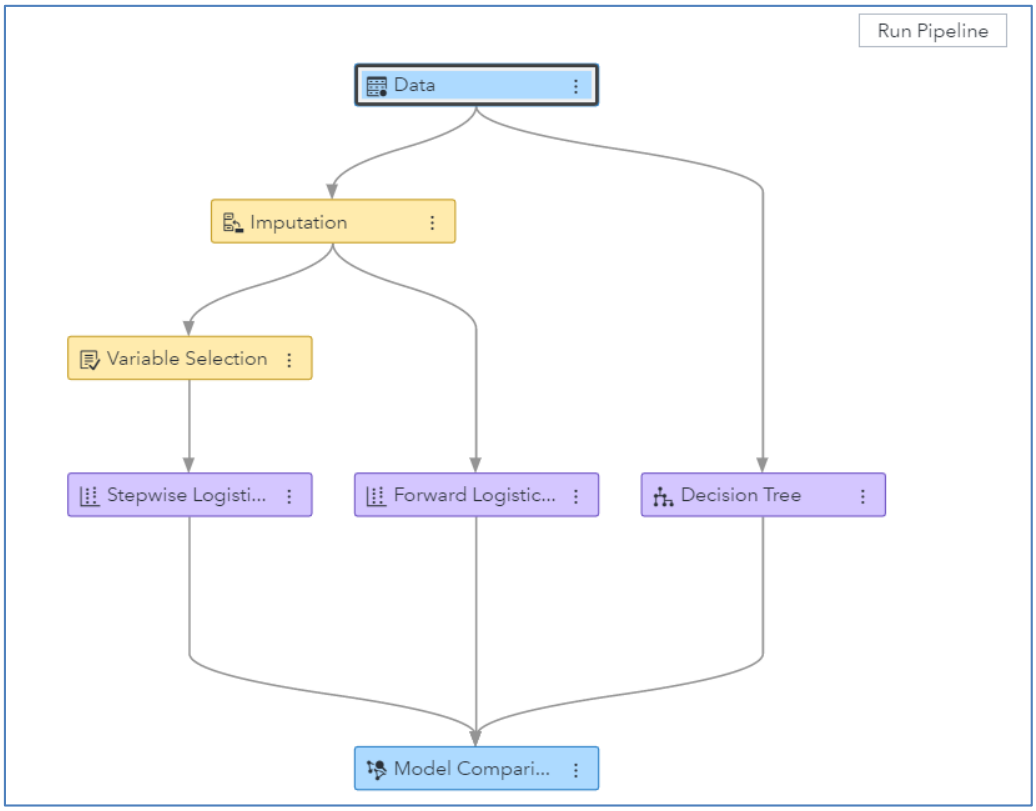

Figure 2. A Model Studio Pipeline in SAS Visual Data Mining and Machine Learning

Building pipelines is considered a best practice for predictive modeling tasks because pipelines can be saved and shared with other SAS Visual Data Mining and Machine Learning users for training future machine learning models on similar data. In addition to many feature engineering capabilities, Model Studio also offers numerous ways to tune, assess, combine, and interpret models.

In Model Studio, model interpretability functionalities are provided as a post-training property of all supervised learning nodes. Changing post-training properties and retrieving

interpretability results does not require retraining the machine learning model. Figure 3 shows the model interpretability properties for the Gradient Boosting supervised learning node.

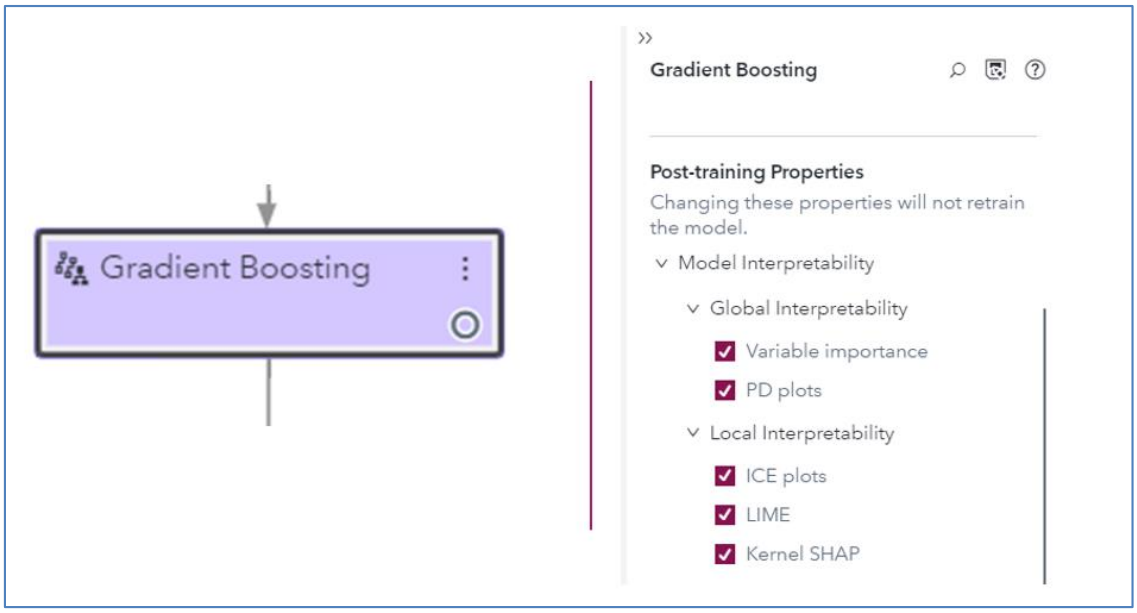

Figure 3. Model Interpretability Properties for the Gradient Boosting Node

In Model Studio, model interpretability results are presented in a user-friendly way by decreasing the huge amount of information that might be overwhelming to users. Instead, Model Studio includes text explanations that are provided by natural language generation for easier understanding of the results. This enables users who are less experienced with these techniques to find some meaningful insight into the relationships between the predictors and the target variable in black-box models. For more information about Model Studio, see *SAS Visual Data Mining and Machine Learning: User's Guide.*

CASE STUDY I: HELOC DATA

This case study demonstrates how to use SAS Model Studio for performing post-hoc model interpretability.

Data and Model

The data come from the FICO xML Challenge, which can be found at https://community.fico.com/s/explainable-machine-learning-challenge. The data are contained in an anonymized data set of home equity line of credit (HELOC) applications made by real homeowners. A HELOC is a line of credit typically offered by a bank as a percentage of home equity (the difference between the current market value of a home and any liens on the home). The customers in this data set have requested a credit line in the range of $5,000–$150,000.

The goal is to predict whether the applicants will repay their HELOC account within two years. The data set has 10,459 observations for a mix of 23 interval and nominal input variables, which include predictors such as the number of installment trades with balance, the number of months since the most recent delinquency, and the average number of months in file. The target variable is RiskPerformance, a binary variable that takes the value Good or Bad. The value Bad indicates that a customer's payment was at least 90 days past due at some point in the 24-month period after the credit account was opened. The value Good indicates that payments were made without ever being more than 90 days overdue. The data are balanced with around 52% Bad observations.

The focus of this paper is on explaining model interpretability, so the data preprocessing and feature generation steps are skipped. All the input variables were taken as-is, except for the variables Max Delq/Public Records Last 12 Months and Max Delinquency Ever, which were converted to strings according to the FICO data dictionary. In addition, an ID variable was created in order to specify individual applicants when performing local interpretability.

A Model Studio project (called heloc) is created using this data set. For more information about how to create a SAS Model Studio project, see the section Getting Started with SAS Visual Data Mining and Machine Learning in Model Studio in *SAS Visual Data Mining and Machine Learning: Users Guide*.

The data are partitioned by the default values of 60% for training, 30% for validation, and 10% for test sets, as shown in Figure 4.

