interpret-community

Release 0.28.0

Microsoft

Jan 04, 2023

CONTENTS

Interpret-Community is an experimental repository extending Interpret, with additional interpretability techniques and utility functions to handle real-world datasets and workflows for explaining models trained on **tabular data**. This repository contains the Interpret-Community SDK and Jupyter notebooks with examples to showcase its use.

The code is available from GitHub.

For the InterpretML organization, which interpret-community is part of, please see the common website at https://interpret.ml/.

The explanation dashboard, which is a web-based interface to the interpret-community code, is in the responsible-ai-toolbox repository.

You can find the main documentation for the all-in-one Responsible AI Toolbox which uses interpret-community as the backend at https://responsibleaitoolbox.ai/.

CHAPTER

ONE

INSTALLATION

The package can be installed from pypi with:

```
pip install interpret-community
```

1.1 Overview

Interpret-Community extends the Interpret repository and incorporates further community developed and experimental interpretability techniques and functionalities that are designed to enable interpretability for real world scenarios. Interpret-Community enables adding new experimental techniques (or functionalities) and performing comparative analysis to evaluate them.

Interpret-Community

1. Actively incorporates innovative experimental interpretability techniques and allows for further expansion by researchers and data scientists
2. Applies optimizations to make it possible to run interpretability techniques on real-world datasets at scale
3. Provides improvements such as the capability to "reverse the feature engineering pipeline" to provide model insights in terms of the original raw features rather than engineered features
4. Provides interactive and exploratory visualizations to empower data scientists to gain meaningful insight into their data

1.2 Getting Started

The package can be installed from pypi with:

```
pip install interpret-community
```

You can use Anaconda to simplify package and environment management.

To setup on your local machine:

1.3 Supported Models

This API supports models that are trained on datasets in Python *numpy.ndarray*, *pandas.DataFrame*, or *scipy.sparse.csr_matrix* format.

The explanation functions accept both models and pipelines as input as long as the model or pipeline implements a *predict* or *predict_proba* function that conforms to the Scikit convention. If not compatible, you can wrap your model's prediction function into a wrapper function that transforms the output into the format that is supported (predict or predict_proba of Scikit), and pass that wrapper function to your selected interpretability techniques.

If a pipeline script is provided, the explanation function assumes that the running pipeline script returns a prediction. The repository also supports models trained via **PyTorch**, **TensorFlow**, and **Keras** deep learning frameworks.

1.4 Supported Explainers

The following are a list of the explainers available in the community repository:

Table 1: Explainers

Interpretability Technique	Description	Type
SHAP Kernel Explainer	SHAP's Kernel explainer uses a specially weighted local linear regression to estimate SHAP values for **any model**.	Model-agnostic
GPU SHAP Kernel Explainer	GPU Kernel explainer uses cuML's GPU accelerated version of SHAP's Kernel Explainer to estimate SHAP values for **any model**. It's main advantage is to provide acceleration to fast GPU models, like those in cuML. But it can also be used with CPU-based models, where speedups can still be achieved but they might be limited due to data transfers and speed of models themselves.	Model-agnostic
SHAP Tree Explainer	SHAP's Tree explainer, which focuses on the polynomial time fast SHAP value estimation algorithm specific to **trees and ensembles of trees**.	Model-specific
SHAP Deep Explainer	Based on the explanation from SHAP, Deep Explainer "is a high-speed approximation algorithm for SHAP values in deep learning models that builds on a connection with DeepLIFT described in the SHAP NIPS paper. **TensorFlow** models and **Keras** models using the TensorFlow backend are supported (there is also preliminary support for PyTorch)".	Model-specific
SHAP Linear Explainer	SHAP's Linear explainer computes SHAP values for a **linear model**, optionally accounting for inter-feature correlations.	Model-specific
Mimic Explainer (Global Surrogate)	Mimic explainer is based on the idea of training global surrogate models to mimic blackbox models. A global surrogate model is an intrinsically interpretable model that is trained to approximate the predictions of **any black box model** as accurately as possible. Data scientists can interpret the surrogate model to draw conclusions about the black box model. You can use one of the following interpretable models as your surrogate model: LightGBM (LGBMExplainableModel), Linear Regression (LinearExplainableModel), Stochastic Gradient Descent explainable model (SGDExplainableModel), and Decision Tree (DecisionTreeExplainableModel).	Model-agnostic
Permutation Feature Importance Explainer (PFI)	Permutation Feature Importance is a technique used to explain classification and regression models that is inspired by Breiman's Random Forests paper (see section 10). At a high level, the way it works is by randomly shuffling data one feature at a time for the entire dataset and calculating how much the performance metric of interest changes. The larger the change, the more important that feature is. PFI can explain the overall behavior of **any underlying model** but does not explain individual predictions.	Model-agnostic
LIME Explainer	Local Interpretable Model-agnostic Explanations (LIME) is a local linear approximation of the model's behavior. The explainer wraps the LIME tabular explainer with a uniform API and additional functionality.	Model-agnostic

Besides the interpretability techniques described above, Interpret-Community supports another SHAP-based explainer, called *TabularExplainer*. Depending on the model, *TabularExplainer* uses one of the supported SHAP explainers:

Table 2: TabularExplainer

Original Model	Invoked Explainer
Tree-based models	SHAP TreeExplainer
Deep Neural Network models	SHAP DeepExplainer
Linear models	SHAP LinearExplainer
None of the above	SHAP KernelExplainer or GPUKernelExplainer

1.5 Example Notebooks

Please take a look at our example notebooks in the *notebooks/* directory:

- Blackbox interpretability for binary classification
- Blackbox interpretability for multi-class classification
- Blackbox interpretability for regression
- Blackbox interpretability with simple raw feature transformations
- Blackbox interpretability with advanced raw feature transformations
- Captum integration example
- Explain binary classification model predictions on GPU
- Explain regression model predictions using MimicExplainer

1.6 Use Interpret-Community

1.6.1 Interpretability in training

1. Train your model

```
# load breast cancer dataset, a well-known small dataset that comes with scikit-
↪learn
from sklearn.datasets import load_breast_cancer
from sklearn import svm
from sklearn.model_selection import train_test_split
breast_cancer_data = load_breast_cancer()
classes = breast_cancer_data.target_names.tolist()

# split data into train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(breast_cancer_data.data,
                                                    breast_cancer_data.target,
                                                    test_size=0.2,
                                                    random_state=0)
clf = svm.SVC(gamma=0.001, C=100., probability=True)
model = clf.fit(x_train, y_train)

# alternatively, a cuML estimator can be trained here for GPU model
# ensure RAPIDS is installed - refer to https://rapids.ai/ for more information
import cuml
from cuml.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(breast_cancer_data.data,
                                                    breast_cancer_data.target,
                                                    test_size=0.2,
                                                    random_state=0)
clf = cuml.svm.SVC(gamma=0.001, C=100., probability=True)
model = clf.fit(x_train, y_train)
```

2. Call the explainer: To initialize an explainer object, you need to pass your model and some training data to the explainer's constructor. You can also optionally pass in feature names and output class names (if doing classification) which will be used to make your explanations and visualizations more informative. Here is how to instantiate an explainer object using *TabularExplainer*, *MimicExplainer*, or *PFIExplainer* locally. *TabularExplainer* calls one of the five SHAP explainers underneath (*TreeExplainer*, *DeepExplainer*, *LinearExplainer*, *KernelExplainer*, or *GPUKernelExplainer*), and automatically selects the most appropriate one for your use case. You can also call any of its four underlying explainers directly.

```
  from interpret.ext.blackbox import TabularExplainer

  # "features" and "classes" fields are optional
  explainer = TabularExplainer(model,
                               x_train,
                               features=breast_cancer_data.feature_names,
                               classes=classes)
  # to utilise the GPU KernelExplainer, set parameter `use_gpu=True`

or
```

```
from interpret.ext.blackbox import MimicExplainer

# you can use one of the following four interpretable models as a global surrogate␣
↪to the black box model

from interpret.ext.glassbox import LGBMExplainableModel
from interpret.ext.glassbox import LinearExplainableModel
from interpret.ext.glassbox import SGDExplainableModel
from interpret.ext.glassbox import DecisionTreeExplainableModel

# "features" and "classes" fields are optional
# augment_data is optional and if true, oversamples the initialization examples to␣
↪improve surrogate model accuracy to fit original model.  Useful for high-
↪dimensional data where the number of rows is less than the number of columns.
# max_num_of_augmentations is optional and defines max number of times we can␣
↪increase the input data size.
# LGBMExplainableModel can be replaced with LinearExplainableModel,␣
↪SGDExplainableModel, or DecisionTreeExplainableModel
explainer = MimicExplainer(model,
                           x_train,
                           LGBMExplainableModel,
                           augment_data=True,
                           max_num_of_augmentations=10,
                           features=breast_cancer_data.feature_names,
                           classes=classes)
```

or

```
from interpret.ext.blackbox import PFIExplainer

# "features" and "classes" fields are optional
explainer = PFIExplainer(model,
                         features=breast_cancer_data.feature_names,
                         classes=classes)
```

After instantiating an explainer object, you can call the *explain_local* and *explain_global* methods to get local and global explanations.

For information on how to compute the explanation and view the feature importance values, please see the next section on **`importances`_**.

1.7 Importance Values

The feature importance values are used to rank the features in the model from most to least important.

Broadly, we can think of the importance values at a global and local level.

Instance-level feature importance measures focus on the contribution of features for a specific prediction (e.g., why did the model predict an 80% chance of breast cancer for Mary?), whereas aggregate-level feature importance takes all predictions into account (Overall, what are the top important features in predicting a high risk for breast cancer?):

Some feature importance values also have useful properties.

For example, for shap values, which come from game theory, the output score of the model is the sum of the feature importance values for each feature.

The following two sections demonstrate how you can get aggregate (global) and instance-level (local) feature importance values from an interpret-community style explanation.

1.7.1 Overall (Global) feature importance values

Get the aggregate feature importance values.

```
# you can use the training data or the test data here
global_explanation = explainer.explain_global(x_train)

# if you used the PFIExplainer in the previous step, use the next line of code␣
↪instead
# global_explanation = explainer.explain_global(x_train, true_labels=y_test)

# sorted feature importance values and feature names
sorted_global_importance_values = global_explanation.get_ranked_global_values()
sorted_global_importance_names = global_explanation.get_ranked_global_names()


# alternatively, you can print out a dictionary that holds the top K feature␣
↪names and values
global_explanation.get_feature_importance_dict()
```

1.7.2 Instance-level (Local) feature importance values

Get the instance-level feature importance values: use the following function calls to explain an individual instance or a group of instances. Please note that PFIExplainer does not support instance-level explanations.

```
# explain the first data point in the test set
local_explanation = explainer.explain_local(x_test[0])

# sorted feature importance values and feature names
sorted_local_importance_names = local_explanation.get_ranked_local_names()
sorted_local_importance_values = local_explanation.get_ranked_local_values()
```

or

```
# explain the first five data points in the test set
local_explanation = explainer.explain_local(x_test[0:5])

# sorted feature importance values and feature names
sorted_local_importance_names = local_explanation.get_ranked_local_names()
sorted_local_importance_values = local_explanation.get_ranked_local_values()
```

1.8 Raw feature transformations

Optionally, you can pass your feature transformation pipeline to the explainer to receive explanations in terms of the raw features before the transformation (rather than engineered features). If you skip this, the explainer provides explanations in terms of engineered features.

The format of supported transformations is same as the one described in sklearn-pandas. In general, any transformations are supported as long as they operate on a single column and are therefore clearly one to many.

