
SHAP Documentation

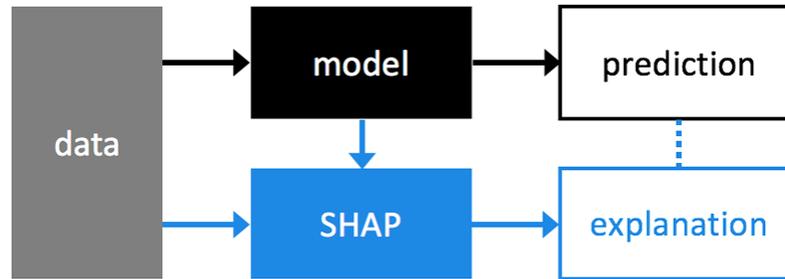
Release latest

Scott Lundberg

Nov 12, 2019

Contents

1 Plots	5
Index	7



SHAP (SHapley Additive exPlanations) is a unified approach to explain the output of any machine learning model. SHAP connects game theory with local explanations, uniting several previous methods and representing the only possible consistent and locally accurate additive feature attribution method based on expectations (see the SHAP NIPS paper for details).

```
class shap.TreeExplainer(model, data=None, model_output='margin',
                          feature_perturbation='interventional', **deprecated_options)
```

Uses Tree SHAP algorithms to explain the output of ensemble tree models.

Tree SHAP is a fast and exact method to estimate SHAP values for tree models and ensembles of trees, under several different possible assumptions about feature dependence. It depends on fast C++ implementations either inside an external model package or in the local compiled C extension.

model [model object] The tree based machine learning model that we want to explain. XGBoost, LightGBM, CatBoost, Pyspark and most tree-based scikit-learn models are supported.

data [numpy.array or pandas.DataFrame] The background dataset to use for integrating out features. This argument is optional when `feature_perturbation="tree_path_dependent"`, since in that case we can use the number of training samples that went down each tree path as our background dataset (this is recorded in the model object).

feature_perturbation ["interventional" (default) or "tree_path_dependent" (default when `data=None`)] Since SHAP values rely on conditional expectations we need to decide how to handle correlated (or otherwise dependent) input features. The "interventional" approach breaks the dependencies between features according to the rules dictated by casual inference (Janzing et al. 2019). Note that the "interventional" option requires a background dataset and its runtime scales linearly with the size of the background dataset you use. Anywhere from 100 to 1000 random background samples are good sizes to use. The "tree_path_dependent" approach is to just follow the trees and use the number of training examples that went down each leaf to represent the background distribution. This approach does not require a background dataset and so is used by default when no background dataset is provided.

model_output ["margin", "probability", or "logloss"] What output of the model should be explained. If "margin" then we explain the raw output of the trees, which varies by model (for binary classification in XGBoost this is the log odds ratio). If "probability" then we explain the output of the model transformed into probability space (note that this means the SHAP values now sum to the probability output of the model). If "logloss" then we explain the log base e of the model loss function, so that the SHAP values sum up to the log loss of the model for each sample. This is helpful for breaking down model performance by feature. Currently the probability and logloss options are only supported when `feature_dependence="independent"`.

```
shap_interaction_values(X, y=None, tree_limit=None)
```

Estimate the SHAP interaction values for a set of samples.

X [numpy.array, pandas.DataFrame or catboost.Pool (for catboost)] A matrix of samples (# samples x # features) on which to explain the model's output.

y [numpy.array] An array of label values for each sample. Used when explaining loss functions (not yet supported).

tree_limit [None (default) or int] Limit the number of trees used by the model. By default None means no use the limit of the original model, and -1 means no limit.

For models with a single output this returns a tensor of SHAP values (# samples x # features x # features). The matrix (# features x # features) for each sample sums to the difference between the model output for that sample and the expected value of the model output (which is stored in the `expected_value` attribute of the explainer). Each row of this matrix sums to the SHAP value for that feature for that sample. The diagonal entries of the matrix represent the “main effect” of that feature on the prediction and the symmetric off-diagonal entries represent the interaction effects between all pairs of features for that sample. For models with vector outputs this returns a list of tensors, one for each output.

shap_values (*X*, *y=None*, *tree_limit=None*, *approximate=False*, *check_additivity=True*)
Estimate the SHAP values for a set of samples.