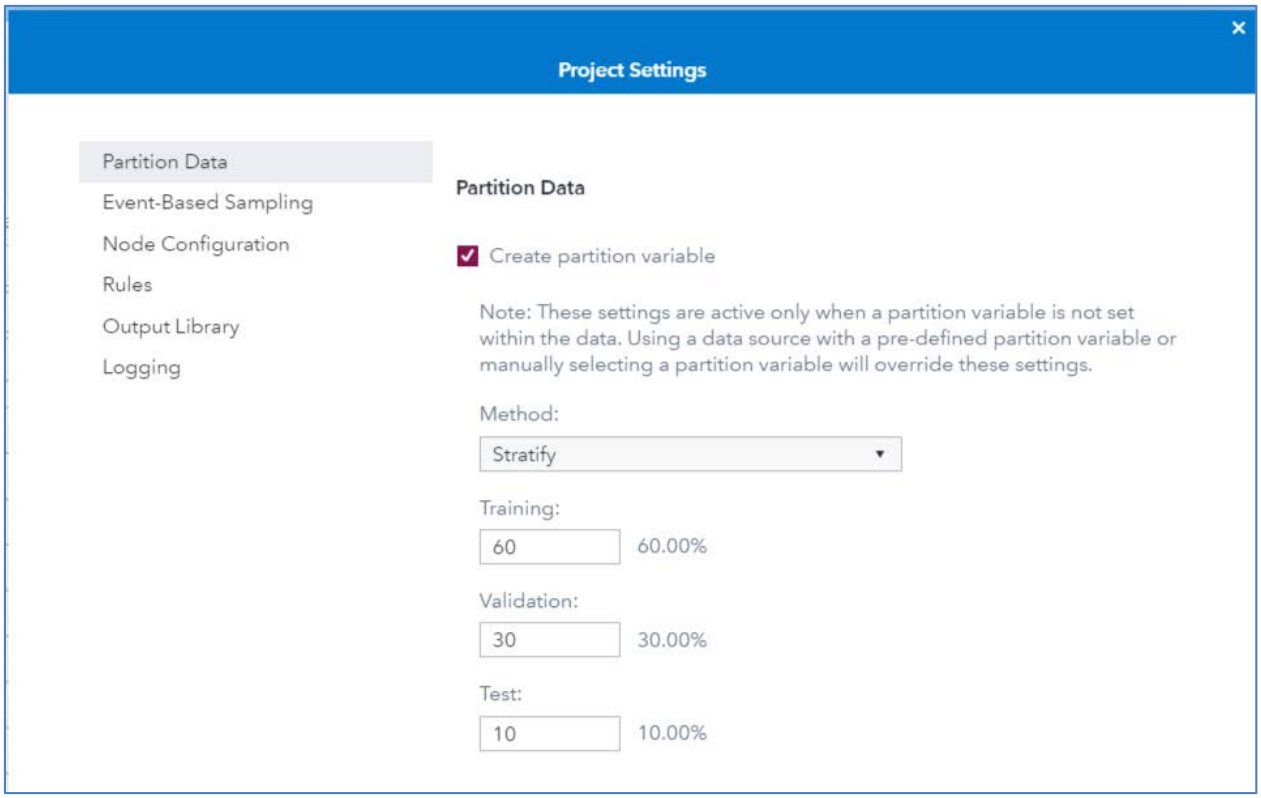

Figure 4. Project Settings

By clicking the Data tab, you can assign different roles to your input variables. Figure 5 shows that the newly created ID variable is assigned the role Key. This step is necessary if you want to specify individual predictions for local interpretability. Figure 5 also shows that the binary variable RiskPerformance is specified as the target variable.

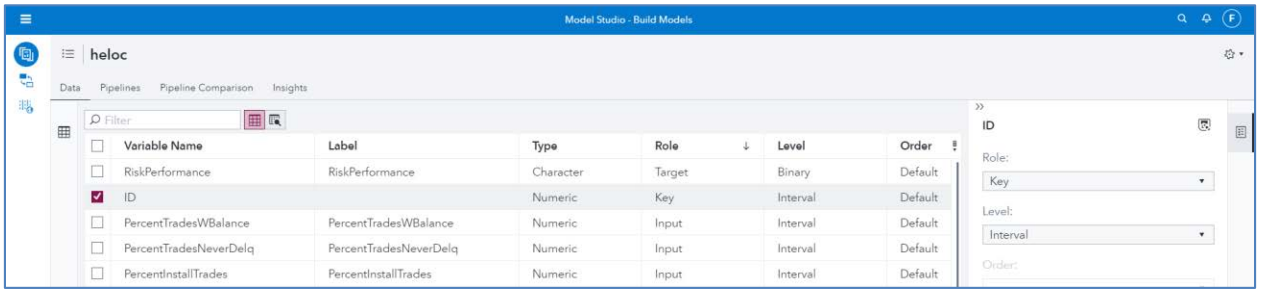

Figure 5. Data Tab

When specifying the target variable, you can choose the event level of the target variable. Figure 6 shows that event level is specified as Bad. This means the predicted probabilities of the trained models will represent the probabilities of a customer making a late payment.

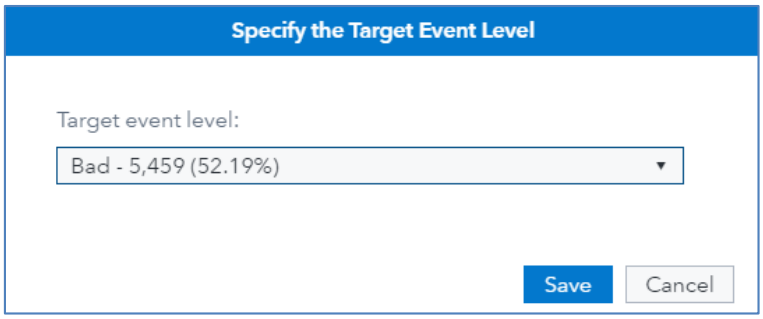

Figure 6. Specifying the Event Level for the Target Variable

To train a gradient boosting model in Model Studio, you simply need to connect the Gradient Boosting supervised learning node to the Data node. For this example, the Gradient Boosting node is run with its default settings without doing any hyperparameter tuning. By default, the validation set is used for early stopping to decide when to stop training boosted trees. Figure 7 shows the fit statistics of the black-box gradient boosting model.

Fit Statistics

Data Role	Average Squared Error	Misclassification Rate	Multi-Class Log Loss	Area Under ROC	KS (Youden)
TEST	0.1779	0.2782	0.5336	0.8076	0.4708
TRAIN	0.1494	0.2209	0.4609	0.8688	0.5611
VALIDATE	0.1857	0.2823	0.5525	0.7905	0.4349

Figure 7. Model Assessment Results, Model Fit Statistics

Figure 7 shows that the model's misclassification rate on the test set is 27.8%. Figure 8 shows the corresponding event classification plot, where the larger portion of the model's misclassification events are good applications that are predicted as bad.

Figure 8. Assessment Results, Event-Based Classification Plot

To improve prediction accuracy, you can perform a hyperparameter search for your gradient boosting model by turning on the Perform Autotuning property, which is available in all the supervised machine learning nodes in Model Studio. To learn more about automated hyperparameter tuning functionality in SAS Viya, see Koch et. al. (2018).

Global Interpretability

This section shows how you can request and view global interpretability plots.
Figure 9 shows the check boxes you use enable a node's global interpretability methods (variable importance and partial dependence plots) in Model Studio. Note that since the model interpretability techniques covered here are post-hoc, they are done after training

the gradient boosting model. This means that unless you change any model training properties, changing a post-training property such as model interpretability does not require retraining the model.

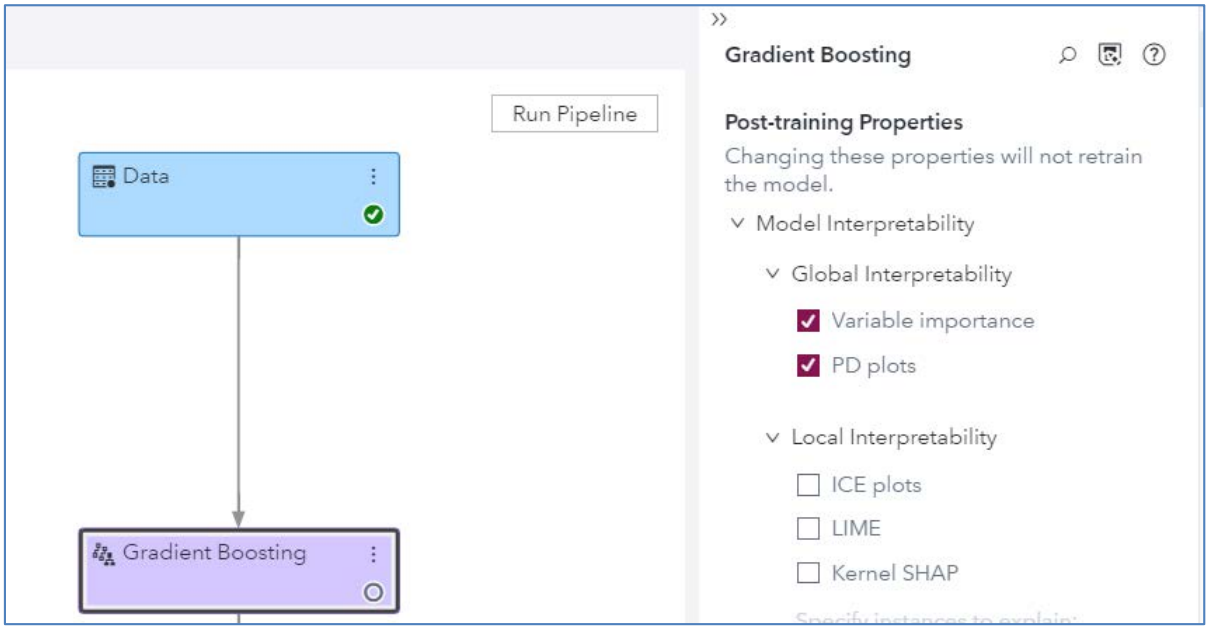

Figure 9. Requesting Global Interpretability for the Trained Gradient Boosting Model