We can explain raw features by either using a *sklearn.compose.ColumnTransformer* or a list of fitted transformer tuples. The cell below uses *sklearn.compose.ColumnTransformer*.

```
from sklearn.compose import ColumnTransformer

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# append classifier to preprocessing pipeline.
# now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(solver='lbfgs'))])
```

(continues on next page)

(continued from previous page)

```
# append classifier to preprocessing pipeline.
# now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(solver='lbfgs'))])


# clf.steps[-1][1] returns the trained classification model
# pass transformation as an input to create the explanation object
# "features" and "classes" fields are optional
tabular_explainer = TabularExplainer(clf.steps[-1][1],
                                     initialization_examples=x_train,
                                     features=dataset_feature_names,
                                     classes=dataset_classes,
                                     transformations=preprocessor)
```

In case you want to run the example with the list of fitted transformer tuples, use the following code:

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn_pandas import DataFrameMapper

# assume that we have created two arrays, numerical and categorical, which␣
↪holds the numerical and categorical feature names

numeric_transformations = [([f], Pipeline(steps=[('imputer', SimpleImputer(
    strategy='median')), ('scaler', StandardScaler())])) for f in numerical]

categorical_transformations = [([f], OneHotEncoder(
    handle_unknown='ignore', sparse=False)) for f in categorical]

transformations = numeric_transformations + categorical_transformations

# append model to preprocessing pipeline.
# now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', DataFrameMapper(transformations)),
                      ('classifier', LogisticRegression(solver='lbfgs'))])

# clf.steps[-1][1] returns the trained classification model
# pass transformation as an input to create the explanation object
# "features" and "classes" fields are optional
tabular_explainer = TabularExplainer(clf.steps[-1][1],
                                     initialization_examples=x_train,
                                     features=dataset_feature_names,
                                     classes=dataset_classes,
                                     transformations=transformations)
```

1.9 Visualizations

Install the raiwidgets package, the ExplanationDashboard has moved to the responsible-ai-toolbox repo:

```
pip install raiwidgets
```

Load the visualization dashboard in your notebook to understand and interpret your model:

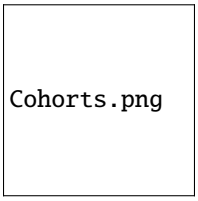
```
from raiwidgets import ExplanationDashboard

ExplanationDashboard(global_explanation, model, dataset=x_test, trueY=y_test)
```

Once you load the visualization dashboard, you can investigate different aspects of your dataset and trained model via four tab views:

- Model Performance
- Data Explorer
- Aggregate Feature Importance
- Individual Feature Importance and What-If

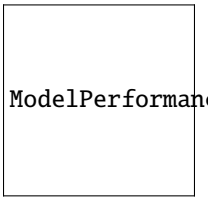
Note: Click on "Open in a new tab" on the top left corner to get a better view of the dashboard in a new tab.

You can further create custom cohorts (subgroups of your dataset) to explore the insights across different subgroups (e.g., women vs. men). The created cohorts can contain more than one filter (e.g., age < 30 and sex = female) and will be visible from all of the four tabs. The following sections demonstrate the visualization dashboard capabilities on a classification model trained on employee attrition dataset. Besides the default cohort (including the whole dataset), there are two additional cohorts created: employees with Age <= 35 and employees with Age > 35.



1.9.1 Model performance

This tab enables you to evaluate your model by observing its performance metrics and prediction probabilities/classes/values across different cohorts.



1.9.2 Dataset explorer

You can explore your dataset statistics by selecting different filters along the X, Y, and color axes of this tab to slice your data into different dimensions.

The following plots provide a global view of the trained model along with its predictions and explanations.

1.9.3 Aggregate feature importance (global explanation)

This view consists of two charts:

Table 3: Aggregate feature importance

Plot	Description
Feature Importance	Explore the top K important features that impact your overall model predictions (a.k.a. global explanation). Use the slider to show additional less important feature values. Select up to three cohorts to see their feature importance values side by side.
Dependence Plot	Click on any of the feature bars in the feature importance graph to see the relationship of the values of the selected feature to its corresponding feature importance values. Overall, this plot show how values of the selected feature impact model prediction.

1.9.4 Individual feature importance (local explanation) and what-if

You can click on any individual data point on the scatter plot to view its local feature importance values (local explanation) and individual conditional expectation (ICE) plot below. These are the capabilities covered in this tab:

Table 4: Individual feature importance

Plot	Description
Feature Importance Plot	Shows the top K (configurable K) important features for an individual prediction. Helps illustrate the local behavior of the underlying model on a specific data point.
Individual Conditional Expectation (ICE)	Allows feature value changes from a minimum value to a maximum value. Helps illustrate how the data point's prediction changes when a feature changes.
Perturbation Exploration (what if analysis)	Allows changes to feature values of the selected data point and observe resulting changes to prediction value. You can then save your hypothetical what-if data point.

1.10 Contributing

Interpret-Community is an open-source project and anyone is welcome to contribute to the project.

1.10.1 Acceptance criteria

All pull requests need to abide by the following criteria to be accepted:

- passing pipelines on the GitHub pull request
- approval from at least one maintainer
- tests for added / changed functionality

1.10.2 Development process

The project can be installed locally in editable mode using the following command:

```
pip install -e .
```

1.10.3 Testing

To run tests locally, please install all of the dependencies in your python environment:

```
pip install -r requirements.txt
pip install -r requirements-dev.txt
pip install -r requirements-vis.txt
pip install -r requirements-test.txt
```

The unit tests can then be run via the command:

```
pytest ./tests -m "not notebooks" -s -v
```

Code coverage can be run locally via the command:

```
pytest ./tests -m "not notebooks" -s -v --cov='interpret_community' --cov-
↪report=xml --cov-report=html
```

Notebook tests can also be run via the command:

```
python -m pytest tests/ -m "notebooks" -s -v
```

1.10.4 Linting

This repository uses flake8 for linting and isort to automatically sort imports.

Before running flake8, first please ensure that requirements-test.txt has been installed.

Then, you can run flake8 at the root folder of the repository via:

```
flake8 --max-line-length=119 .
```

You can use isort to identify if any imports are out of order:

```
isort . -c
```

You can even automatically fix your code by removing the -c argument:

```
isort .
```

After automatically sorting the imports in your code, make sure to add and commit your changes to git.

1.11 API Reference

1.11.1 interpret_community package

Module for interpreting, including feature and class importance for blackbox, greybox and glassbox models.

You can use model interpretability to explain why a model model makes the predictions it does and help build confidence in the model.

class interpret_community.**TabularExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, *use_gpu=False*, ***kwargs*)

Bases: *interpret_community.common.base_explainer.BaseExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Globally explains the black box model or function.

Parameters

- **evaluation_examples** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation. If SHAP is used for the explanation, it will also have the properties of a LocalExplanation and the ExpectedValuesMixin. If the model does classification, it will have the properties of the PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Locally explains the black box model or function.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation. If SHAP is used for the explanation, it will also have the properties of the ExpectedValuesMixin. If the model does classification, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

The tabular explainer meta-api for returning the best explanation result based on the given model.

Parameters

- **model** (*object*) – The model or pipeline to explain. A model that implements sklearn.predict() or sklearn.predict_proba() or pipeline function that accepts a 2d ndarray
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation, which will speed up the explanation process when number of features is large and the user already knows the set of interested features. The subset can be the top-k features from the model summary. This argument is not supported when transformations are set.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If the user is using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. A user can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
```

(continues on next page)