X [numpy.array, pandas.DataFrame or catboost.Pool (for catboost)] A matrix of samples (# samples x # features) on which to explain the model’s output.

y [numpy.array] An array of label values for each sample. Used when explaining loss functions.

tree_limit [None (default) or int] Limit the number of trees used by the model. By default None means no use the limit of the original model, and -1 means no limit.

approximate [bool] Run fast, but only roughly approximate the Tree SHAP values. This runs a method previously proposed by Saabas which only considers a single feature ordering. Take care since this does not have the consistency guarantees of Shapley values and places too much weight on lower splits in the tree.

check_additivity [bool] Run a validation check that the sum of the SHAP values equals the output of the model. This check takes only a small amount of time, and will catch potential unforeseen errors. Note that this check only runs right now when explaining the margin of the model.

For models with a single output this returns a matrix of SHAP values (# samples x # features). Each row sums to the difference between the model output for that sample and the expected value of the model output (which is stored in the `expected_value` attribute of the explainer when it is constant). For models with vector outputs this returns a list of such matrices, one for each output.

class `shap.KernelExplainer` (*model*, *data*, *link=<shap.common.IdentityLink object>*, ***kwargs*)
Uses the Kernel SHAP method to explain the output of any function.

Kernel SHAP is a method that uses a special weighted linear regression to compute the importance of each feature. The computed importance values are Shapley values from game theory and also coefficients from a local linear regression.

model [function or `iml.Model`] User supplied function that takes a matrix of samples (# samples x # features) and computes the output of the model for those samples. The output can be a vector (# samples) or a matrix (# samples x # model outputs).

data [numpy.array or pandas.DataFrame or `shap.common.DenseData` or any `scipy.sparse` matrix] The background dataset to use for integrating out features. To determine the impact of a feature, that feature is set to “missing” and the change in the model output is observed. Since most models aren’t designed to handle arbitrary missing data at test time, we simulate “missing” by replacing the feature with the values it takes in the background dataset. So if the background dataset is a simple sample of all zeros, then we would approximate a feature being missing by setting it to zero. For small problems this background dataset can be the whole training set, but for larger problems consider using a single reference value or using the `kmeans` function to summarize the dataset. Note: for sparse case we accept any sparse matrix but convert to `lil` format for performance.

link [“identity” or “logit”] A generalized linear model link to connect the feature importance values to the model output. Since the feature importance values, ϕ_i , sum up to the model output, it often makes sense

to connect them to the output with a link function where $\text{link}(\text{outout}) = \text{sum}(\phi)$. If the model output is a probability then the LogitLink link function makes the feature importance values have log-odds units.

shap_values (*X*, ***kwargs*)

Estimate the SHAP values for a set of samples.

X [numpy.array or pandas.DataFrame or any scipy.sparse matrix] A matrix of samples (# samples x # features) on which to explain the model's output.

nsamples ["auto" or int] Number of times to re-evaluate the model when explaining each prediction. More samples lead to lower variance estimates of the SHAP values. The "auto" setting uses $nsamples = 2 * X.shape[1] + 2048$.

l1_reg ["num_features(int)", "auto" (default for now, but deprecated), "aic", "bic", or float] The l1 regularization to use for feature selection (the estimation procedure is based on a debiased lasso). The auto option currently uses "aic" when less than 20% of the possible sample space is enumerated, otherwise it uses no regularization. THE BEHAVIOR OF "auto" WILL CHANGE in a future version to be based on num_features instead of AIC. The "aic" and "bic" options use the AIC and BIC rules for regularization. Using "num_features(int)" selects a fix number of top features. Passing a float directly sets the "alpha" parameter of the sklearn.linear_model.Lasso model used for feature selection.