Variable Importance

The model variable importance table in Figure 10 shows the ranking of the importance of the features in construction of the gradient boosting model.

Model Variable Importance

Variable Label	Relative Importance
ExternalRiskEstimate	1
AverageMInFile	0.1912
MSinceMostRecentInqexcl7days	0.1848
NumSatisfactoryTrades	0.1782
NumTradesOpeninLast12M	0.1751
NumBank2NatlTradesWHighUtilization	0.1710
NetFractionRevolvingBurden	0.1629
PercentTradesWBalance	0.1235
PercentInstallTrades	0.1165
MSinceOldestTradeOpen	0.0999

Figure 10. Model-Based Variable Importance Table

Partial Dependence Plots

In Model Studio, by default partial dependence plots are generated for the top five variables in the variable importance table. Figure 11 and Figure 12 show the partial dependency plots for the top two variables. In Figure 11, you can see that the predicted probability of payments being 90 days overdue decreases monotonically as the external risk estimate

value increases. A text box to the right of the graph explains the graph by using natural language generation (NLG) in SAS Viya. All model interpretability plots have NLG text boxes. These explanations can help you understand the graphs and are especially useful if you are not familiar with the graph type.

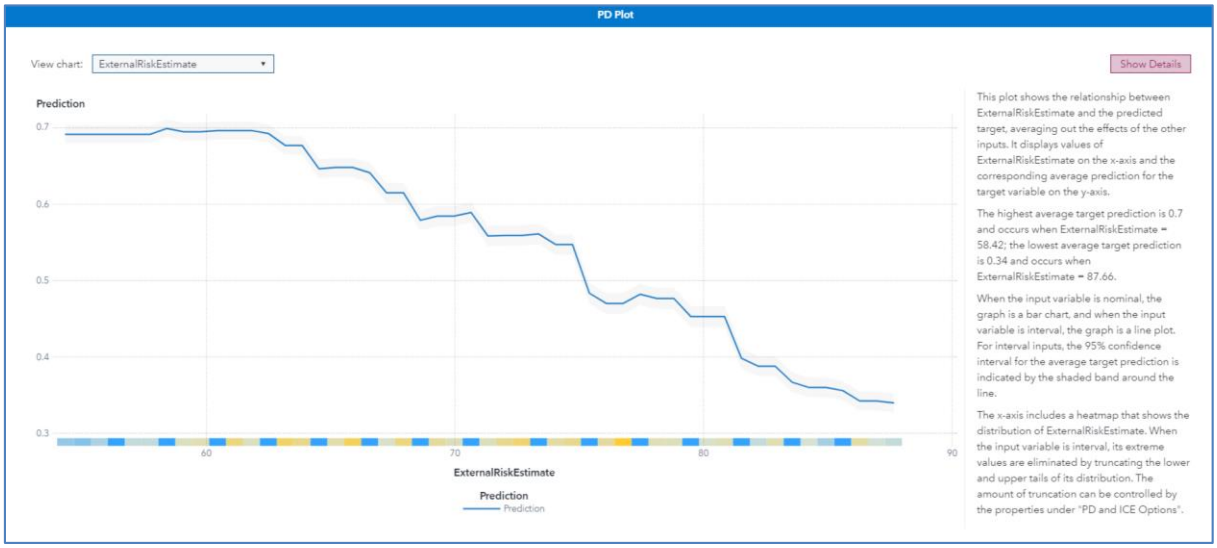

Figure 11. Partial Dependency Plot of External Risk Estimate

Figure 12 shows that the predicted probability of Bad payments decreases gradually as the applicant's number of months in file increases from 50 months to 100 months. This is expected, because applicants who have a longer credit history are deemed less risky. The heat map on the X axis shows that not many observations have an average number of months in file greater than 100. After the number of months in file reaches 100, the probability of Bad payments first increases slightly and then flattens because the model has less information in this domain. Hence, you should be cautious in explaining the part of the plot where the population density is low.

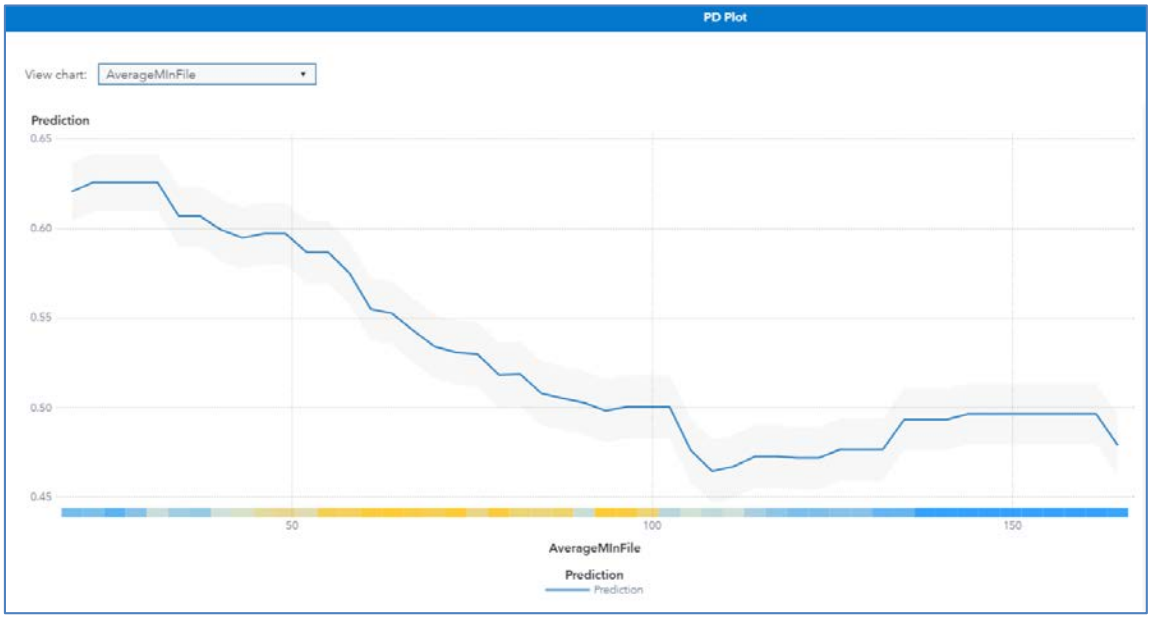

Figure 12. Partial Dependence Plot for Average Months in File

Local Interpretability

Local interpretability helps you understand individual predictions. Figure 13 shows the Gradient Boosting node's options for requesting local interpretability (ICE, LIME, and Kernel SHAP) for five applicants who are specified by their IDs. This identification variable should have unique values and must be specified to have the role Key (only one variable can have this role) on the Data tab.