(continued from previous page)

```
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations

Subpackages

interpret_community.adapter package

Defines adapters for converting feature importance values to an explanation.

class interpret_community.adapter.**ExplanationAdapter**(*features=None*, *classification=False*, *method='Adapter'*)

Bases: object

An adapter for creating an interpret-community explanation from local importance values.

Parameters

- **features** (*list[str]*) – A list of feature names.
- **classification** (*bool*) – Indicates if this is a classification or regression explanation.
- **method** (*str*) – The explanation method used to explain the model (e.g., SHAP, LIME).

create_global(*local_importance_values*, *evaluation_examples=None*, *expected_values=None*, *include_local=True*, *batch_size=100*)

Create a global explanation from the list of local feature importance values.

Parameters

- **local_importance_values** (*numpy.ndarray or scipy.sparse.csr_matrix or list[scipy.sparse.csr_matrix]*) – The feature importance values.
- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **expected_values** (*numpy.ndarray*) – The expected values of the model.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.

- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

create_local(*local_importance_values*, *evaluation_examples=None*, *expected_values=None*)
Create a local explanation from the list of local feature importance values.

Parameters

- **local_importance_values** (*numpy.ndarray or scipy.sparse.csr_matrix or list[scipy.sparse.csr_matrix]*) – The feature importance values.
- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **expected_values** (*numpy.ndarray*) – The expected values of the model.

Submodules

interpret_community.adapter.explanation_adapter module

Defines an adapter for creating an interpret-community style explanation from other frameworks.

class interpret_community.adapter.explanation_adapter.**ExplanationAdapter**(*features=None*, *classification=False*, *method='Adapter'*)

Bases: object

An adapter for creating an interpret-community explanation from local importance values.

Parameters

- **features** (*list[str]*) – A list of feature names.
- **classification** (*bool*) – Indicates if this is a classification or regression explanation.
- **method** (*str*) – The explanation method used to explain the model (e.g., SHAP, LIME).

create_global(*local_importance_values*, *evaluation_examples=None*, *expected_values=None*, *include_local=True*, *batch_size=100*)
Create a global explanation from the list of local feature importance values.

Parameters

- **local_importance_values** (*numpy.ndarray or scipy.sparse.csr_matrix or list[scipy.sparse.csr_matrix]*) – The feature importance values.
- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **expected_values** (*numpy.ndarray*) – The expected values of the model.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

create_local(*local_importance_values*, *evaluation_examples=None*, *expected_values=None*)
Create a local explanation from the list of local feature importance values.

Parameters

- **local_importance_values** (*numpy.ndarray or scipy.sparse.csr_matrix or list[scipy.sparse.csr_matrix]*) – The feature importance values.
- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **expected_values** (*numpy.ndarray*) – The expected values of the model.

interpret_community.common package

Common infrastructure, class hierarchy and utilities for model explanations.

class interpret_community.common.**ModelSummary**

Bases: object

A structure for gathering and storing the parts of an explanation asset.

add_from_get_model_summary(*name*, *artifact_metadata_tuple*)

Update artifacts and metadata with new information.

Parameters

- **name** (*str*) – The name the new data should be associated with.
- **artifact_metadata_tuple** ((*list[dict], dict*)) – The tuple of artifacts and metadata to add to existing.

get_artifacts()

Get the list of artifacts.

Returns Artifact list.

Return type list[list[dict]]

get_metadata_dictionary()

Get the combined dictionary of metadata.

Returns Metadata dictionary.

Return type dict

Subpackages

interpret_community.common.model_wrapper package

Reimports helpful model wrapper and utils for implicitly rewrapping the model to conform to explainer contracts.

Submodules

interpret_community.common.aggregate module

Defines the aggregate explainer decorator for aggregating local explanations to global.

interpret_community.common.aggregate.**add_explain_global_method**(*cls*)

Decorate an explainer to allow aggregating local explanations to global.

Adds a protected method _explain_global that creates local explanations and then aggregates them to a global explanation by averaging.

interpret_community.common.aggregate.**init_aggregator_decorator**(*init_func*)
Decorate a constructor to wrap initialization examples in a DatasetWrapper.

Provided for convenience for tabular data explainers.

Parameters **init_func** (*Initialization constructor.*) – Initialization constructor where the second argument is a dataset.

interpret_community.common.base_explainer module

Defines the base explainer API to create explanations.

class interpret_community.common.base_explainer.**BaseExplainer**(**args*, ***kwargs*)
Bases: *interpret_community.common.base_explainer.GlobalExplainer*, *interpret_community.common.base_explainer.LocalExplainer*

The base class for explainers that create global and local explanations.

class interpret_community.common.base_explainer.**GlobalExplainer**(**args*, ***kwargs*)
Bases: abc.ABC, *interpret_community.common.chained_identity.ChainedIdentity*

The base class for explainers that create global explanations.

abstract explain_global(**args*, ***kwargs*)
Abstract method to globally explain the given model.

Note evaluation examples can be optional on derived classes since some explainers don't support it, for example MimicExplainer.

Returns A model explanation object containing the global explanation.

Return type *GlobalExplanation*

class interpret_community.common.base_explainer.**LocalExplainer**(**args*, ***kwargs*)
Bases: abc.ABC, *interpret_community.common.chained_identity.ChainedIdentity*

The base class for explainers that create local explanations.

abstract explain_local(*evaluation_examples*, ***kwargs*)
Abstract method to explain local instances.

Parameters **evaluation_examples** (*object*) – The evaluation examples.

Returns A model explanation object containing the local explanation.

Return type *LocalExplanation*

interpret_community.common.blackbox_explainer module

Defines the black box explainer API, which can either take in a black box model or function.

class interpret_community.common.blackbox_explainer.**BlackBoxExplainer**(*model*, *is_function=False*, *model_task=ModelTask.Unknown*, ***kwargs*)
Bases: *interpret_community.common.base_explainer.BaseExplainer*, *interpret_community.common.blackbox_explainer.BlackBoxMixin*

The base class for black box models or functions.

Parameters

- **model** (*object*) – The model to explain or function if is_function is True. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **is_function** (*bool*) – Default is false. Set to True if passing sklearn.predict or sklearn.predict_proba function instead of model.

class interpret_community.common.blackbox_explainer.**BlackBoxMixin**(*model*, *is_function=False*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.chained_identity.ChainedIdentity*

Mixin for black box models or functions.

Parameters

- **model** (*object*) – The model to explain or function if is_function is True. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **is_function** (*bool*) – Default is False. Set to True if passing sklearn.predict or sklearn.predict_proba function instead of model.

interpret_community.common.blackbox_explainer.**add_prepare_function_and_summary_method**(*cls*)

Decorate blackbox explainer to allow aggregating local explanations to global.

Adds two protected methods _function_subset_wrapper and _prepare_function_and_summary to the blackbox explainer. The former creates a wrapper around the prediction function for explaining subsets of features in the evaluation samples dataset. The latter calls the former to create a wrapper and also computes the summary background dataset for the explainer.

interpret_community.common.blackbox_explainer.**init_blackbox_decorator**(*init_func*)

Decorate a constructor to wrap initialization examples in a DatasetWrapper.

Provided for convenience for tabular data explainers.

Parameters **init_func** (*Initialization constructor.*) – Initialization constructor where the second argument is a dataset.

interpret_community.common.chained_identity module

Defines a light-weight chained identity for logging.

class interpret_community.common.chained_identity.**ChainedIdentity**(***kwargs*)

Bases: object

The base class for logging information.

interpret_community.common.constants module

Defines constants for interpret community.

class interpret_community.common.constants.**Attributes**
Bases: object

Provide constants for attributes.

EXPECTED_VALUE = 'expected_value'

class interpret_community.common.constants.**DNNFramework**
Bases: object

Provide DNN framework constants.

PYTORCH = 'pytorch'

TENSORFLOW = 'tensorflow'

class interpret_community.common.constants.**Defaults**
Bases: object

Provide constants for default values to explain methods.

AUTO = 'auto'

DEFAULT_BATCH_SIZE = 100

HDBSCAN = 'hdbscan'

MAX_DIM = 50

class interpret_community.common.constants.**Dynamic**
Bases: object

Provide constants for dynamically generated classes.

GLOBAL_EXPLANATION = 'DynamicGlobalExplanation'

LOCAL_EXPLANATION = 'DynamicLocalExplanation'

class interpret_community.common.constants.**ExplainParams**
Bases: object

Provide constants for interpret community (init, explain_local and explain_global) parameters.

BATCH_SIZE = 'batch_size'

CLASSES = 'classes'

CLASSIFICATION = 'classification'

EVAL_DATA = 'eval_data'

EVAL_Y_PRED = 'eval_y_predicted'

EVAL_Y_PRED_PROBA = 'eval_y_predicted_proba'

EXPECTED_VALUES = 'expected_values'

EXPLAIN_SUBSET = 'explain_subset'

EXPLANATION_ID = 'explanation_id'

FEATURES = 'features'

GLOBAL_IMPORTANCE_NAMES = 'global_importance_names'

GLOBAL_IMPORTANCE_RANK = 'global_importance_rank'

GLOBAL_IMPORTANCE_VALUES = 'global_importance_values'

GLOBAL_NAMES = 'global_names'

GLOBAL_RANK = 'global_rank'

GLOBAL_VALUES = 'global_values'

ID = 'id'

INCLUDE_LOCAL = 'include_local'

INIT_DATA = 'init_data'

IS_ENG = 'is_engineered'

IS_LOCAL_SPARSE = 'is_local_sparse'

IS_RAW = 'is_raw'

LOCAL_EXPLANATION = 'local_explanation'

LOCAL_IMPORTANCE_VALUES = 'local_importance_values'

METHOD = 'method'

MODEL_ID = 'model_id'

MODEL_TASK = 'model_task'

MODEL_TYPE = 'model_type'

NUM_CLASSES = 'num_classes'

NUM_EXAMPLES = 'num_examples'

NUM_FEATURES = 'num_features'

PER_CLASS_NAMES = 'per_class_names'

PER_CLASS_RANK = 'per_class_rank'

PER_CLASS_VALUES = 'per_class_values'

PROBABILITIES = 'probabilities'

SAMPLING_POLICY = 'sampling_policy'

SHAP_VALUES_OUTPUT = 'shap_values_output'

classmethod get_private(*explain_param*)

Return the private version of the ExplainParams property.

Parameters

- **cls** (ExplainParams) – ExplainParams input class.
- **explain_param** (*str*) – The ExplainParams property to get private version of.

Returns The private version of the property.

Return type str

classmethod get_serializable()

Return only the ExplainParams properties that have meaningful data values for serialization.

Parameters **cls** (ExplainParams) – ExplainParams input class.

Returns A set of property names, e.g., 'GLOBAL_IMPORTANCE_VALUES', 'MODEL_TYPE', etc.

Return type set[str]

class interpret_community.common.constants.**ExplainType**

Bases: object

Provide constants for model and explainer type information, useful for visualization.

CLASSIFICATION = 'classification'

DATA = 'data_type'

EXPLAIN = 'explain_type'

EXPLAINER = 'explainer'

FUNCTION = 'function'

GLOBAL = 'global'

HAN = 'han'

IS_ENG = 'is_engineered'

IS_RAW = 'is_raw'

LIME = 'lime'

LOCAL = 'local'

METHOD = 'method'

MIMIC = 'mimic'

MODEL = 'model_type'

MODEL_CLASS = 'model_class'

MODEL_TASK = 'model_task'

PFI = 'pfi'

REGRESSION = 'regression'

SHAP = 'shap'

SHAP_DEEP = 'shap_deep'

SHAP_GPU_KERNEL = 'shap_gpu_kernel'

SHAP_KERNEL = 'shap_kernel'

SHAP_LINEAR = 'shap_linear'

SHAP_TREE = 'shap_tree'

TABULAR = 'tabular'

class interpret_community.common.constants.**ExplainableModelType**(*value*)

Bases: str, enum.Enum

Provide constants for the explainable model type.

LINEAR_EXPLAINABLE_MODEL_TYPE = 'linear_explainable_model_type'

TREE_EXPLAINABLE_MODEL_TYPE = 'tree_explainable_model_type'

class interpret_community.common.constants.**ExplanationParams**
Bases: object

Provide constants for explanation parameters.

CLASSES = 'classes'

EXPECTED_VALUES = 'expected_values'

class interpret_community.common.constants.**Extension**
Bases: object

Provide constants for extensions to interpret package.

BLACKBOX = 'blackbox'

GLASSBOX = 'model'

GLOBAL = 'global'

GREYBOX = 'specific'

LOCAL = 'local'

class interpret_community.common.constants.**InterpretData**
Bases: object

Provide Data and Visualize constants for interpret core.

BASE_VALUE = 'Base Value'

EXPLANATION_CLASS_DIMENSION = 'explanation_class_dimension'

EXPLANATION_TYPE = 'explanation_type'

EXTRA = 'extra'

FEATURE_LIST = 'feature_list'

GLOBAL_FEATURE_IMPORTANCE = 'global_feature_importance'

INTERCEPT = 'intercept'

LOCAL_FEATURE_IMPORTANCE = 'local_feature_importance'

MLI = 'mli'

MULTICLASS = 'multiclass'

NAMES = 'names'

OVERALL = 'overall'

PERF = 'perf'

SCORES = 'scores'

SINGLE = 'single'

SPECIFIC = 'specific'

TYPE = 'type'

UNIVARIATE = 'univariate'

VALUE = 'value'

VALUES = 'values'

class interpret_community.common.constants.**LightGBMParams**
Bases: object

Provide constants for LightGBM.

CATEGORICAL_FEATURE = 'categorical_feature'

MIN_DATA_IN_LEAF = 'min_data_in_leaf'

class interpret_community.common.constants.**LightGBMSerializationConstants**
Bases: object

Provide internal class that defines fields used for MimicExplainer serialization.

IDENTITY = '_identity'

LOGGER = '_logger'

MODEL_STR = 'model_str'

MULTICLASS = 'multiclass'

OBJECTIVE = 'objective'

REGRESSION = 'regression'

TREE_EXPLAINER = '_tree_explainer'

enum_properties = ['_shap_values_output']

nonify_properties = ['_logger', '_tree_explainer']

save_properties = ['_lgbm']

class interpret_community.common.constants.**MimicSerializationConstants**
Bases: object

Provide internal class that defines fields used for MimicExplainer serialization.

ALLOW_ALL_TRANSFORMATIONS = '_allow_all_transformations'

FUNCTION = 'function'

IDENTITY = '_identity'

INITIALIZATION_EXAMPLES = 'initialization_examples'

LOGGER = '_logger'

MODEL = 'model'

ORIGINAL_EVAL_EXAMPLES = '_original_eval_examples'

PREDICT_PROBA_FLAG = 'predict_proba_flag'

RESET_INDEX = 'reset_index'

TIMESTAMP_FEATURIZER = '_timestamp_featurizer'

enum_properties = ['_shap_values_output']

nonify_properties = ['_logger', 'model', 'function', 'initialization_examples',
'_original_eval_examples', '_timestamp_featurizer']

save_properties = ['surrogate_model']

class interpret_community.common.constants.**ModelTask**(*value*)
Bases: str, enum.Enum

Provide model task constants. Can be 'classification', 'regression', or 'unknown'.

By default the model domain is inferred if 'unknown', but this can be overridden if you specify 'classification' or 'regression'.

Classification = 'classification'

Regression = 'regression'

Unknown = 'unknown'

class interpret_community.common.constants.**ResetIndex**(*value*)
Bases: str, enum.Enum

Provide index column handling constants. Can be 'ignore', 'reset' or 'reset_teacher'.

By default the index column is ignored, but you can override to reset it and make it a feature column that is then featurized to numeric, or reset it and ignore it during featurization but set it as the index when calling predict on the original model.

Ignore = 'ignore'

Reset = 'reset'

ResetTeacher = 'reset_teacher'

class interpret_community.common.constants.**SHAPDefaults**
Bases: object

Provide constants for default values to SHAP.

INDEPENDENT = 'independent'

class interpret_community.common.constants.**SKLearn**
Bases: object

Provide scikit-learn related constants.

EXAMPLES = 'examples'

LABELS = 'labels'

PREDICTIONS = 'predictions'

PREDICT_PROBA = 'predict_proba'

class interpret_community.common.constants.**Scipy**
Bases: object

Provide scipy related constants.

CSR_FORMAT = 'csr'

class interpret_community.common.constants.**ShapValuesOutput**(*value*)
Bases: str, enum.Enum

Provide constants for the SHAP values output from the explainer.

Can be 'default', 'probability' or 'teacher_probability'. If 'teacher_probability' is specified, we use the probabilities from the teacher model.

DEFAULT = 'default'

PROBABILITY = 'probability'

TEACHER_PROBABILITY = 'teacher_probability'

class interpret_community.common.constants.**Spacy**
Bases: object

Provide spaCy related constants.

EN = 'en'

NER = 'ner'

TAGGER = 'tagger'

class interpret_community.common.constants.**Tensorflow**
Bases: object

Provide TensorFlow and TensorBoard related constants.

CPU0 = '/CPU:0'

TFLOG = 'tflog'

interpret_community.common.error_handling module

Defines error handling utilities.

interpret_community.common.exception module

Defines different types of exceptions that this package can raise.

exception interpret_community.common.exception.**ScenarioNotSupportedException**
Bases: Exception

An exception indicating that some scenario is not supported.

Parameters **exception_message** (*str*) – A message describing the error.

interpret_community.common.explanation_utils module