For models with a single output this returns a matrix of SHAP values (# samples x # features). Each row sums to the difference between the model output for that sample and the expected value of the model output (which is stored as expected_value attribute of the explainer). For models with vector outputs this returns a list of such matrices, one for each output.

class shap.DeepExplainer (*model*, *data*, *session=None*, *learning_phase_flags=None*)

Meant to approximate SHAP values for deep learning models.

This is an enhanced version of the DeepLIFT algorithm (Deep SHAP) where, similar to Kernel SHAP, we approximate the conditional expectations of SHAP values using a selection of background samples. Lundberg and Lee, NIPS 2017 showed that the per node attribution rules in DeepLIFT (Shrikumar, Greenside, and Kundaje, arXiv 2017) can be chosen to approximate Shapley values. By integrating over many background samples DeepExplainer estimates approximate SHAP values such that they sum up to the difference between the expected model output on the passed background samples and the current model output ($f(x) - E[f(x)]$).

shap_values (*X*, *ranked_outputs=None*, *output_rank_order='max'*)

Return approximate SHAP values for the model applied to the data given by X.

X [list,] if framework == 'tensorflow': numpy.array, or pandas.DataFrame if framework == 'pytorch': torch.tensor A tensor (or list of tensors) of samples (where $X.shape[0] == \# \text{ samples}$) on which to explain the model's output.

ranked_outputs [None or int] If ranked_outputs is None then we explain all the outputs in a multi-output model. If ranked_outputs is a positive integer then we only explain that many of the top model outputs (where "top" is determined by output_rank_order). Note that this causes a pair of values to be returned (shap_values, indexes), where shap_values is a list of numpy arrays for each of the output ranks, and indexes is a matrix that indicates for each sample which output indexes were chosen as "top".

output_rank_order ["max", "min", or "max_abs"] How to order the model outputs when using ranked_outputs, either by maximum, minimum, or maximum absolute value.

For a models with a single output this returns a tensor of SHAP values with the same shape as X. For a model with multiple outputs this returns a list of SHAP value tensors, each of which are the same shape as X. If ranked_outputs is None then this list of tensors matches the number of model outputs. If ranked_outputs is a positive integer a pair is returned (shap_values, indexes), where shap_values is a list of tensors with a length of ranked_outputs, and indexes is a matrix that indicates for each sample which output indexes were chosen as "top".

`shap.summary_plot` (*shap_values*, *features=None*, *feature_names=None*, *max_display=None*, *plot_type=None*, *color=None*, *axis_color='#333333'*, *title=None*, *alpha=1*, *show=True*, *sort=True*, *color_bar=True*, *plot_size='auto'*, *layered_violin_max_num_bins=20*, *class_names=None*, *color_bar_label='Feature value'*, *auto_size_plot=None*)

Create a SHAP summary plot, colored by feature values when they are provided.

shap_values [numpy.array] For single output explanations this is a matrix of SHAP values (# samples x # features). For multi-output explanations this is a list of such matrices of SHAP values.

features [numpy.array or pandas.DataFrame or list] Matrix of feature values (# samples x # features) or a feature_names list as shorthand

feature_names [list] Names of the features (length # features)

max_display [int] How many top features to include in the plot (default is 20, or 7 for interaction plots)

plot_type [“dot” (default for single output), “bar” (default for multi-output), “violin”,] or “compact_dot”. What type of summary plot to produce. Note that “compact_dot” is only used for SHAP interaction values.

plot_size [“auto” (default), float, (float, float), or None] What size to make the plot. By default the size is auto-scaled based on the number of features that are being displayed. Passing a single float will cause each row to be that many inches high. Passing a pair of floats will scale the plot by that number of inches. If None is passed then the size of the current figure will be left unchanged.

`shap.dependence_plot` (*ind*, *shap_values*, *features*, *feature_names=None*, *display_features=None*, *interaction_index='auto'*, *color='#1E88E5'*, *axis_color='#333333'*, *cmap=None*, *dot_size=16*, *x_jitter=0*, *alpha=1*, *title=None*, *xmin=None*, *xmax=None*, *ax=None*, *show=True*)

Create a SHAP dependence plot, colored by an interaction feature.