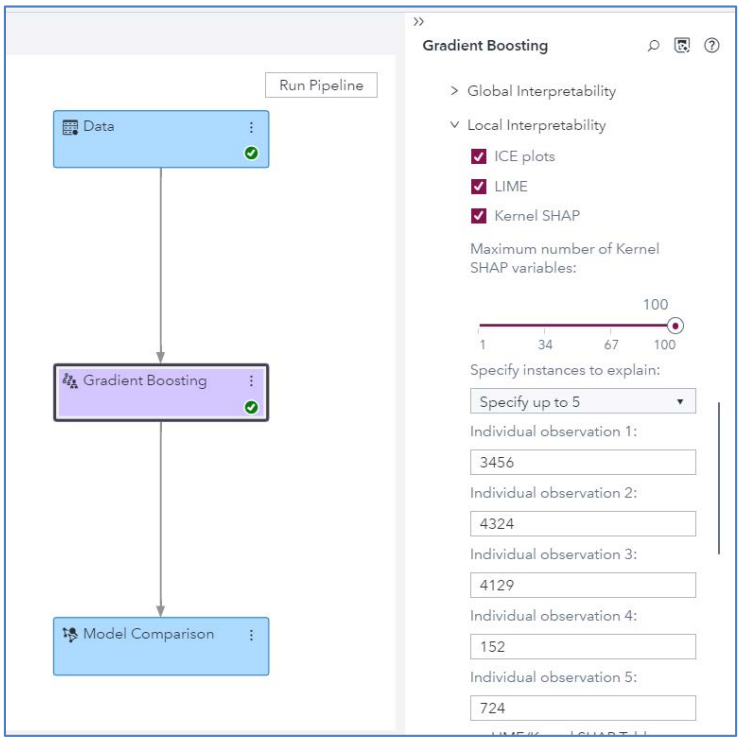

Figure 13. Requesting Local Interpretability Results for Individual Instances

ICE Plots

Figure 14 shows the five ICE plots for the specified observations for the external risk estimate input variable. The change in the model's prediction for each of these observations decreases as the external risk estimate increases, which matches the behavior that is seen in the partial dependence curve (shown in blue). Each observation is affected by the external risk estimate slightly differently. For observation 152, there is a steep decline in the model's predicted probability of late payment when the external risk estimate is between 60 and 70, whereas for observation for 4129, the decline is more gradual between 60 and 70 and is steeper after 70.

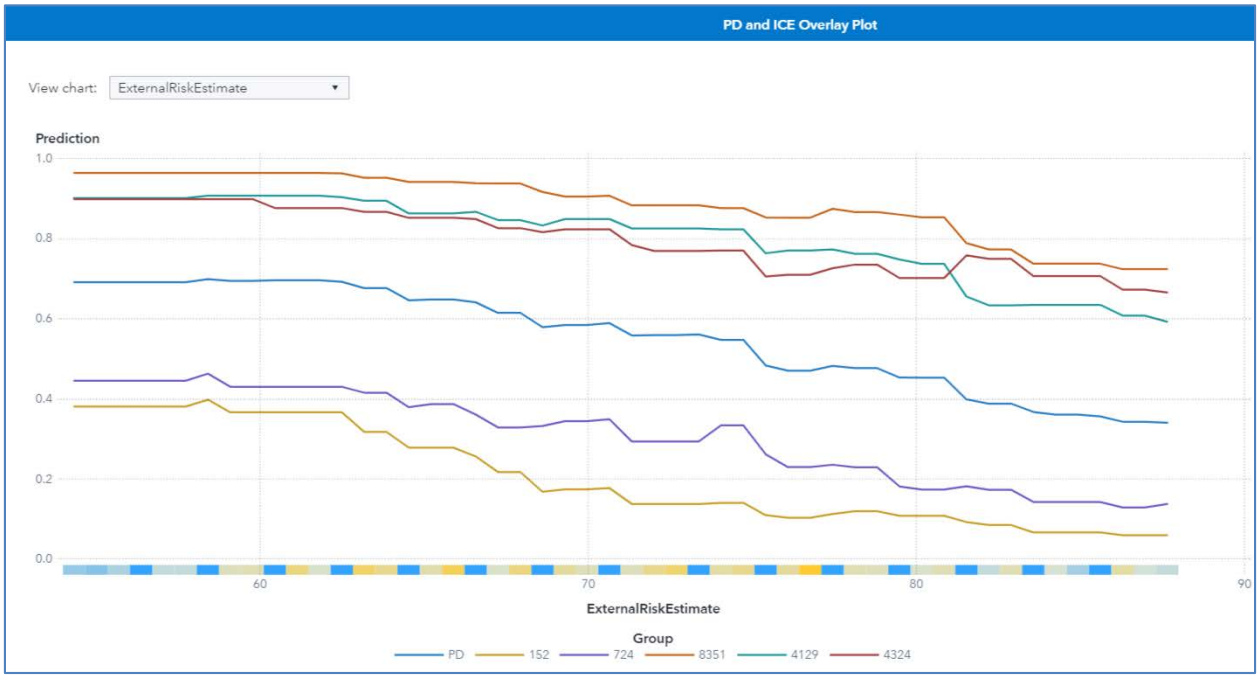

Figure 14. PD and ICE Overlay Plot

True Positive Prediction of High-Risk (LIME Explanation)

Figure 15 shows the LIME explanation of the prediction of the black-box gradient boosting model for instance 8351. Gradient boosting models predict this instance as a high-risk HELOC application with a predicted probability of 0.965.

When LIME is implemented, a local model is fit after converting each feature into a binary feature according to its proximity to the explained observation. Therefore, the coefficients in the LIME explanation represent the impact of the observed feature values of the instance. The feature values of instance 8351 are shown in Table 1.

Variable Label	Value
Number of Trades Open in Last 12 Months	0
External Risk Estimate	59
Months Since Most Recent Delinquency	0
Average Months in File	56
Months Since Most Recent Trade Open	15
Number Trades 60+ Ever	1
Percent Trades with Balance	59
Months Since Oldest Trade Open	136
Max Delinquency Ever	Never Delinquent

Table 1. Feature Values for LIME Explanation of Instance 8351

The LIME explanation for instance 8351 shows that Number Trades 60+ Ever = 1 and Max Delinquency Ever = Never Delinquent decrease the risk of default, whereas all other predictors increase the risk of default.

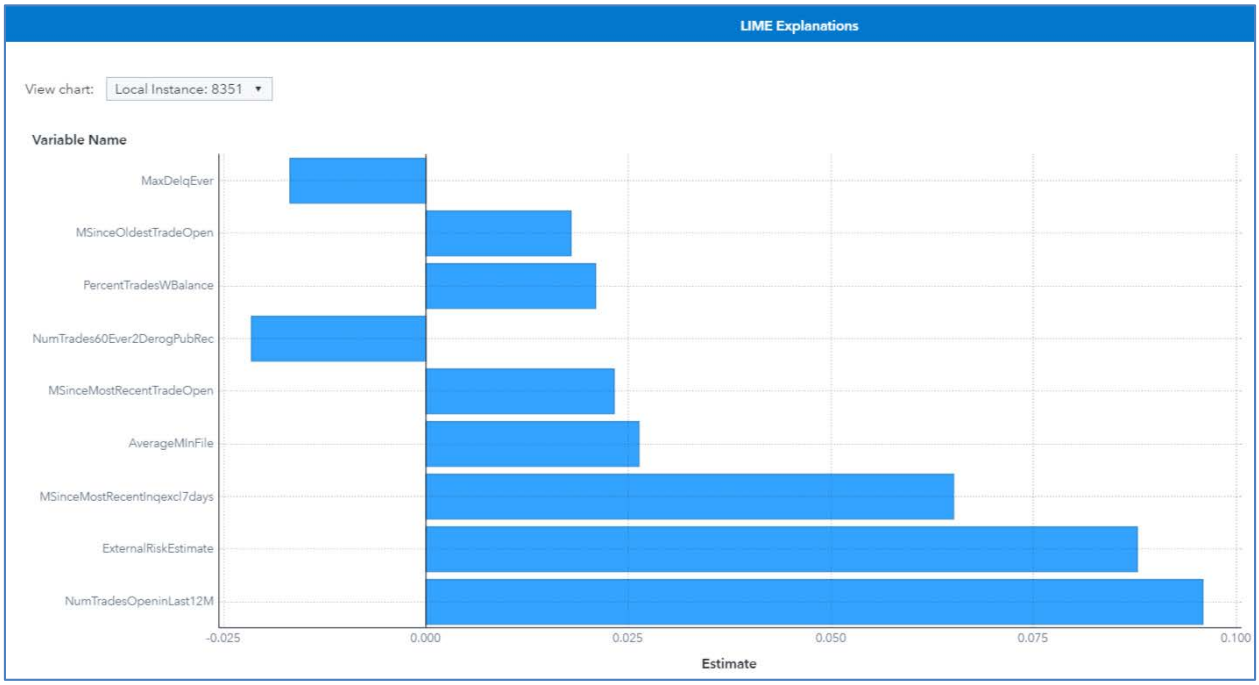

Figure 15. LIME Explanation of Instance 8351 as True Positive of High-Risk Application

False Positive Prediction of High Risk (Kernel SHAP Explanation)

Figure 16 shows the Kernel SHAP explanation of the prediction of the black-box gradient boosting model for instance 4129. The ground truth for this instance is Good, but the models outputs a high probability (0.904) of predicting it as Bad.