Defines helpful utilities for summarizing and uploading data.

interpret_community.common.gpu_kmeans module

The code is based on the similar utility function from SHAP: https://github.com/slundberg/shap/blob/9411b68e8057a6c6f3621765b89b24d82bee13d4/shap/utils/_legacy.py This version makes use of cuml kmeans instead of sklearn for speed.

class interpret_community.common.gpu_kmeans.**Data**
Bases: object

class interpret_community.common.gpu_kmeans.**DenseData**(*data*, *group_names*, **args*)
Bases: *interpret_community.common.gpu_kmeans.Data*

interpret_community.common.gpu_kmeans.**kmeans**(*X*, *k*, *round_values=True*)
Summarize a dataset with k mean samples weighted by the number of data points they each represent.

Parameters

- **X** (*numpy.ndarray or pandas.DataFrame or any scipy.sparse matrix*) – Matrix of data samples to summarize (# samples x # features)
- **k** (*int*) – Number of means to use for approximation.
- **round_values** (*bool*) – For all i, round the ith dimension of each mean sample to match the nearest value from X[:,i]. This ensures discrete features always get a valid value.

Returns DenseData object.

Return type *DenseData*

interpret_community.common.metrics module

Defines metrics for validating model explanations.

interpret_community.common.metrics.**dcg**(*validate_order*, *ground_truth_order_relevance*, *top_values=10*)
Compute the discounted cumulative gain (DCG).

Compute the DCG as the sum of relevance scores penalized by the logarithmic position of the result. See https://en.wikipedia.org/wiki/Discounted_cumulative_gain for reference.

Parameters

- **validate_order** (*list*) – The order to validate.
- **ground_truth_order_relevance** (*list*) – The ground truth relevancy of the documents to compare to.
- **top_values** (*int*) – Specifies the top values to compute the DCG for. The default is 10.

interpret_community.common.metrics.**ndcg**(*validate_order*, *ground_truth_order*, *top_values=10*)
Compute the normalized discounted cumulative gain (NDCG).

Compute the NDCG as the ratio of the DCG for the validation order compared to the maximum DCG possible for the ground truth order. If the validation order is the same as the ground truth the NDCG will be the maximum of 1.0, and the least possible NDCG is 0.0. See https://en.wikipedia.org/wiki/Discounted_cumulative_gain for reference.

Parameters

- **validate_order** (*list*) – The order to validate for the documents. The values should be unique.
- **ground_truth_order** (*list*) – The true order of the documents. The values should be unique.
- **top_values** (*int*) – Specifies the top values to compute the NDCG for. The default is 10.

interpret_community.common.model_summary module

Defines a structure for gathering and storing the parts of an explanation asset.

class interpret_community.common.model_summary.**ModelSummary**
Bases: object

A structure for gathering and storing the parts of an explanation asset.

add_from_get_model_summary(*name*, *artifact_metadata_tuple*)
Update artifacts and metadata with new information.

Parameters

- **name** (*str*) – The name the new data should be associated with.
- **artifact_metadata_tuple** ((*list[dict]*, *dict*)) – The tuple of artifacts and metadata to add to existing.

get_artifacts()
Get the list of artifacts.

Returns Artifact list.

Return type list[list[dict]]

get_metadata_dictionary()
Get the combined dictionary of metadata.

Returns Metadata dictionary.

Return type dict

interpret_community.common.policy module

Defines explanation policies.

class interpret_community.common.policy.**SamplingPolicy**(*allow_eval_sampling=False*, *max_dim_clustering=50*, *sampling_method='hdbscan'*, ***kwargs*)
Bases: *interpret_community.common.chained_identity.ChainedIdentity*

Defines the sampling policy for downsampling the evaluation examples.

The policy is a set of parameters that can be tuned to speed up or improve the accuracy of the explain_model function during sampling.

Parameters

- **allow_eval_sampling** (*bool*) – Default to 'False'. Specify whether to allow sampling of evaluation data. If 'True', cluster the evaluation data and determine the optimal number of points for sampling. Set to 'True' to speed up the process when the evaluation data set is large and you only want to generate model summary info.
- **max_dim_clustering** (*int*) – Default to 50 and only take effect when 'allow_eval_sampling' is set to 'True'. Specify the dimensionality to reduce the evaluation data before clustering for sampling. When doing sampling to determine how aggressively to downsample without getting poor explanation results uses a heuristic to find the optimal number of clusters. Since KMeans performs poorly on high dimensional data PCA or Truncated SVD is first run to reduce the dimensionality, which is followed by finding the optimal k by running KMeans until a local minimum is reached as determined by computing the silhouette score, reducing k each time.
- **sampling_method** (*str*) – The sampling method for determining how much to downsample the evaluation data by. If allow_eval_sampling is True, the evaluation data is downsampled to a max_threshold, and then this heuristic is used to determine how much more to downsample the evaluation data without losing accuracy on the calculated feature importance values. By default, this is set to hdbscan, but you can also specify kmeans. With hdbscan the number of clusters is automatically determined and multiplied by a threshold. With kmeans, the optimal number of clusters is found by running KMeans until the maximum silhouette score is calculated, with k halved each time.

Return type dict

Returns The arguments for the sampling policy

property allow_eval_sampling
Get whether to allow sampling of evaluation data.

Returns Whether to allow sampling of evaluation data.

Return type bool

property max_dim_clustering
Get the dimensionality to reduce the evaluation data before clustering for sampling.

Returns The dimensionality to reduce the evaluation data before clustering for sampling.

Return type int

property sampling_method
Get the sampling method for determining how much to downsample the evaluation data by.

Returns The sampling method for determining how much to downsample the evaluation data by.

Return type str

interpret_community.common.progress module

Defines utilities for getting progress status for explanation.

interpret_community.common.progress.**get_tqdm**(*logger*, *show_progress*)
Get the tqdm progress bar function.

Parameters

- **logger** (*logger*) – The logger for logging info messages.
- **show_progress** (*bool*) – Default to 'True'. Determines whether to display the explanation status bar when using PFIExplainer.

Returns The tqdm (https://github.com/tqdm/tqdm) progress bar.

Return type function

interpret_community.common.serialization_utils module

Defines utility functions for serialization of data.

interpret_community.common.structured_model_explainer module

Defines the structured model based APIs for explainers used on specific types of models.

class interpret_community.common.structured_model_explainer.**PureStructuredModelExplainer**(*model*, ***kwargs*)

Bases: *interpret_community.common.base_explainer.BaseExplainer*

The base PureStructuredModelExplainer API for explainers used on specific models.

Parameters **model** (*object*) – The white box model to explain.

class interpret_community.common.structured_model_explainer.**StructuredInitModelExplainer**(*model*, *initialization_examples*, ***kwargs*)

Bases: *interpret_community.common.base_explainer.BaseExplainer*

The base StructuredInitModelExplainer API for explainers.

Used on specific models that require initialization examples.

Parameters

- **model** (*object*) – The white box model to explain.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.

interpret_community.common.warnings_suppressor module

Suppresses warnings on imports.

class interpret_community.common.warnings_suppressor.**shap_warnings_suppressor**

Bases: object

Context manager to suppress warnings from shap.

class interpret_community.common.warnings_suppressor.**tf_warnings_suppressor**

Bases: object

Context manager to suppress warnings from tensorflow.

interpret_community.dataset package

Defines a common dataset wrapper and common functions for data manipulation.

Subpackages

interpret_community.dataset.dataset_wrapper package

Reimports a helpful dataset wrapper to allow operations such as summarizing data, taking the subset or sampling.

Submodules

interpret_community.dataset.decorator module

Defines a decorator for tabular data which wraps pandas dataframes, scipy and numpy arrays in a DatasetWrapper.

interpret_community.dataset.decorator.**init_tabular_decorator**(*init_func*)

Decorate a constructor to wrap initialization examples in a DatasetWrapper.

Provided for convenience for tabular data explainers.

Parameters **init_func** (*Initialization constructor.*) – Initialization constructor where the second argument is a dataset.

interpret_community.dataset.decorator.**tabular_decorator**(*explain_func*)
Decorate an explanation function to wrap evaluation examples in a DatasetWrapper.

Parameters **explain_func** (*explanation function*) – An explanation function where the first argument is a dataset.

interpret_community.dataset.decorator.**wrap_dataset**(*dataset*)

interpret_community.explanation package

Defines the building blocks for explanations returned by explainers.

interpret_community.explanation.**load_explanation**(*path*)
Deserialize the explanation.

Parameters **path** (*str*) – The path to the directory in which the explanation will be saved. By default, must be a new directory to avoid overwriting any previous explanations. Set exist_ok to True to overrule this behavior.

Returns The deserialized explanation.

Return type Explanation

interpret_community.explanation.**save_explanation**(*explanation*, *path*, *exist_ok=False*)
Serialize the explanation.

Parameters

- **explanation** (*Explanation*) – The Explanation to be serialized.
- **path** (*str*) – The path to the directory in which the explanation will be saved. By default, must be a new directory to avoid overwriting any previous explanations. Set exist_ok to True to overrule this behavior.
- **exist_ok** (*bool*) – If False (default), the path provided by the user must not already exist and will be created by this function. If True, a prexisting path may be passed. Any preexisting files whose names match those of the files that make up the explanation will be overwritten.

Submodules

interpret_community.explanation.explanation module

Defines the explanations that are returned from explaining models.

class interpret_community.explanation.explanation.**BaseExplanation**(*method*, *model_task*, *model_type=None*, *explanation_id=None*, ***kwargs*)
Bases: abc.ABC, *interpret_community.common.chained_identity.ChainedIdentity*

The common explanation returned by explainers.

Parameters

- **method** (*str*) – The explanation method used to explain the model (e.g., SHAP, LIME).
- **model_task** (*str*) – The task of the original model i.e., classification or regression.

- **model_type** (*str*) – The type of the original model that was explained, e.g., sklearn.linear_model.LinearRegression.
- **explanation_id** (*str*) – The unique identifier for the explanation.

data(*key=None*)
Return the data of the explanation.

Parameters **key** (*int*) – The key for the local data to be retrieved.

Returns The explanation data.

Return type dict

property id
Get the explanation ID.

Returns The explanation ID.

Return type str

property method
Get the explanation method.

Returns The explanation method.

Return type str

property model_task
Get the task of the original model, i.e., classification or regression (others possibly in the future).

Returns The task of the original model.

Return type str

property model_type
Get the type of the original model that was explained.

Returns A class name or 'function', if that information is available.

Return type str

property name
Get the name of the explanation.

Returns The name of the explanation.

Return type str

abstract property selector
Get the local or global selector.

Returns The selector as a pandas dataframe of records.

Return type pandas.DataFrame

visualize(*key=None*)

class interpret_community.explanation.explanation.**ClassesMixin**(*classes=None*, *num_classes=None*, ***kwargs*)
Bases: *interpret_community.explanation.explanation.BaseExplanation*

The explanation mixin for classes.

This mixin is added when you specify classes in the classification scenario for creating a global or local explanation. This is activated when you specify the classes parameter for global or local explanations.

Parameters **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output.

property classes
Get the classes.

Returns The list of classes.

Return type list

property num_classes
Get the number of classes on the explanation.

Returns The number of classes on the explanation.

Return type int

class interpret_community.explanation.explanation.**ExpectedValuesMixin**(*expected_values=None*, ***kwargs*)

Bases: *interpret_community.explanation.explanation.BaseExplanation*

The explanation mixin for expected values.

Parameters **expected_values** (*numpy.ndarray*) – The expected values of the model.

data(*key=None*)
Return the data of the explanation with expected values added.

Parameters **key** (*int*) – The key for the local data to be retrieved.

Returns The explanation with expected values metadata added.

Return type dict

property expected_values
Get the expected values.

In the classification case where there are multiple expected values, they will be in the same order as the numeric indices that the classifier outputs.

Returns The expected value of the model applied to the set of initialization examples.

Return type list

class interpret_community.explanation.explanation.**FeatureImportanceExplanation**(*features=None*, *num_features=None*, *is_raw=False*, *is_engineered=False*, ***kwargs*)

Bases: *interpret_community.explanation.explanation.BaseExplanation*

The common feature importance explanation returned by explainers.

Parameters **features** (*Union[list[str], list[int]]*) – The feature names.

property features
Get the feature names.

Returns The feature names.

Return type list[str]

property is_engineered
Get the engineered explanation flag.

Returns True if it's an engineered explanation (specifically not raw). False if raw or unknown.

Return type bool

property is_raw
Get the raw explanation flag.

Returns True if it's a raw explanation. False if engineered or unknown.

Return type bool

property num_features
Get the number of features on the explanation.

Returns The number of features on the explanation.

Return type int

class interpret_community.explanation.explanation.**GlobalExplanation**(*global_importance_values=None*, *global_importance_rank=None*, *ranked_global_names=None*, *ranked_global_values=None*, ***kwargs*)

Bases: *interpret_community.explanation.explanation.FeatureImportanceExplanation*

The common global explanation returned by explainers.

Parameters

- **global_importance_values** (*numpy.ndarray*) – The feature importance values in the order of the original features.
- **global_importance_rank** (*numpy.ndarray*) – The feature indexes sorted by importance.
- **ranked_global_names** (*list[str] TODO*) – The feature names sorted by importance.
- **ranked_global_values** (*numpy.ndarray*) – The feature importance values sorted by importance.

data(*key=None*)
Return the data of the explanation with global importance values added.

Parameters **key** (*int*) – The key for the local data to be retrieved.

Returns The explanation with global importance values added.

Return type dict

get_feature_importance_dict(*top_k=None*)
Get a dictionary pairing ranked global names and feature importance values.

Parameters **top_k** (*int*) – If specified, only the top k names and values will be returned.

Returns A dictionary of feature names and their importance values.

Return type dict

get_ranked_global_names(*top_k=None*)
Get feature names sorted by global feature importance values, highest to lowest.

Parameters **top_k** (*int*) – If specified, only the top k names will be returned.

Returns The list of sorted features unless feature names are unavailable, feature indexes otherwise.

Return type list[str] or list[int]

get_ranked_global_values(*top_k=None*)
Get global feature importance sorted from highest to lowest.

Parameters **top_k** (*int*) – If specified, only the top k values will be returned.

Returns The list of sorted values.

Return type list[float]

get_raw_explanation(*feature_maps*, *raw_feature_names=None*, *eval_data=None*)
Get raw explanation using input feature maps.

Parameters

- **feature_maps** (*list[Union[numpy.ndarray, scipy.sparse.csr_matrix]]*) – list of feature maps from raw to generated feature where each array entry (raw_index, generated_index) is the weight for each raw, generated feature pair. The other entries are set to zero. For a sequence of transformations [t1, t2, ..., tn] generating generated features from raw features, the list of feature maps correspond to the raw to generated maps in the same order as t1, t2, etc. If the overall raw to generated feature map from t1 to tn is available, then just that feature map in a single element list can be passed.
- **raw_feature_names** (*[str]*) – list of raw feature names
- **eval_data** (*numpy.ndarray or pandas.DataFrame*) – Evaluation data.

Returns raw explanation

Return type *GlobalExplanation*

get_raw_feature_importances(*feature_maps*)
Get global raw feature importance.

Parameters

- **raw_feat_indices** (*list[list]*) – A list of lists of generated feature indices for each raw feature.