Plots the value of the feature on the x-axis and the SHAP value of the same feature on the y-axis. This shows how the model depends on the given feature, and is like a richer extension of the classical partial dependence plots. Vertical dispersion of the data points represents interaction effects. Grey ticks along the y-axis are data points where the feature’s value was NaN.

ind [int or string] If this is an int it is the index of the feature to plot. If this is a string it is either the name of the feature to plot, or it can have the form “rank(int)” to specify the feature with that rank (ordered by mean absolute SHAP value over all the samples).

shap_values [numpy.array] Matrix of SHAP values (# samples x # features).

features [numpy.array or pandas.DataFrame] Matrix of feature values (# samples x # features).

feature_names [list] Names of the features (length # features).

display_features [numpy.array or pandas.DataFrame] Matrix of feature values for visual display (such as strings instead of coded values).

interaction_index [“auto”, None, int, or string] The index of the feature used to color the plot. The name of a feature can also be passed as a string. If “auto” then `shap.common.approximate_interactions` is used to pick what seems to be the strongest interaction (note that to find the true strongest interaction you need to compute the SHAP interaction values).

x_jitter [float (0 - 1)] Adds random jitter to feature values. May increase plot readability when feature is discrete.

alpha [float] The transparency of the data points (between 0 and 1). This can be useful to show density of the data points when using a large dataset.

xmin [float or string] Represents the lower bound of the plot’s x-axis. It can be a string of the format “percentile(float)” to denote that percentile of the feature’s value used on the x-axis.

xmax [float or string] Represents the upper bound of the plot’s x-axis. It can be a string of the format “percentile(float)” to denote that percentile of the feature’s value used on the x-axis.

ax [matplotlib Axes object] Optionally specify an existing matplotlib Axes object, into which the plot will be placed. In this case we do not create a Figure, otherwise we do.

`shap.force_plot` (*base_value*, *shap_values*, *features=None*, *feature_names=None*, *out_names=None*, *link='identity'*, *plot_cmap='RdBu'*, *matplotlib=False*, *show=True*, *figsize=(20, 3)*, *ordering_keys=None*, *ordering_keys_time_format=None*, *text_rotation=0*)

Visualize the given SHAP values with an additive force layout.

base_value [float] This is the reference value that the feature contributions start from. For SHAP values it should be the value of `explainer.expected_value`.

shap_values [numpy.array] Matrix of SHAP values (# features) or (# samples x # features). If this is a 1D array then a single force plot will be drawn, if it is a 2D array then a stacked force plot will be drawn.

features [numpy.array] Matrix of feature values (# features) or (# samples x # features). This provides the values of all the features, and should be the same shape as the `shap_values` argument.

feature_names [list] List of feature names (# features).

out_names [str] The name of the output of the model (plural to support multi-output plotting in the future).

link [“identity” or “logit”] The transformation used when drawing the tick mark labels. Using logit will change log-odds numbers into probabilities.

matplotlib [bool] Whether to use the default Javascript output, or the (less developed) matplotlib output. Using matplotlib can be helpful in scenarios where rendering Javascript/HTML is inconvenient.

`shap.image_plot` (*shap_values*, *x*, *labels=None*, *show=True*, *width=20*, *aspect=0.2*, *hspace=0.2*, *label-pad=None*)

Plots SHAP values for image inputs.

D

DeepExplainer (*class in shap*), 3
dependence_plot () (*in module shap*), 5

F

force_plot () (*in module shap*), 6

I

image_plot () (*in module shap*), 6

K

KernelExplainer (*class in shap*), 2

S

shap_interaction_values ()
 (*shap.TreeExplainer method*), 1
shap_values () (*shap.DeepExplainer method*), 3
shap_values () (*shap.KernelExplainer method*), 3
shap_values () (*shap.TreeExplainer method*), 2
summary_plot () (*in module shap*), 5

T

TreeExplainer (*class in shap*), 1