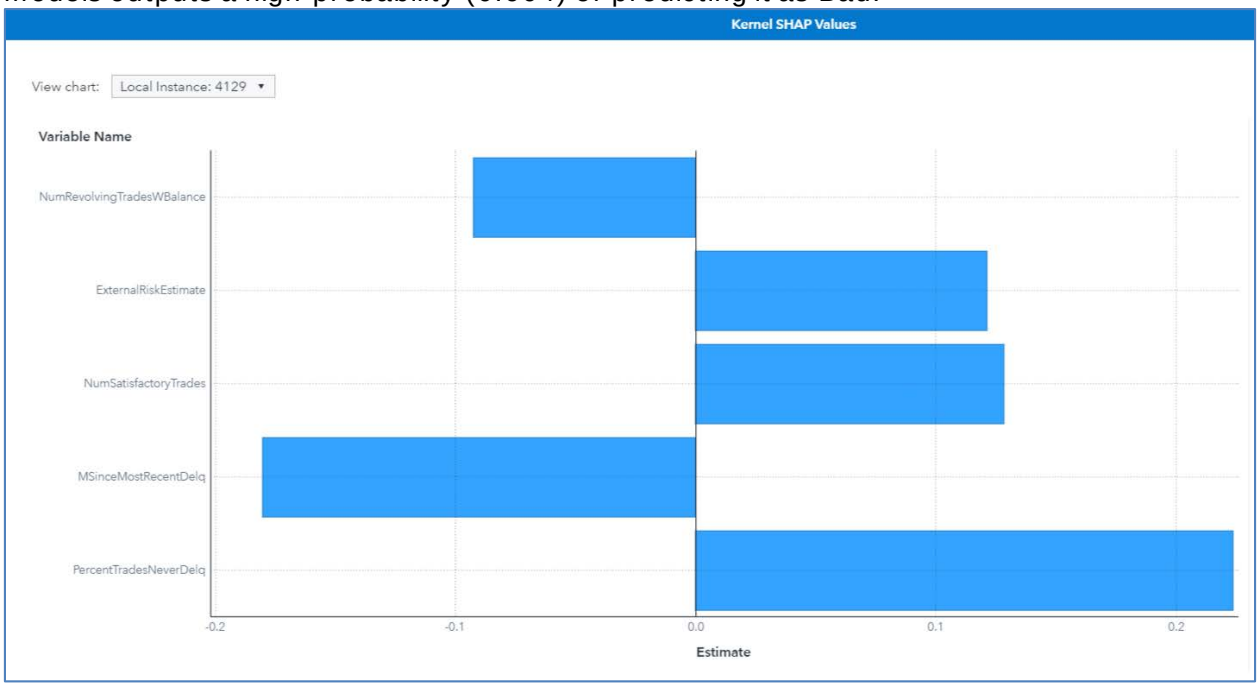

Figure 16. Kernel SHAP Explanation of Instance 4129 as False Positive of High-Risk Application

Variable Label	Value
Percent Trades Never Delinquent	67
Months Since Most Recent Delinquency	17
Number Satisfactory Trades	5
External Risk Estimate	62
Number Revolving Trades with Balance	3

Table 2. Feature Values for Kernel SHAP Explanation of Instance 4129

The Kernel SHAP explanation in Figure 16 and the feature values in Table 1 show that the features that contribute most toward increasing the high risk of late payments are Percent Trades Never Delinquent=67, Number Satisfactory Trades=5, and External Risk Estimate=62. Even though the model has such a high confidence in its prediction, the same confidence is not seen by examining the top five Kernel SHAP explanations, which can be used as a warning sign for this false prediction.

ACCESSING INTERPRETABILITY THROUGH PROGRAMMATIC USAGE

SAS Viya uses SAS® Cloud Analytic Services (CAS) to perform tasks. The smallest unit of work for the CAS server is a CAS action. SAS Model Studio nodes run CAS actions behind the scenes to perform analytical tasks. The explainModel action set provides post-hoc model-agnostic interpretations for machine learning models. For greater flexibility in how you use model interpretability, you might prefer to directly access the CAS actions through a programming interface.

In SAS Viya, you can run CAS actions via a variety of interfaces, including the following:

- SAS session, which uses the CAS procedure
- Python, Lua, or R, which use the SAS Scripting Wrapper for Analytics Transfer (SWAT) libraries
- Java, which uses the CAS Client class
- Representational state transfer (REST), which uses the CAS REST APIs

The actions in the explainModel action set can be used to explain a model whose score code is saved in SAS DATA step code or a SAS analytic store. The action set includes three actions:

- The partialDependence action calculates partial dependence (PD) and individual conditional expectation (ICE), which explain model behaviors on a variable-by-variable basis.
- The linearExplainer action also provides both local and global explanations. The local explanations (Shapley value estimates and LIME values) provide information about variable influence and local model behavior for an individual observation, and the global explanations (global regression) shed light on the overall model behavior by fitting a global surrogate regression model. The Shapley value estimator in the linearExplainer action uses the Kernel SHAP method.
- The shapleyExplainer action is a specialized Shapley value explainer that provides scalable and accurate Shapley value estimates. The Shapley values reflect the contribution of each variable toward the final prediction for an individual observation. The shapleyExplainer action uses the HyperSHAP method to calculate the Shapley values.

For implementation details about these techniques and user options, see the chapter

"Explain Model Action Set" in SAS Visual Data Mining and Machine Learning 8.5: Programming Guide.

CASE STUDY II: BREAST CANCER DATA

This section uses a health-care application to demonstrate how to access interpretability through a programmatic interface. The full SAS Viya code for this example can be found at: https://github.com/sassoftware/sas-viya-machine-learning/tree/master/interpretability

Data and Model

Diagnosing illness is a frequent and difficult task undertaken by doctors in the health-care industry. The consequences for being incorrect about a diagnosis can be severe. To mitigate some of the resulting pressure on doctors, many studies attempt to use machine learning to assist the process of diagnosis by training a model on historical clinical cases. Unfortunately, many of the most complex machine learning models cannot participate in some diagnostic procedures because they lack adequate interpretations.

This section uses the interpretability actions available in SAS Viya to train a model that predicts the malignancy of potential breast cancer biopsies and to explain the predictions of this model. Data for this section are from the University of Wisconsin Hospitals in Madison, WI, from Dr. William H. Wolberg, as described by Mangasarian and Wolberg (1990). The data contain nine observed quantities that are calculated by a physician upon the collection of a fine needle aspirate (FNA) from a potentially malignant area in a patient and are labeled according to their malignancy.

The data contain nine input variables and a target variable. The input variables are clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses. Each of these variables ranges from 1 to 10, with larger numbers generally indicating a greater likelihood of malignancy according to the examining physician. The target variable indicates whether the region was malignant or benign. None of these input variables by itself is enough to determine whether the region is malignant, so they must be aggregated in some way. A random forest model is trained to predict sample malignancy from these variables.

The data contain 699 observations. The bare nuclei variable contains a few instances of missing data, which were imputed to have value 0. The data were partitioned into a training set (70%) and a test set (30%). The following code trains a random forest model on the training data by using the decisionTree action set in SAS Visual Data Mining and Machine Learning:

```
proc cas;

   inputs = &inputs;

   decisionTree.forestTrain   result    = forest_res
                            / table     = "BREAST_CANCER_TRAIN"
                              target    = "class"
                              inputs    = inputs
                              oob       = True
                              nTree     = 500
                              maxLevel  = 12
                              prune     = True
                              seed      = 1234
                              varImp    = True
                              casOut    = {name    = "FOREST_MODEL_TABLE",
                                           replace = True}
                              savestate = {name    = "FOREST_MODEL",
```

```
                                                    replace = True};
    run;

    print forest_res;
    run;

quit;
```

The macro variable *&inputs* defines a list that contains the names of the input variables (except the ID variable). Following this training, a cutoff of 0.4 optimized the model's misclassification rate on the training set. This cutoff was then used to assess the model on the test set, which achieved a misclassification rate of 2.91%. This accuracy is competitive with other black-box models that are trained and published on the same data. Although the predictive accuracy is good, the forest model remains difficult to interpret because of its complexity. This complexity can render the model entirely useless in clinical settings, where interpretability is paramount for a patient's well-being and the physician's confidence.