- **weights** (*list[list]*) – A list of list of weights to be applied to the generated feature importance.

Returns Raw feature importances.

Return type list[list] or list[list[list]]

property global_importance_rank
Get the overall feature importance rank or indexes.

For example, if original features are [f0, f1, f2, f3] and in global importance order they are [f2, f3, f0, f1], global_importance_rank would be [2, 3, 0, 1].

Returns The feature indexes sorted by importance.

Return type list[int]

property global_importance_values
Get the global feature importance values.

Values will be in their original order, the same as features, unless top_k was passed into upload_model_explanation or download_model_explanation. In those cases, returns the most important k values in highest to lowest importance order.

Returns The model level feature importance values.

Return type list[float]

property selector
Get the global selector if this is only a global explanation otherwise local.

Returns The selector as a pandas dataframe of records.

Return type pandas.DataFrame

class interpret_community.explanation.explanation.**LocalExplanation**(*local_importance_values=None*, ***kwargs*)

Bases: *interpret_community.explanation.explanation.FeatureImportanceExplanation*

The common local explanation returned by explainers.

Parameters **local_importance_values** (*numpy.ndarray or scipy.sparse.csr_matrix or list[scipy.sparse.csr_matrix]*) – The feature importance values.

data(*key=None*)

Return the data of the explanation with local importance values added.

Parameters **key** (*int*) – The key for the local data to be retrieved.

Returns The explanation with local importance values metadata added.

Return type dict

get_local_importance_rank()

Get local feature importance rank or indexes.

For example, if original features are [f0, f1, f2, f3] and in local importance order for the first data point they are [f2, f3, f0, f1], local_importance_rank[0] would be [2, 3, 0, 1] (or local_importance_rank[0][0] if classification).

For documentation regarding order of classes in the classification case, please see the docstring for local_importance_values.

Returns The feature indexes sorted by importance.

Return type list[list[int]] or list[list[list[int]]]

get_ranked_local_names(*top_k=None*)

Get feature names sorted by local feature importance values, highest to lowest.

For documentation regarding order of classes in the classification case, please see the docstring for local_importance_values.

Parameters **top_k** (*int*) – If specified, only the top k names will be returned.

Returns The list of sorted features unless feature names are unavailable, feature indexes otherwise.

Return type list[list[int or str]] or list[list[list[int or str]]]

get_ranked_local_values(*top_k=None*)

Get local feature importance sorted from highest to lowest.

For documentation regarding order of classes in the classification case, please see the docstring for local_importance_values.

Parameters **top_k** (*int*) – If specified, only the top k values will be returned.

Returns The list of sorted values.

Return type list[list[float]] or list[list[list[float]]]

get_raw_explanation(*feature_maps*, *raw_feature_names=None*, *eval_data=None*)

Get raw explanation using input feature maps.

Parameters

- **feature_maps** (*list[Union[numpy.ndarray, scipy.sparse.csr_matrix]]*) – list of feature maps from raw to generated feature where each array entry (raw_index, generated_index) is the weight for each raw, generated feature pair. The other entries are set to zero. For a sequence of transformations [t1, t2, ..., tn] generating generated features from raw features, the list of feature maps correspond to the raw to generated maps in the same order as t1, t2, etc. If the overall raw to generated feature map from t1 to tn is available, then just that feature map in a single element list can be passed.
- **raw_feature_names** (*[str]*) – list of raw feature names
- **eval_data** (*numpy.ndarray or pandas.DataFrame*) – Evaluation data.

Returns raw explanation

Return type *LocalExplanation*

get_raw_feature_importances(*raw_to_output_maps*)

Get local raw feature importance.

For documentation regarding order of classes in the classification case, please see the docstring for local_importance_values.

Parameters **raw_to_output_maps** (*list[numpy.ndarray]*) – A list of feature maps from raw to generated feature.

Returns Raw feature importance.

Return type list[list] or list[list[list]]

property is_local_sparse

Determines whether the local importance values are sparse.

Returns True if the local importance values are sparse.

Return type bool

property local_importance_values

Get the feature importance values in original order.

Returns

For a model with a single output such as regression, this returns a list of feature importance values for each data point. For models with vector outputs this function returns a list of such lists, one for each output. The dimension of this matrix is (# examples x # features) or (# classes x # examples x # features).

In the classification case, the order of classes is the order of the numeric indices that the classifier outputs. For example, if your target values are [2, 2, 0, 1, 2, 1, 0], where 0 is "dog", 1 is "cat", and 2 is "fish", the first 2d matrix of importance values will be for "dog", the second will be for "cat", and the last will be for "fish". If you choose to pass in a classes array to the explainer, the names should be passed in using this same order.

Return type list[list[float]] or list[list[list[float]]] or scipy.sparse.csr_matrix or list[scipy.sparse.csr_matrix]

property num_examples

Get the number of examples on the explanation.

Returns The number of examples on the explanation.

Return type int

property selector

Get the local selector.

Returns The selector as a pandas dataframe of records.

Return type pandas.DataFrame

class interpret_community.explanation.explanation.**PerClassMixin**(*per_class_values=None*, *per_class_rank=None*, *ranked_per_class_names=None*, *ranked_per_class_values=None*, ***kwargs*)

Bases: *interpret_community.explanation.explanation.ClassesMixin*

The explanation mixin for per class aggregated information.

This mixin is added for the classification scenario for global explanations. The per class importance values are group averages of local importance values across different classes.

Parameters

- **per_class_values** (*numpy.ndarray*) – The feature importance values for each class in the order of the original features.
- **per_class_importance_rank** (*numpy.ndarray*) – The feature indexes for each class sorted by importance.
- **ranked_per_class_names** (*list[str]*) – The feature names for each class sorted by importance.
- **ranked_per_class_values** (*numpy.ndarray*) – The feature importance values sorted by importance.

get_ranked_per_class_names(*top_k=None*)

Get feature names sorted by per class feature importance values, highest to lowest.

For documentation regarding order of classes, please see the docstring for per_class_values.

Parameters **top_k** (*int*) – If specified, only the top k names will be returned.

Returns The list of sorted features unless feature names are unavailable, feature indexes otherwise.

Return type list[list[str]] or list[list[int]]

get_ranked_per_class_values(*top_k=None*)

Get per class feature importance sorted from highest to lowest.

For documentation regarding order of classes, please see the docstring for per_class_values.

Parameters **top_k** (*int*) – If specified, only the top k values will be returned.

Returns The list of sorted values.

Return type list[list[float]]

property per_class_rank

Get the per class importance rank or indexes.

For example, if original features are [f0, f1, f2, f3] and in per class importance order they are [[f2, f3, f0, f1], [f0, f2, f3, f1]], per_class_rank would be [[2, 3, 0, 1], [0, 2, 3, 1]].

For documentation regarding order of classes, please see the docstring for per_class_values.

Returns The per class indexes that would sort per_class_values.

Return type list

property per_class_values
Get the per class importance values.

Values will be in their original order, the same as features, unless top_k was passed into upload_model_explanation or download_model_explanation. In those cases, returns the most important k values in highest to lowest importance order.

The order of classes in the output is the order of the numeric indices that the classifier outputs. For example, if your target values are [2, 2, 0, 1, 2, 1, 0], where 0 is "dog", 1 is "cat", and 2 is "fish", the first 2d matrix of importance values will be for "dog", the second will be for "cat", and the last will be for "fish". If you choose to pass in a classes array to the explainer, the names should be passed in using this same order.

Returns The model level per class feature importance values in original feature order.

Return type list

interpret_community.explanation.serialization module

Defines the save and load explanation methods and helpers for serialization.

interpret_community.explanation.serialization.**load_explanation**(*path*)
Deserialize the explanation.

Parameters **path** (*str*) – The path to the directory in which the explanation will be saved. By default, must be a new directory to avoid overwriting any previous explanations. Set exist_ok to True to overrule this behavior.

Returns The deserialized explanation.

Return type Explanation

interpret_community.explanation.serialization.**save_explanation**(*explanation*, *path*, *exist_ok=False*)
Serialize the explanation.

Parameters

- **explanation** (*Explanation*) – The Explanation to be serialized.
- **path** (*str*) – The path to the directory in which the explanation will be saved. By default, must be a new directory to avoid overwriting any previous explanations. Set exist_ok to True to overrule this behavior.
- **exist_ok** (*bool*) – If False (default), the path provided by the user must not already exist and will be created by this function. If True, a prexisting path may be passed. Any preexisting files whose names match those of the files that make up the explanation will be overwritten.

interpret_community.lime package

Module for LIME explainer.

class interpret_community.lime.**LIMEExplainer**(*model*, *initialization_examples*, *is_function=False*, *explain_subset=None*, *nclusters=10*, *features=None*, *classes=None*, *verbose=False*, *categorical_features=[]*, *show_progress=True*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, ***kwargs*)
Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)
Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (*interpret_community.common.policy.SamplingPolicy*) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object containing the global explanation.

Return type *GlobalExplanation*

explain_local(*evaluation_examples*)
Explain the function locally by using LIME.

Parameters

- **evaluation_examples** (*ml_wrappers.dataset.dataset_wrapper.DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.

Returns A model explanation object containing the local explanation.

Return type *LocalExplanation*

explainer_type = 'blackbox'
Defines the LIME Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The model to explain or function if is_function is True. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **is_function** (*bool*) – Default set to false, set to True if passing function instead of model.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.

- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **verbose** (*bool*) – If true, uses verbose logging in LIME.
- **categorical_features** (*Union[list[str], list[int]]*) – Categorical feature names or indexes. If names are passed, they will be converted into indexes first.
- **show_progress** (*bool*) – Default to 'True'. Determines whether to display the explanation status bar when using LIMEExplainer.
- **transformations** – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and

transformer. When transformations are provided, explanations are of the features before the transformation. The format for list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

If the user is using a transformation that is not in the list of sklearn.preprocessing transformations that we support then we cannot take a list of more than one column as input for the transformation. A user can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

Example of transformations that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

This would not work since it is hard to make out whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns. :type transformations: sklearn.compose.ColumnTransformer or list[tuple] :param allow_all_transformations: Allow many to many and many to one transformations :type allow_all_transformations: bool :param model_task: Optional parameter to specify whether the model is a classification or regression model.

> In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

Submodules

interpret_community.lime.lime_explainer module

Defines the LIMEExplainer for computing explanations on black box models using LIME.

class interpret_community.lime.lime_explainer.**LIMEExplainer**(*model*, *initialization_examples*, *is_function=False*, *explain_subset=None*, *nclusters=10*, *features=None*, *classes=None*, *verbose=False*, *categorical_features=[]*, *show_progress=True*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (*interpret_community.common.policy.SamplingPolicy*) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object containing the global explanation.

Return type *GlobalExplanation*

explain_local(*evaluation_examples*)

Explain the function locally by using LIME.

Parameters

- **evaluation_examples** (*ml_wrappers.dataset.dataset_wrapper.DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.

Returns A model explanation object containing the local explanation.

Return type *LocalExplanation*

explainer_type = 'blackbox'

Defines the LIME Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The model to explain or function if is_function is True. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **is_function** (*bool*) – Default set to false, set to True if passing function instead of model.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **verbose** (*bool*) – If true, uses verbose logging in LIME.
- **categorical_features** (*Union[list[str], list[int]]*) – Categorical feature names or indexes. If names are passed, they will be converted into indexes first.
- **show_progress** (*bool*) – Default to 'True'. Determines whether to display the explanation status bar when using LIMEExplainer.
- **transformations** – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and

transformer. When transformations are provided, explanations are of the features before the transformation. The format for list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

If the user is using a transformation that is not in the list of sklearn.preprocessing transformations that we support then we cannot take a list of more than one column as input for the transformation. A user can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

Example of transformations that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

This would not work since it is hard to make out whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns. :type transformations: sklearn.compose.ColumnTransformer or list[tuple] :param allow_all_transformations: Allow many to many and many to one transformations :type allow_all_transformations: bool :param model_task: Optional parameter to specify whether the model is a classification or regression model.

In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

interpret_community.mimic package

Module for mimic explainer and explainable surrogate models.

class interpret_community.mimic.**MimicExplainer**(*model*, *initialization_examples*, *explainable_model*, *explainable_model_args=None*, *is_function=False*, *augment_data=True*, *max_num_of_augmentations=10*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *shap_values_output=ShapValuesOutput.DEFAULT*, *categorical_features=None*, *model_task=ModelTask.Unknown*, *reset_index=ResetIndex.Ignore*, ***kwargs*)

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples=None*, *include_local=True*, *batch_size=100*)

Globally explains the blackbox model using the surrogate model.

If evaluation_examples are unspecified, retrieves global feature importance from explainable surrogate model. Note this will not include per class feature importance. If evaluation_examples are specified, aggregates local explanations to global from the given evaluation_examples - which computes both global and per class feature importance.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output. If specified, computes feature importance through aggregation.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If evaluation examples are specified and include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation. If evaluation_examples are passed in, it will also have the properties of a LocalExplanation. If the model is a classifier (has predict_proba), it will have the properties of ClassesMixin, and if evaluation_examples were passed in it will also have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Locally explains the blackbox model using the surrogate model.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

The Mimic Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explainable_model** (interpret_community.mimic.models.BaseExplainableModel) – The uninitialized surrogate model used to explain the black box model. Also known as the student model.
- **explainable_model_args** (*dict*) – An optional map of arguments to pass to the explainable model for initialization.
- **is_function** (*bool*) – Default is False. Set to True if passing function instead of model.
- **augment_data** (*bool*) – If True, oversamples the initialization examples to improve surrogate model accuracy to fit teacher model. Useful for high-dimensional data where the number of rows is less than the number of columns.
- **max_num_of_augmentations** (*int*) – Maximum number of times we can increase the input data size.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. Note for mimic explainer this will not affect the execution time of getting the global explanation. This argument is not supported when transformations are set.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **shap_values_output** (interpret_community.common.constants.ShapValuesOutput) – The shap values output from the explainer. Only applies to tree-based models that are in terms of raw feature values instead of probabilities. Can be default, probability or teacher_probability. If probability or teacher_probability are specified, we approximate the feature importance values as probabilities instead of using the default values. If teacher probability is specified, we use the probabilities from the teacher model as opposed to the surrogate model.
- **categorical_features** (*Union[list[str], list[int]]*) – Categorical feature names or indexes. If names are passed, they will be converted into indexes first. Note if pandas indexes are categorical, you can either pass the name of the index or the index as if the pandas index was inserted at the end of the input dataframe.
- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.
- **reset_index** (*str*) – Uses the pandas DataFrame index column as part of the features when training the surrogate model.

get_surrogate_model_replication_measure(*training_data*)

Return the metric which tells how well the surrogate model replicates the teacher model.

For classification scenarios, this function will return accuracy. For regression scenarios, this function will return r2_score.

Parameters **training_data** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The data for getting the replication metric.

Returns Metric that tells how well the surrogate model replicates the behavior of teacher model.

Return type float

Subpackages

interpret_community.mimic.models package

Module for explainable surrogate models.

class interpret_community.mimic.models.**BaseExplainableModel**(***kwargs*)

Bases: abc.ABC, *interpret_community.common.chained_identity.ChainedIdentity*

The base class for models that can be explained.

abstract property expected_values
Abstract property to get the expected values.

abstract explain_global(***kwargs*)
Abstract method to get the global feature importances from the trained explainable model.

abstract explain_local(*evaluation_examples*, ***kwargs*)
Abstract method to get the local feature importances from the trained explainable model.

static explainable_model_type()
Retrieve the model type.

abstract fit(***kwargs*)
Abstract method to fit the explainable model.

abstract property model
Abstract property to get the underlying model.

abstract predict(*dataset*, ***kwargs*)
Abstract method to predict labels using the explainable model.

abstract predict_proba(*dataset*, ***kwargs*)
Abstract method to predict probabilities using the explainable model.

class interpret_community.mimic.models.**DecisionTreeExplainableModel**(*multiclass=False*, *random_state=123*, *shap_values_output=ShapValuesOutput.DEFA*, *classification=True*, ***kwargs*)

Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values
Use TreeExplainer to get the expected values.

Returns The expected values of the decision tree tree model.

Return type list

explain_global(***kwargs*)
Call tree model feature importances to get the global feature importances from the tree surrogate model.

Returns The global explanation of feature importances.

Return type list

explain_local(*evaluation_examples*, *probabilities=None*, ***kwargs*)
Use TreeExplainer to get the local feature importances from the trained explainable model.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.
- **probabilities** (*numpy.ndarray*) – If output_type is probability, can specify the teacher model's probability for scaling the shap values.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

static explainable_model_type()
Retrieve the model type.

Returns Tree explainable model type.

Return type *interpret_community.common.constants.ExplainableModelType*

explainer_type = 'model'
Decision Tree explainable model.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.
- **random_state** (*int*) – Int to seed the model.
- **shap_values_output** (interpret_community.common.constants.ShapValuesOutput) – The type of the output from explain_local when using TreeExplainer. Currently only types 'default', 'probability' and 'teacher_probability' are supported. If 'probability' is specified, then we approximately scale the raw log-odds values from the TreeExplainer to probabilities.
- **classification** (*bool*) – Indicates if this is a classification or regression explanation.

fit(*dataset*, *labels*, ***kwargs*)
Call tree fit to fit the explainable model.

param dataset The dataset to train the model on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.ndarray

If multiclass=True, uses the parameters for DecisionTreeClassifier: Build a decision tree classifier from the training set (X, y).

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The training input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csc_matrix`.

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The target values (class labels) as integers or strings.

sample_weight [arraylike of shape (n_samples,), default=None] Sample weights. If None, then samples are equally weighted. Splits that would create child nodes with net zero or negative weight are ignored while searching for a split in each node. Splits are also ignored if they would result in any single class carrying a negative weight in either child node.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

self [DecisionTreeClassifier] Fitted estimator.

Otherwise, if multiclass=False, uses the parameters for DecisionTreeRegressor: Build a decision tree regressor from the training set (X, y).

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The training input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csc_matrix`.

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The target values (real numbers). Use `dtype=np.float64` and `order='C'` for maximum efficiency.

sample_weight [arraylike of shape (n_samples,), default=None] Sample weights. If None, then samples are equally weighted. Splits that would create child nodes with net zero or negative weight are ignored while searching for a split in each node.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

self [DecisionTreeRegressor] Fitted estimator.

`property model`

Retrieve the underlying model.

Returns The decision tree model, either classifier or regressor.

Return type Union[sklearn.tree.DecisionTreeClassifier, sklearn.tree.DecisionTreeRegressor]

`predict`(*dataset*, ***kwargs*)

Call tree predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for DecisionTreeClassifier: Predict class or regression value for X.

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The predicted classes, or the predict values.

Otherwise, if multiclass=False, uses the parameters for DecisionTreeRegressor: Predict class or regression value for X.

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The predicted classes, or the predict values.

`predict_proba`(*dataset*, ***kwargs*)

Call tree predict_proba to predict probabilities using the explainable model.

param dataset The dataset to predict probabilities on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for DecisionTreeClassifier: Predict class probabilities of the input samples X.

The predicted class probability is the fraction of samples of the same class in a leaf.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

proba [ndarray of shape (n_samples, n_classes) or list of n_outputs such arrays if n_outputs > 1] The class probabilities of the input samples. The order of the classes corresponds to that in the attribute *classes_*.

Otherwise predict_proba is not supported for regression or binary classification.

`class interpret_community.mimic.models.`**`LGBMExplainableModel`**(*multiclass=False*, *random_state=123*, *shap_values_output=ShapValuesOutput.DEFAULT*, *classification=True*, ***kwargs*)

Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values
Use TreeExplainer to get the expected values.

Returns The expected values of the LightGBM tree model.

Return type list

explain_global(***kwargs*)
Call lightgbm feature importances to get the global feature importances from the explainable model.

Returns The global explanation of feature importances.

Return type numpy.ndarray

explain_local(*evaluation_examples*, *probabilities=None*, ***kwargs*)
Use TreeExplainer to get the local feature importances from the trained explainable model.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.
- **probabilities** (*numpy.ndarray*) – If output_type is probability, can specify the teacher model's probability for scaling the shap values.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

static explainable_model_type()
Retrieve the model type.

Returns Tree explainable model type.

Return type *ExplainableModelType*

explainer_type = 'model'
LightGBM (fast, high performance framework based on decision tree) explainable model.

Please see documentation for more details: https://github.com/Microsoft/LightGBM

Additional arguments to LightGBMClassifier and LightGBMRegressor can be passed through kwargs.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.
- **random_state** (*int*) – Int to seed the model.
- **shap_values_output** (*interpret_community.common.constants.ShapValuesOutput*) – The type of the output from explain_local when using TreeExplainer. Currently only types 'default', 'probability' and 'teacher_probability' are supported. If 'probability' is specified, then we approximately scale the raw log-odds values from the TreeExplainer to probabilities.
- **classification** (*bool*) – Indicates if this is a classification or regression explanation.

fit(*dataset*, *labels*, ***kwargs*)
Call lightgbm fit to fit the explainable model.

Parameters

- **dataset** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The dataset to train the model on.

- **labels** (*numpy.ndarray*) – The labels to train the model on.

property model
Retrieve the underlying model.

Returns The lightgbm model, either classifier or regressor.

Return type Union[LGBMClassifier, LGBMRegressor]

predict(*dataset*, ***kwargs*)
Call lightgbm predict to predict labels using the explainable model.

Parameters **dataset** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – The dataset to predict on.

Returns The predictions of the model.

Return type list

predict_proba(*dataset*, ***kwargs*)
Call lightgbm predict_proba to predict probabilities using the explainable model.

Parameters **dataset** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – The dataset to predict probabilities on.

Returns The predictions of the model.

Return type list

class interpret_community.mimic.models.**LinearExplainableModel**(*multiclass=False*, *random_state=123*, *classification=True*, *sparse_data=False*, ***kwargs*)
Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values
Use LinearExplainer to get the expected values.

Returns The expected values of the linear model.

Return type list

explain_global(***kwargs*)
Call coef to get the global feature importances from the linear surrogate model.

Returns The global explanation of feature importances.

Return type list

explain_local(*evaluation_examples*, ***kwargs*)
Use LinearExplainer to get the local feature importances from the trained explainable model.

Parameters **evaluation_examples** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

static explainable_model_type()
Retrieve the model type.

Returns Linear explainable model type.

Return type *ExplainableModelType*

`explainer_type = 'model'`
Linear explainable model.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.
- **random_state** (*int*) – Int to seed the model.
- **classification** (*bool*) – Indicates whether the model is used for classification or regression scenario.
- **sparse_data** (*bool*) – Indicates whether the training data will be sparse.

`fit`(*dataset*, *labels*, ***kwargs*)
Call linear fit to fit the explainable model.

Store the mean and covariance of the background data for local explanation.

param dataset The dataset to train the model on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.ndarray

If multiclass=True, uses the parameters for LogisticRegression:

Fit the model according to the given training data.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] Training vector, where *n_samples* is the number of samples and *n_features* is the number of features.

y [arraylike of shape (n_samples,)] Target vector relative to X.

sample_weight [arraylike of shape (n_samples,) default=None] Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

New in version 0.17: *sample_weight* support to LogisticRegression.

Returns

self Fitted estimator.

Notes

The SAGA solver supports both float64 and float32 bit arrays.

Otherwise, if multiclass=False, uses the parameters for LinearRegression:

Fit linear model.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] Training data.

y [arraylike of shape (n_samples,) or (n_samples, n_targets)] Target values. Will be cast to X's dtype if necessary.

sample_weight [arraylike of shape (n_samples,), default=None] Individual weights for each sample.

New in version 0.17: parameter *sample_weight* support to LinearRegression.

Returns

self [object] Fitted Estimator.

`property model`

Retrieve the underlying model.

Returns The linear model, either classifier or regressor.

Return type Union[LogisticRegression, LinearRegression]

`predict`(*dataset*, ***kwargs*)

Call linear predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LogisticRegression:

Predict class labels for samples in X.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The data matrix for which we want to get the predictions.

Returns

y_pred [ndarray of shape (n_samples,)] Vector containing the class labels for each sample.

Otherwise, if multiclass=False, uses the parameters for LinearRegression:

Predict using the linear model.

Parameters

X [arraylike or sparse matrix, shape (n_samples, n_features)] Samples.

Returns

C [array, shape (n_samples,)] Returns predicted values.

`predict_proba`(*dataset*, ***kwargs*)

Call linear predict_proba to predict probabilities using the explainable model.

param dataset The dataset to predict probabilities on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LogisticRegression:

Probability estimates.

The returned estimates for all classes are ordered by the label of classes.

For a multi_class problem, if multi_class is set to be "multinomial" the softmax function is used to find the predicted probability of each class. Else use a onevsrest approach, i.e calculate the probability of each class assuming it to be positive using the logistic function. and normalize these values across all the classes.

Parameters

X [arraylike of shape (n_samples, n_features)] Vector to be scored, where *n_samples* is the number of samples and *n_features* is the number of features.

Returns

T [arraylike of shape (n_samples, n_classes)] Returns the probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

Otherwise predict_proba is not supported for regression or binary classification.

class interpret_community.mimic.models.**SGDExplainableModel**(*multiclass=False*, *random_state=123*, *classification=True*, ***kwargs*)

Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values

Use LinearExplainer to get the expected values.

Returns The expected values of the linear model.

Return type list

explain_global(***kwargs*)

Call coef to get the global feature importances from the SGD surrogate model.

Returns The global explanation of feature importances.

Return type list

explain_local(*evaluation_examples*, ***kwargs*)

Use LinearExplainer to get the local feature importances from the trained explainable model.

Parameters **evaluation_examples** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

explainer_type = 'model'

Stochastic Gradient Descent explainable model.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.
- **random_state** (*int*) – Int to seed the model.

fit(*dataset*, *labels*, ***kwargs*)

Call linear fit to fit the explainable model.

Store the mean and covariance of the background data for local explanation.

param dataset The dataset to train the model on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.ndarray

If multiclass=True, uses the parameters for SGDClassifier: Fit linear model with Stochastic Gradient Descent.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Training data.

y [ndarray of shape (n_samples,)] Target values.

coef_init [ndarray of shape (n_classes, n_features), default=None] The initial coefficients to warmstart the optimization.

intercept_init [ndarray of shape (n_classes,), default=None] The initial intercept to warmstart the optimization.

sample_weight [arraylike, shape (n_samples,), default=None] Weights applied to individual samples. If not provided, uniform weights are assumed. These weights will be multiplied with class_weight (passed through the constructor) if class_weight is specified.

Returns

self [object] Returns an instance of self.

Otherwise, if multiclass=False, uses the parameters for SGDRegressor: Fit linear model with Stochastic Gradient Descent.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Training data.