Variable Importance

Table 2 contains the model-specific variable importance of the input variables as calculated by the forestTrain action.

Variable Name	Importance
cell_size_uniformity	36.9120
cell_shape_uniformity	28.3112
bare_nuclei	23.0752
bland_chromatin	15.5280
clump_thickness	9.8553
single_cell_size	5.7949
normal_nucleoli	5.0672
marginal_adhesion	3.5239
mitoses	0.2673

Table 2. ForestTrain Action Variable Importance Table

Table 2 shows that the five most important variables for the model's predictions are the uniformity of cell size and shape, bare nuclei, bland chromatin, and clump thickness. As you can see, this table does not tell you the direct effect of these variables in the model; it only says which variables were important in the model's construction. However, you can use this information to decide which variables to investigate further through other interpretability techniques such as PD and ICE plots.

The preceding information establishes that the forest model is highly predictive on unseen data and that not all the input variables are equally important. One remaining question is how each of the input variables affects the prediction of the model. The partial dependence plots for a particular variable illustrate how the model's predictions change when that variable's value fluctuates with all other variables being held constant.

The following CAS action call requests the partial dependence plot for the most important variable (cell size uniformity) of the variable importance table:

```
proc cas;

    /* Inputs and Nominals macro -> CASL Var */
    inputs = &inputs;
```

```
    /* Action Call */
    explainModel.partialDependence   result           = pd_res
                                   / table            = "BREAST_CANCER"
                                     inputs           = inputs
                                     modelTable       = "&model"
                                     modelTableType   = "ASTORE"
                                     predictedTarget  = "&pred_target"
                                     analysisVariable =
                                         {name  = "&var_name",
                                          nBins = 50}
                                     iceTable         =
                                         {casout   = {name = "ICE_TABLE",
                                                      replace = True},
                                          copyVars = "ALL"}
                                     seed             = 1234
    ;
    run;

    /* Save PD Results */
    saveresult pd_res["PartialDependence"] dataset = PD_RES;
    run;

quit;
```

Again, the *&inputs* macro variable is used to specify the list of input variables to the partialDependence action. The *modelTable, modelTableType,* and *predictedTarget* parameters specify the model being explained and its expected output column. The *table* and *iceTable* parameters indicate which data sets to use for the PD calculation and for storing replicated observations, respectively. The variable being explained is specified in the *analysisVariable* parameter, which contains other subparameters for slightly altering the produced PD calculation. For more information, see the chapter "Explain Model Action Set" in SAS Visual Data Mining and Machine Learning 8.5: Programming Guide.

Figure 17 shows the partial dependence of the forest's predictions with respect to the cell size uniformity variable.

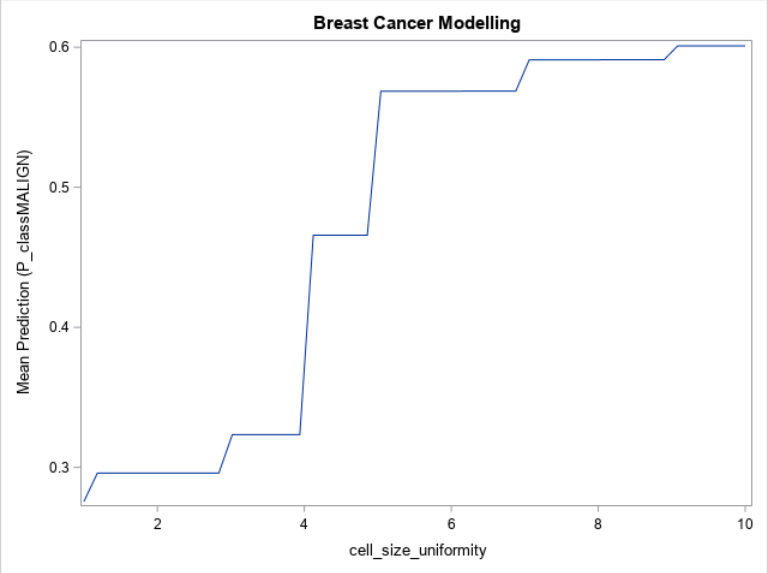

Figure 17. Partial Dependence of Forest Prediction of Malignancy with Respect to Cell Size Uniformity

Figure 17 shows that the model's predictions change greatly between the extremes of the cell size uniformity variable—for lower values of cell size uniformity, the model's average prediction is lower than 0.3, and for larger values it reaches 0.6. The value of this variable in the input data can have a large effect on the prediction from the model, which is why its variable importance metric is so high. Figure 17 also shows that the model's prediction with respect to cell size uniformity is monotonic, always increasing as the variable's value increases, which is as expected based on the description of the data. This plot builds trust in the model by demonstrating that it is not behaving in an unexpected way.

Typically, the partial dependencies of variables are shown on individual plots. However, since all the input variables in this problem lie on the same scale and have similar impact on the target (larger values indicate increased chance of malignancy), the partial dependencies with respect to each variable can be overlaid as shown in Figure 18.

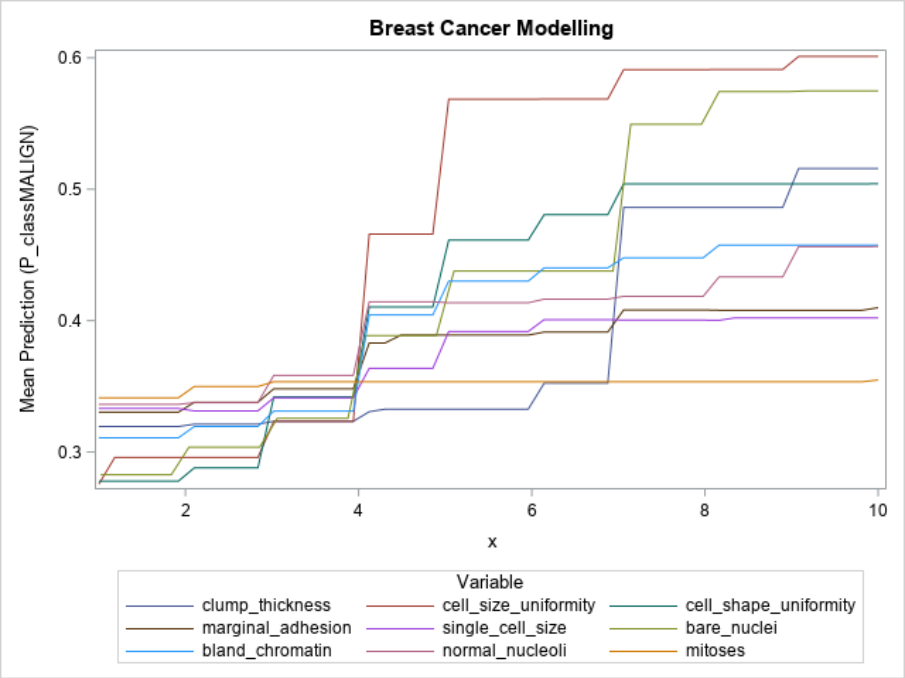

Figure 18. Partial Dependence of Forest Prediction of Malignancy for All Inputs

Figure 18 shows that the input variables are being used exactly as defined by the data description; the mean prediction of the model increases as the value of each of the input variables increases. Also depicted is the relative influence of each variable on the model's prediction, with cell size uniformity, cell shape uniformity, bare nuclei, clump thickness, and bland chromatin all showing a large range in their mean predictions. These large ranges correspond to their high variable importance values and build further trust in the model. The mitoses variable's partial dependence plot is almost perfectly flat, which would indicate that the variable contributes almost nothing to the prediction of the model.

LIME and HyperSHAP

The partial dependence plots are useful for understanding the effect of input variables on all observations. Sometimes, however, the role of a variable for an individual observation can be rather different from the role of that variable for the overall population. For example, for a particular patient, it would be useful to be able to determine which variables in an observation contribute most to the prediction of malignancy so that the patient could be further convinced of the need of a biopsy, a costly but necessary follow-up procedure. A reason such as "The thickness of the clumps in the FNA procedure lead us to believe that we should proceed with a biopsy" is more convincing to a patient than just saying "Looking at

your FNA, we think we should proceed with a biopsy." Explanations like these can be generated by using LIME and Shapley value methods.

The following code is used within a SAS macro to compute LIME coefficients:

```
explainModel.linearExplainer   result           = lex_res
                              / table            = "BREAST_CANCER_TRAIN"
                                query            =
                                   {name  = "BREAST_CANCER_TRAIN",
                                    where = "sample_id = &observation;"}
                                inputs           = inputs
                                modelTable       = "FOREST_MODEL"
                                modelTableType   = "ASTORE"
                                predictedTarget  = "P_classMALIGN"
                                preset           = "LIME"
                                explainer        =
                                   {standardizeEstimates = "INTERVALS",
                                    maxEffects           = &num_vars+1}
                                seed             = 1234
;
run;
```

In the code, the *preset* parameter is used to select the LIME method for generating explanations. The *table, modelTable, modelTableType,* and *predictedTarget* parameters are used in the same way as in the previous partialDependence action call; they specify the data and model to use. The &observation macro variable specifies which observation's prediction is being explained, and the &num_vars macro variable species how many input variables are being reported by LIME. The *standardizeEstimates* parameter is set to INTERVALS, which tells the action to standardize the least squares estimates of the LIME coefficients so that their magnitudes can be compared.

The following code is used to compute the Shapley values for explaining an individual prediction:

```
explainModel.shapleyExplainer   result           = shx_res
                               / table            = "BREAST_CANCER_TRAIN"
                                 query            =
                                    {name  = "BREAST_CANCER_TRAIN",
                                     where = "sample_id = &observation;"}
                                 inputs           = inputs
                                 modelTable       = "FOREST_MODEL"
                                 modelTableType   = "ASTORE"
                                 predictedTarget  = "P_classMALIGN"
;
run;
```

As in the linearExplainer action call, the *&observation* macro variable in the shapleyExplainer call is used to specify the observation to be explained. The shapleyExplainer action call reuses all previous parameters: *table, query, inputs, modelTable, modelTableType,* and *predictedTarget.* The values of the input variables for the specified observation are shown in Table 3. This observation was determined by the physician to be malignant. The input variables in this observation are large, which would correspond to a high likelihood of malignancy according to the attending physician. The forest model produces a predicted probability of 100% that this observation is malignant.

Clump Thickness	Cell Size Uniformity	Cell Shape Uniformity	Marginal Adhesion	Single Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli
8	10	10	8	7	10	9	7

Table 3. Correctly Predicted Malignant Observation from Training Data

Figure 19 shows the LIME coefficients for explaining the forest model for the observation that is shown in Table 3.

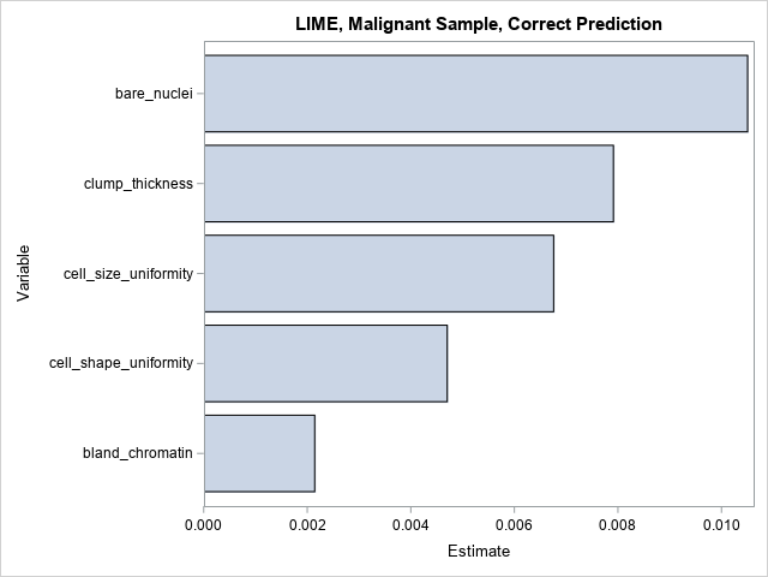

Figure 19. LIME Coefficients for Correctly Predicted Malignant Observation

The LIME values for the observation are all positive, indicating that the model's prediction increases as each variable value increases in the local region around this observation. This local explanation agrees with the global explanation that comes from the partial dependence plots, where the mean prediction increases along with each input variable.

Figure 20 shows the five largest Shapley values for explaining the prediction for the same observation that is shown in Table 3.

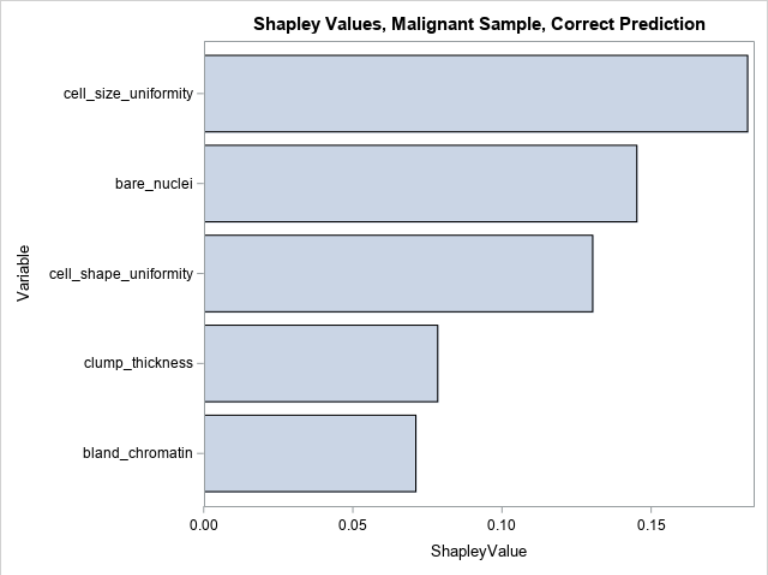

Figure 20. Shapley Values for Correctly Predicted Malignant Observation

The Shapley values for this observation are all positive, indicating that the values of the input variables to the model in this observation increase the model's prediction relative to other observations in the training data. This makes sense because the input variables in this observation are all high, taking values between 7 and 10, which would all indicate high likelihood for malignancy, and thus contribute positively to the model's prediction. You can see that LIME and Shapley explanations mostly agree for explaining the pretrained forest model's prediction for this observation.

Table 4 shows another sample from the data for which the model produced an incorrect prediction. Although the sample did prove to be malignant, the model predicts a likelihood of malignancy of 0.14, which is a large deviation from the truth. Most of the input variables in this observation take low values, meaning the investigating physician did not think any variable indicated a strong likelihood of malignancy. The LIME and Shapley values might provide insight into why the model produces this prediction.

Clump Thickness	Cell Size Uniformity	Cell Shape Uniformity	Marginal Adhesion	Single Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli
4	1	1	3	1	5	2	1

Table 4. Incorrectly Predicted Malignant Observation from Training Data

Figure 21 shows the LIME coefficients for explaining the forest model's false prediction.

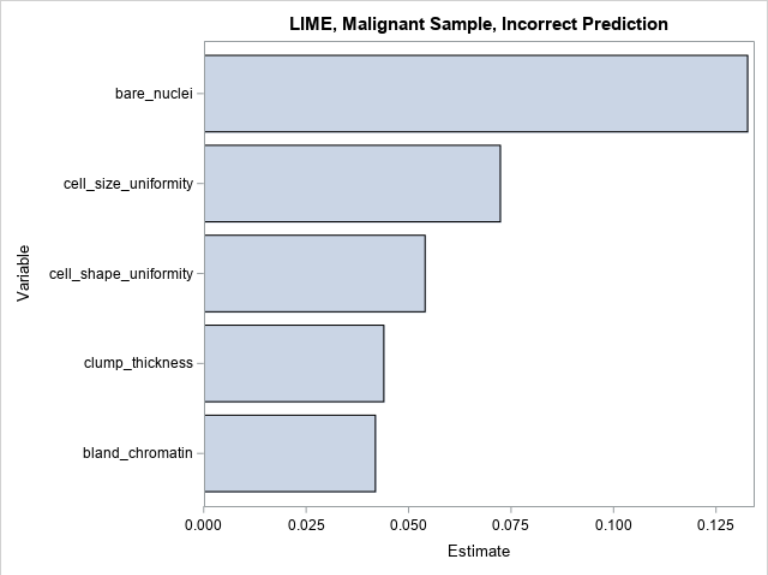

Figure 21. LIME Coefficients for Incorrectly Predicted Malignant Observation from Training Data

The LIME coefficients offer little insight beyond what is already understood about the model—that its prediction increases with respect to each increasing input variable. Many of the variables' values are small in this observation, and it would therefore be expected that the model's prediction would also be small.