y [ndarray of shape (n_samples,)] Target values.

coef_init [ndarray of shape (n_features,), default=None] The initial coefficients to warmstart the optimization.

intercept_init [ndarray of shape (1,), default=None] The initial intercept to warmstart the optimization.

sample_weight [arraylike, shape (n_samples,), default=None] Weights applied to individual samples (1. for unweighted).

Returns

self [object] Fitted *SGDRegressor* estimator.

`property model`

Retrieve the underlying model.

Returns The SGD model, either classifier or regressor.

Return type Union[SGDClassifier, SGDRegressor]

`predict`(*dataset*, ***kwargs*)

Call SGD predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for SGDClassifier:

Predict class labels for samples in X.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The data matrix for which we want to get the predictions.

Returns

y_pred [ndarray of shape (n_samples,)] Vector containing the class labels for each sample.

Otherwise, if multiclass=False, uses the parameters for SGDRegressor: Predict using the linear model.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Input data.

Returns

ndarray of shape (n_samples,) Predicted target values per element in X.

`predict_proba`(*dataset*, ***kwargs*)

Call SGD predict_proba to predict probabilities using the explainable model.

param dataset The dataset to predict probabilities on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for SGDClassifier: Probability estimates.

This method is only available for log loss and modified Huber loss.

Multiclass probability estimates are derived from binary (onevs.rest) estimates by simple normalization, as recommended by Zadrozny and Elkan.

Binary probability estimates for loss="modified_huber" are given by (clip(decision_function(X), 1, 1) + 1) / 2. For other loss functions it is necessary to perform proper probability calibration by wrapping the classifier with `CalibratedClassifierCV` instead.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Input data for prediction.

Returns

ndarray of shape (n_samples, n_classes) Returns the probability of the sample for each class in the model, where classes are ordered as they are in *self.classes_*.

References

Zadrozny and Elkan, "Transforming classifier scores into multiclass probability estimates", SIGKDD'02, https://dl.acm.org/doi/pdf/10.1145/775047.775151

The justification for the formula in the loss="modified_huber" case is in the appendix B in: http://jmlr.csail.mit.edu/papers/volume2/zhang02c/zhang02c.pdf

Otherwise predict_proba is not supported for regression or binary classification.

Submodules

interpret_community.mimic.models.explainable_model module

Defines the base API for explainable models.

class interpret_community.mimic.models.explainable_model.**BaseExplainableModel**(***kwargs*)

Bases: abc.ABC, *interpret_community.common.chained_identity.ChainedIdentity*

The base class for models that can be explained.

abstract property expected_values
Abstract property to get the expected values.

abstract explain_global(***kwargs*)
Abstract method to get the global feature importances from the trained explainable model.

abstract explain_local(*evaluation_examples*, ***kwargs*)
Abstract method to get the local feature importances from the trained explainable model.

static explainable_model_type()
Retrieve the model type.

abstract fit(***kwargs*)
Abstract method to fit the explainable model.

abstract property model
Abstract property to get the underlying model.

abstract predict(*dataset*, ***kwargs*)
Abstract method to predict labels using the explainable model.

abstract predict_proba(*dataset*, ***kwargs*)
Abstract method to predict probabilities using the explainable model.

interpret_community.mimic.models.lightgbm_model module

Defines an explainable lightgbm model.

class interpret_community.mimic.models.lightgbm_model.**LGBMExplainableModel**(*multiclass=False*, *random_state=123*, *shap_values_output=ShapValuesOut*, *classification=True*, ***kwargs*)

Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values
Use TreeExplainer to get the expected values.

Returns The expected values of the LightGBM tree model.

Return type list

explain_global(***kwargs*)
Call lightgbm feature importances to get the global feature importances from the explainable model.

Returns The global explanation of feature importances.

Return type numpy.ndarray

explain_local(*evaluation_examples*, *probabilities=None*, ***kwargs*)
Use TreeExplainer to get the local feature importances from the trained explainable model.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.
- **probabilities** (*numpy.ndarray*) – If output_type is probability, can specify the teacher model's probability for scaling the shap values.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

static explainable_model_type()
Retrieve the model type.

Returns Tree explainable model type.

Return type *ExplainableModelType*

explainer_type = 'model'
LightGBM (fast, high performance framework based on decision tree) explainable model.

Please see documentation for more details: https://github.com/Microsoft/LightGBM

Additional arguments to LightGBMClassifier and LightGBMRegressor can be passed through kwargs.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.
- **random_state** (*int*) – Int to seed the model.
- **shap_values_output** (*interpret_community.common.constants.ShapValuesOutput*) – The type of the output from explain_local when using TreeExplainer. Currently only types 'default', 'probability' and 'teacher_probability' are supported. If 'probability' is specified, then we approximately scale the raw log-odds values from the TreeExplainer to probabilities.
- **classification** (*bool*) – Indicates if this is a classification or regression explanation.

fit(*dataset*, *labels*, ***kwargs*)
Call lightgbm fit to fit the explainable model.

Parameters

- **dataset** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The dataset to train the model on.
- **labels** (*numpy.ndarray*) – The labels to train the model on.

property model
Retrieve the underlying model.

Returns The lightgbm model, either classifier or regressor.

Return type Union[LGBMClassifier, LGBMRegressor]

predict(*dataset*, ***kwargs*)
Call lightgbm predict to predict labels using the explainable model.

Parameters **dataset** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The dataset to predict on.

Returns The predictions of the model.

Return type list

predict_proba(*dataset*, ***kwargs*)
Call lightgbm predict_proba to predict probabilities using the explainable model.

Parameters **dataset** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The dataset to predict probabilities on.

Returns The predictions of the model.

Return type list

interpret_community.mimic.models.linear_model module

Defines an explainable linear model.

class interpret_community.mimic.models.linear_model.**LinearExplainableModel**(*multiclass=False*, *random_state=123*, *classification=True*, *sparse_data=False*, ***kwargs*)

Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values
Use LinearExplainer to get the expected values.

Returns The expected values of the linear model.

Return type list

explain_global(***kwargs*)
Call coef to get the global feature importances from the linear surrogate model.

Returns The global explanation of feature importances.

Return type list

explain_local(*evaluation_examples*, ***kwargs*)
Use LinearExplainer to get the local feature importances from the trained explainable model.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

static explainable_model_type()
Retrieve the model type.

Returns Linear explainable model type.

Return type *ExplainableModelType*

explainer_type = 'model'
Linear explainable model.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.
- **random_state** (*int*) – Int to seed the model.
- **classification** (*bool*) – Indicates whether the model is used for classification or regression scenario.
- **sparse_data** (*bool*) – Indicates whether the training data will be sparse.

fit(*dataset*, *labels*, ***kwargs*)
Call linear fit to fit the explainable model.

Store the mean and covariance of the background data for local explanation.

param dataset The dataset to train the model on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.ndarray

If multiclass=True, uses the parameters for LogisticRegression:

Fit the model according to the given training data.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] Training vector, where *n_samples* is the number of samples and *n_features* is the number of features.

y [arraylike of shape (n_samples,)] Target vector relative to X.

sample_weight [arraylike of shape (n_samples,) default=None] Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

New in version 0.17: *sample_weight* support to LogisticRegression.

Returns

self Fitted estimator.

Notes

The SAGA solver supports both float64 and float32 bit arrays.

Otherwise, if multiclass=False, uses the parameters for LinearRegression:

Fit linear model.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] Training data.

y [arraylike of shape (n_samples,) or (n_samples, n_targets)] Target values. Will be cast to X's dtype if necessary.

sample_weight [arraylike of shape (n_samples,), default=None] Individual weights for each sample.

New in version 0.17: parameter *sample_weight* support to LinearRegression.

Returns

self [object] Fitted Estimator.

`property model`

Retrieve the underlying model.

Returns The linear model, either classifier or regressor.

Return type Union[LogisticRegression, LinearRegression]

`predict(`*dataset*, ***kwargs*`)`

Call linear predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LogisticRegression:

Predict class labels for samples in X.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The data matrix for which we want to get the predictions.

Returns

y_pred [ndarray of shape (n_samples,)] Vector containing the class labels for each sample.

Otherwise, if multiclass=False, uses the parameters for LinearRegression:

Predict using the linear model.

Parameters

X [arraylike or sparse matrix, shape (n_samples, n_features)] Samples.

Returns

C [array, shape (n_samples,)] Returns predicted values.

`predict_proba(`*dataset*, ***kwargs*`)`

Call linear predict_proba to predict probabilities using the explainable model.

param dataset The dataset to predict probabilities on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LogisticRegression:

Probability estimates.

The returned estimates for all classes are ordered by the label of classes.

For a multi_class problem, if multi_class is set to be "multinomial" the softmax function is used to find the predicted probability of each class. Else use a onevsrest approach, i.e calculate the probability of each class assuming it to be positive using the logistic function. and normalize these values across all the classes.

Parameters

X [arraylike of shape (n_samples, n_features)] Vector to be scored, where *n_samples* is the number of samples and *n_features* is the number of features.

Returns

T [arraylike of shape (n_samples, n_classes)] Returns the probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

Otherwise predict_proba is not supported for regression or binary classification.

class interpret_community.mimic.models.linear_model.**LinearExplainer**(*model*, *masker*, ***kwargs*)

Bases: shap.explainers._linear.Linear

Linear explainer with support for sparse data and sparse output.

shap_values(*evaluation_examples*)

Estimate the SHAP values for a set of samples.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The evaluation examples.

Returns 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).

Return type Union[list, numpy.ndarray]

class interpret_community.mimic.models.linear_model.**SGDExplainableModel**(*multiclass=False*, *random_state=123*, *classification=True*, ***kwargs*)

Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values

Use LinearExplainer to get the expected values.

Returns The expected values of the linear model.

Return type list

explain_global(***kwargs*)

Call coef to get the global feature importances from the SGD surrogate model.

Returns The global explanation of feature importances.

Return type list

explain_local(*evaluation_examples*, ***kwargs*)

Use LinearExplainer to get the local feature importances from the trained explainable model.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

explainer_type = 'model'

Stochastic Gradient Descent explainable model.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.
- **random_state** (*int*) – Int to seed the model.

fit(*dataset*, *labels*, ***kwargs*)

Call linear fit to fit the explainable model.

Store the mean and covariance of the background data for local explanation.

param dataset The dataset to train the model on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.ndarray

If multiclass=True, uses the parameters for SGDClassifier: Fit linear model with Stochastic Gradient Descent.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Training data.

y [ndarray of shape (n_samples,)] Target values.

coef_init [ndarray of shape (n_classes, n_features), default=None] The initial coefficients to warmstart the optimization.

intercept_init [ndarray of shape (n_classes,), default=None] The initial intercept to warmstart the optimization.

sample_weight [arraylike, shape (n_samples,), default=None] Weights applied to individual samples. If not provided, uniform weights are assumed. These weights will be multiplied with class_weight (passed through the constructor) if class_weight is specified.

Returns

self [object] Returns an instance of self.

Otherwise, if multiclass=False, uses the parameters for SGDRegressor: Fit linear model with Stochastic Gradient Descent.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Training data.

y [ndarray of shape (n_samples,)] Target values.

coef_init [ndarray of shape (n_features,), default=None] The initial coefficients to warmstart the optimization.

intercept_init [ndarray of shape (1,), default=None] The initial intercept to warmstart the optimization.

sample_weight [arraylike, shape (n_samples,), default=None] Weights applied to individual samples (1. for unweighted).

Returns

self [object] Fitted *SGDRegressor* estimator.

property model

Retrieve the underlying model.

Returns The SGD model, either classifier or regressor.

Return type Union[SGDClassifier, SGDRegressor]

`predict`(*dataset*, ***kwargs*)

Call SGD predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for SGDClassifier:

Predict class labels for samples in X.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The data matrix for which we want to get the predictions.

Returns

y_pred [ndarray of shape (n_samples,)] Vector containing the class labels for each sample.

Otherwise, if multiclass=False, uses the parameters for SGDRegressor: Predict using the linear model.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Input data.

Returns

ndarray of shape (n_samples,) Predicted target values per element in X.

`predict_proba`(*dataset*, ***kwargs*)

Call SGD predict_proba to predict probabilities using the explainable model.

param dataset The dataset to predict probabilities on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for SGDClassifier: Probability estimates.

This method is only available for log loss and modified Huber loss.

Multiclass probability estimates are derived from binary (onevs.rest) estimates by simple normalization, as recommended by Zadrozny and Elkan.

Binary probability estimates for loss="modified_huber" are given by (clip(decision_function(X), 1, 1) + 1) / 2. For other loss functions it is necessary to perform proper probability calibration by wrapping the classifier with `CalibratedClassifierCV` instead.

Parameters

X [{arraylike, sparse matrix}, shape (n_samples, n_features)] Input data for prediction.

Returns

ndarray of shape (n_samples, n_classes) Returns the probability of the sample for each class in the model, where classes are ordered as they are in *self.classes_*.

References

Zadrozny and Elkan, "Transforming classifier scores into multiclass probability estimates", SIGKDD'02, https://dl.acm.org/doi/pdf/10.1145/775047.775151

The justification for the formula in the loss="modified_huber" case is in the appendix B in: http://jmlr.csail.mit.edu/papers/volume2/zhang02c/zhang02c.pdf

Otherwise predict_proba is not supported for regression or binary classification.

interpret_community.mimic.models.tree_model module

Defines an explainable tree model.

class interpret_community.mimic.models.tree_model.**DecisionTreeExplainableModel**(*multiclass=False*, *random_state=123*, *shap_values_output=ShapValue*, *classification=True*, ***kwargs*)

Bases: *interpret_community.mimic.models.explainable_model.BaseExplainableModel*

available_explanations = ['global', 'local']

property expected_values

Use TreeExplainer to get the expected values.

Returns The expected values of the decision tree tree model.

Return type list

explain_global(***kwargs*)

Call tree model feature importances to get the global feature importances from the tree surrogate model.

Returns The global explanation of feature importances.

Return type list

explain_local(*evaluation_examples*, *probabilities=None*, ***kwargs*)

Use TreeExplainer to get the local feature importances from the trained explainable model.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The evaluation examples to compute local feature importances for.
- **probabilities** (*numpy.ndarray*) – If output_type is probability, can specify the teacher model's probability for scaling the shap values.

Returns The local explanation of feature importances.