Figure 22 shows the Shapley values for the same observation that is shown in Table 4.

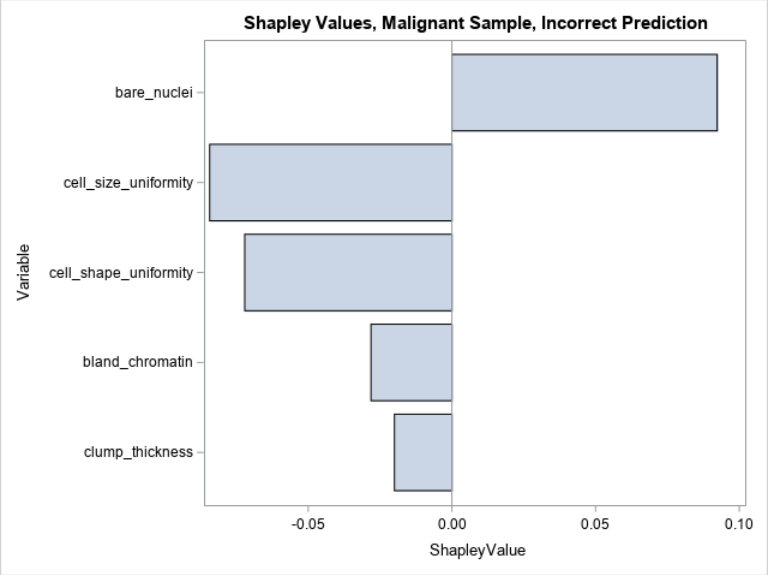

Figure 22. Shapley Values for Incorrectly Predicted Malignant Observation from Training Data

The Shapley values for the observation are more meaningful in the context of the incorrect prediction. It appears that the only variable that contributes positively to the prediction of malignancy is the bare nuclei variable (which takes an intermediate value of 5) and that all other input values cause the model's prediction to decrease with respect to the other observations. It seems that the model is split in determining whether this observation is malignant. This can be caused by the model giving too much weight to some variables or there not being enough information in the input data to model this observation. Ultimately

these results can identify a weakness in the modeling process or might indicate that this particular instance is a really hard to diagnose on the basis of the available input variables. This information can be used to inform further data collection, feature engineering, and model tuning.

Refining the Model

Model interpretability does not necessarily need to be confined to the end of a modeling process. Occasionally, the interpretability results can reveal information that leads to new feature engineering ideas or reveals that certain input variables are useless to the models and only contribute to the curse of dimensionality. For this data set, the partial dependency of the input variables increases monotonically. The simple nature of the relationship between the input variables and the model's prediction might lead you to think that a simpler model will perform just as well as the forest model for this problem. Furthermore, the mitoses variable seems to be effectively unused by the model according to both the variable importance table and the partial dependence plots, which means it can likely be dropped entirely from the input data.

With the preceding information in mind, a logistic regression model is trained on the same data, dropping the mitoses input. Backwards selection is done using the logistic action in the regression action set. Only the clump thickness, cell size uniformity, and bare nuclei variables remain after selection. Based on the training data, a cutoff of 0.19 is selected, which yields a misclassification rate of 4.37% on the test set, a mere 1.46 % decrease in accuracy from the forest model. Figure 23 shows the partial dependence of each input variable with respect to the mean prediction from the logistic model.

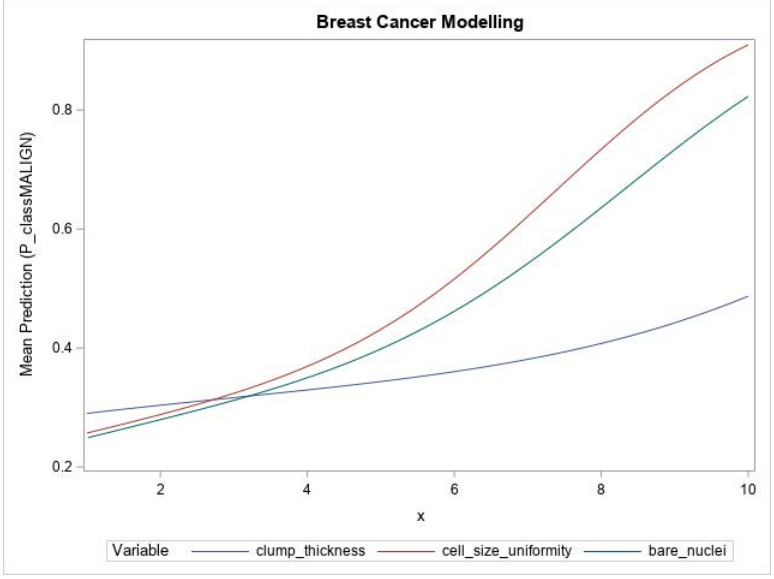

Figure 23. Partial Dependence of Logistic Regression Prediction of Malignancy for All Inputs

The partial dependence curves of the input variables for the logistic regression model show a similar relationship to what they show in the forest model, with the mean prediction increasing as the variable values increase. However, as expected, the logistic curves are much smoother than those of the forest model.

Now you have two models of comparable accuracy, each with its drawbacks. The forest model demonstrates a higher accuracy than the regression model, but it is natively uninterpretable. The logistic regression enables you to directly use the regression coefficients to understand the model, but it has a slightly lower accuracy. Ultimately the best model to use is the one that maximizes prediction accuracy while meeting the

necessary interpretability standard. If using the LIME, Shapley, and partial dependence values for interpretations provides meaningful explanations given the modeling context, then the forest model is better. If not, then the logistic regression should be chosen. Since this model will ultimately be consumed by a clinician who has worked closely with the patient and directly developed the input features, the forest model is likely a better choice, because the clinician is there to safeguard against model inaccuracies.

CONCLUSION

Model-agnostic global and local model interpretability techniques are presented with two case studies in SAS Visual Data Mining and Machine Learning. This paper shows you how you can use these valuable techniques to explain, validate, and debug your black-box machine learning models.

REFERENCES

Altmann, A., Tolosi, L., Sander, O., and Lengauer, T. 2010. "Permutation Importance: A Corrected Feature Importance Measure." *Bioinformatics*, 26. 1340–7.

Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., and Elhadad, N. 2015. "Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission." *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1721–1730.

Friedman, J. H. 2001. "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, 29, pp. 1189–1232.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. 2015. "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation." *Journal of Computational and Graphical Statistics,* 24.1:44–65.

Koch, P., Wujek, W., Golovidov. O., and Gardner., S. 2017. "Automated Hyperparameter Tuning for Effective Machine Learning." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc.

Kononenko, I. 2010. "An Efficient Explanation of Individual Classifications Using Game Theory." *Journal of Machine Learning Research,* 11. Jan:1–18.

Lundberg, S. M., and Lee, S. 2017. "A Unified Approach to Interpreting Model Predictions." *Advances in Neural Information Processing Systems*.

Mangasarian, O. L., and Wolberg, W. H. 1990. "Cancer Diagnosis via Linear Programming." *SIAM News*, 23.5:1–18

Molnar, C. 2019. "Interpretable Machine Learning. A Guide for Making Black Box Models Explainable." Available at https://christophm.github.io/interpretable-ml-book/.

Ribeiro, M. T., Singh, S., Guestrin, C. 2016. "Why Should I Trust You? Explaining the Predictions of Any Classifier." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Ustun, B., and Rudin, C. 2015. "Supersparse Linear Integer Models for Optimized Medical Scoring Systems." *Machine Learning* 102.3: 349–391.

Wright, R. 2018. "Interpreting Black-Box Machine Learning Models Using Partial Dependence and Individual Conditional Expectation Plots." *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the authors:

Funda Güneş
SAS Institute Inc.
funda.gunes@sas.com

Ricky Tharrington
SAS Institute Inc.
ricky.tharrington@sas.com

Ralph Abbey
SAS Institute Inc.
ralph.abbey@sas.com

Xin Hunt
SAS Institute Inc.
xin.hunt@sas.com