Return type Union[list, numpy.ndarray]

static explainable_model_type()

Retrieve the model type.

Returns Tree explainable model type.

Return type *interpret_community.common.constants.ExplainableModelType*

explainer_type = 'model'

Decision Tree explainable model.

Parameters

- **multiclass** (*bool*) – Set to true to generate a multiclass model.

- **random_state** (*int*) – Int to seed the model.
- **shap_values_output** (interpret_community.common.constants.ShapValuesOutput) – The type of the output from explain_local when using TreeExplainer. Currently only types 'default', 'probability' and 'teacher_probability' are supported. If 'probability' is specified, then we approximately scale the raw log-odds values from the TreeExplainer to probabilities.
- **classification** (*bool*) – Indicates if this is a classification or regression explanation.

fit(*dataset*, *labels*, ***kwargs*)

Call tree fit to fit the explainable model.

param dataset The dataset to train the model on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.ndarray

If multiclass=True, uses the parameters for DecisionTreeClassifier: Build a decision tree classifier from the training set (X, y).

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The training input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csc_matrix`.

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The target values (class labels) as integers or strings.

sample_weight [arraylike of shape (n_samples,), default=None] Sample weights. If None, then samples are equally weighted. Splits that would create child nodes with net zero or negative weight are ignored while searching for a split in each node. Splits are also ignored if they would result in any single class carrying a negative weight in either child node.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

self [DecisionTreeClassifier] Fitted estimator.

Otherwise, if multiclass=False, uses the parameters for DecisionTreeRegressor: Build a decision tree regressor from the training set (X, y).

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The training input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csc_matrix`.

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The target values (real numbers). Use `dtype=np.float64` and `order='C'` for maximum efficiency.

sample_weight [arraylike of shape (n_samples,), default=None] Sample weights. If None, then samples are equally weighted. Splits that would create child nodes with net zero or negative weight are ignored while searching for a split in each node.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

self [DecisionTreeRegressor] Fitted estimator.

`property model`

Retrieve the underlying model.

Returns The decision tree model, either classifier or regressor.

Return type Union[sklearn.tree.DecisionTreeClassifier, sklearn.tree.DecisionTreeRegressor]

`predict`(*dataset*, ***kwargs*)

Call tree predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for DecisionTreeClassifier: Predict class or regression value for X.

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The predicted classes, or the predict values.

Otherwise, if multiclass=False, uses the parameters for DecisionTreeRegressor: Predict class or regression value for X.

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

y [arraylike of shape (n_samples,) or (n_samples, n_outputs)] The predicted classes, or the predict values.

`predict_proba`(*dataset*, ***kwargs*)

Call tree predict_proba to predict probabilities using the explainable model.

param dataset The dataset to predict probabilities on.

type dataset numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for DecisionTreeClassifier: Predict class probabilities of the input samples X.

The predicted class probability is the fraction of samples of the same class in a leaf.

Parameters

X [{arraylike, sparse matrix} of shape (n_samples, n_features)] The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

check_input [bool, default=True] Allow to bypass several input checking. Don't use this parameter unless you know what you're doing.

Returns

proba [ndarray of shape (n_samples, n_classes) or list of n_outputs such arrays if n_outputs > 1] The class probabilities of the input samples. The order of the classes corresponds to that in the attribute *classes_*.

Otherwise predict_proba is not supported for regression or binary classification.

interpret_community.mimic.models.tree_model_utils module

Defines utilities for tree-based explainable models.

Submodules

interpret_community.mimic.mimic_explainer module

Defines the Mimic Explainer for computing explanations on black box models or functions.

The mimic explainer trains an explainable model to reproduce the output of the given black box model. The explainable model is called a surrogate model and the black box model is called a teacher model. Once trained to reproduce the output of the teacher model, the surrogate model's explanation can be used to explain the teacher model.

class interpret_community.mimic.mimic_explainer.**MimicExplainer**(*model*, *initialization_examples*, *explainable_model*, *explainable_model_args=None*, *is_function=False*, *augment_data=True*, *max_num_of_augmentations=10*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *shap_values_output=ShapValuesOutput.DEFAULT*, *categorical_features=None*, *model_task=ModelTask.Unknown*, *reset_index=ResetIndex.Ignore*, ***kwargs*)

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples=None*, *include_local=True*, *batch_size=100*)
Globally explains the blackbox model using the surrogate model.

If evaluation_examples are unspecified, retrieves global feature importance from explainable surrogate model. Note this will not include per class feature importance. If evaluation_examples are specified, aggregates local explanations to global from the given evaluation_examples - which computes both global and per class feature importance.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output. If specified, computes feature importance through aggregation.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If evaluation examples are specified and include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation. If evaluation_examples are passed in, it will also have the properties of a LocalExplanation. If the model is a classifier (has predict_proba), it will have the properties of ClassesMixin, and if evaluation_examples were passed in it will also have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)
Locally explains the blackbox model using the surrogate model.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'
The Mimic Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explainable_model** (*interpret_community.mimic.models.BaseExplainableModel*) – The uninitialized surrogate model used to explain the black box model. Also known as the student model.
- **explainable_model_args** (*dict*) – An optional map of arguments to pass to the explainable model for initialization.
- **is_function** (*bool*) – Default is False. Set to True if passing function instead of model.

- **augment_data** (*bool*) – If True, oversamples the initialization examples to improve surrogate model accuracy to fit teacher model. Useful for high-dimensional data where the number of rows is less than the number of columns.
- **max_num_of_augmentations** (*int*) – Maximum number of times we can increase the input data size.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. Note for mimic explainer this will not affect the execution time of getting the global explanation. This argument is not supported when transformations are set.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

  The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.
- **shap_values_output** (interpret_community.common.constants.ShapValuesOutput) – The shap values output from the explainer. Only applies to tree-based models that are in terms of raw feature values instead of probabilities. Can be default, probability or teacher_probability. If probability or teacher_probability are specified, we approximate the feature importance values as probabilities instead of using

the default values. If teacher probability is specified, we use the probabilities from the teacher model as opposed to the surrogate model.

- **categorical_features** (*Union[list[str], list[int]]*) – Categorical feature names or indexes. If names are passed, they will be converted into indexes first. Note if pandas indexes are categorical, you can either pass the name of the index or the index as if the pandas index was inserted at the end of the input dataframe.
- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.
- **reset_index** (*str*) – Uses the pandas DataFrame index column as part of the features when training the surrogate model.

get_surrogate_model_replication_measure(*training_data*)

Return the metric which tells how well the surrogate model replicates the teacher model.

For classification scenarios, this function will return accuracy. For regression scenarios, this function will return r2_score.

Parameters **training_data** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – The data for getting the replication metric.

Returns Metric that tells how well the surrogate model replicates the behavior of teacher model.

Return type float

interpret_community.mimic.model_distill module

Utilities to train a surrogate model from teacher.

interpret_community.mlflow package

Module for interaction with MLflow.

interpret_community.mlflow.**get_explanation**(*run_id*, *name*)

Download and deserialize an explanation that has been logged to MLflow.

Parameters

- **run_id** (*str*) – The ID of the run the explanation was logged to.
- **name** (*str*) – The name given to the explanation when it was logged.

Returns The rehydrated explanation.

Return type Explanation

interpret_community.mlflow.**log_explanation**(*name*, *explanation*)

Log the explanation to MLflow using MLflow model logging.

Parameters

- **name** (*str*) – The name of the explanation. Will be used as a directory name.
- **explanation** (*Explanation*) – The explanation object to log.

interpret_community.mlflow.**save_model**(*path*, *loader_module=None*, *data_path=None*, *conda_env=None*, *mlflow_model=None*, ***kwargs*)

Save the explanation locally using the MLflow model format.

This function is necessary for log_explanation to work properly.

Parameters

- **path** (*str*) – The destination path for the saved explanation.
- **loader_module** (*str*) – The package that will be used to reload a serialized explanation. In this case, always interpret_community.mlflow.
- **data_path** (*str*) – The path to the serialized explanation files.
- **conda_env** (*str*) – The path to a YAML file with basic Python environment information.
- **mlflow_model** (*None*) – In our case, always None.

Returns The MLflow model representation of the explanation.

Return type mlflow.models.Model

Submodules

interpret_community.mlflow.mlflow module

interpret_community.mlflow.mlflow.**get_explanation**(*run_id*, *name*)

Download and deserialize an explanation that has been logged to MLflow.

Parameters

- **run_id** (*str*) – The ID of the run the explanation was logged to.
- **name** (*str*) – The name given to the explanation when it was logged.

Returns The rehydrated explanation.

Return type Explanation

interpret_community.mlflow.mlflow.**log_explanation**(*name*, *explanation*)

Log the explanation to MLflow using MLflow model logging.

Parameters

- **name** (*str*) – The name of the explanation. Will be used as a directory name.
- **explanation** (*Explanation*) – The explanation object to log.

interpret_community.mlflow.mlflow.**save_model**(*path*, *loader_module=None*, *data_path=None*, *conda_env=None*, *mlflow_model=None*, ***kwargs*)

Save the explanation locally using the MLflow model format.

This function is necessary for log_explanation to work properly.

Parameters

- **path** (*str*) – The destination path for the saved explanation.
- **loader_module** (*str*) – The package that will be used to reload a serialized explanation. In this case, always interpret_community.mlflow.
- **data_path** (*str*) – The path to the serialized explanation files.
- **conda_env** (*str*) – The path to a YAML file with basic Python environment information.

- **mlflow_model** (*None*) – In our case, always None.

Returns The MLflow model representation of the explanation.

Return type mlflow.models.Model

interpret_community.permutation package

Module for permutation feature importance.

class interpret_community.permutation.**PFIExplainer**(*model*, *is_function=False*, *metric=None*, *metric_args=None*, *is_error_metric=False*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *seed=0*, *for_classifier_use_predict_proba=False*, *show_progress=True*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.base_explainer.GlobalExplainer*, *interpret_community.common.blackbox_explainer.BlackBoxMixin*

available_explanations = ['global']

explain_global(*evaluation_examples*, *true_labels*)

Globally explains the blackbox model using permutation feature importance.

Note this will not include per class feature importances or local feature importances.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output through permutation feature importance.
- **true_labels** (*numpy.ndarray or pandas.DataFrame*) – An array of true labels used for reference to compute the evaluation metric for base case and after each permutation.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation. If the model is a classifier (has predict_proba), it will have the properties of ClassesMixin.

Return type DynamicGlobalExplanation

explainer_type = 'blackbox'

Defines the Permutation Feature Importance Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **is_function** (*bool*) – Default is False. Set to True if passing function instead of model.
- **metric** (*str or function that accepts two arrays, y_true and y_pred.*) – The metric name or function to evaluate the permutation. Note that if a metric function is provided, a higher value must be better. Otherwise, take the negative of the function or set is_error_metric to True. By default, if no metric is provided, F1 Score is used for binary classification, F1 Score with micro average is used for multiclass classification and mean absolute error is used for regression.
- **metric_args** (*dict*) – Optional arguments for metric function.

- **is_error_metric** (*bool*) – If custom metric function is provided, set to True if a higher value of the metric is better.
- **explain_subset** (*list[int]*) – List of feature indexes. If specified, only selects a subset of the features in the evaluation dataset for explanation. For permutation feature importance, we can shuffle, score and evaluate on the specified indexes when this parameter is set. This argument is not supported when transformations are set.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

  The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.
- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations.
- **seed** (*int*) – Random number seed for shuffling.
- **for_classifier_use_predict_proba** (*bool*) – If specifying a model instead of a function, and the model is a classifier, set to True instead of the default False to use predict_proba instead of predict when calculating the metric.

- **show_progress** (*bool*) – Default to 'True'. Determines whether to display the explanation status bar when using PFIExplainer.
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

Submodules

interpret_community.permutation.metric_constants module

Defines metric constants for PFIExplainer.

class interpret_community.permutation.metric_constants.**MetricConstants**(*value*)
Bases: str, enum.Enum

The metric to use for PFIExplainer.

AVERAGE_PRECISION_SCORE = 'average_precision_score'

EXPLAINED_VARIANCE_SCORE = 'explained_variance_score'

F1_SCORE = 'f1_score'

FBETA_SCORE = 'fbeta_score'

MEAN_ABSOLUTE_ERROR = 'mean_absolute_error'

MEAN_SQUARED_ERROR = 'mean_squared_error'

MEAN_SQUARED_LOG_ERROR = 'mean_squared_log_error'

MEDIAN_ABSOLUTE_ERROR = 'median_absolute_error'

PRECISION_SCORE = 'precision_score'

R2_SCORE = 'r2_score'

RECALL_SCORE = 'recall_score'

interpret_community.permutation.permutation_importance module

Defines the PFIExplainer for computing global explanations on black box models or functions.

The PFIExplainer uses permutation feature importance to compute a score for each column given a model based on how the output metric varies as each column is randomly permuted. Although very fast for computing global explanations, PFI does not support local explanations and can be inaccurate when there are feature interactions.

class interpret_community.permutation.permutation_importance.**PFIExplainer**(*model*, *is_function=False*, *metric=None*, *metric_args=None*, *is_error_metric=False*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *seed=0*, *for_classifier_use_predict_proba=Fal*, *show_progress=True*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.base_explainer.GlobalExplainer*, *interpret_community.common.blackbox_explainer.BlackBoxMixin*

available_explanations = ['global']

explain_global(*evaluation_examples*, *true_labels*)

Globally explains the blackbox model using permutation feature importance.

Note this will not include per class feature importances or local feature importances.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output through permutation feature importance.
- **true_labels** (*numpy.ndarray or pandas.DataFrame*) – An array of true labels used for reference to compute the evaluation metric for base case and after each permutation.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation. If the model is a classifier (has predict_proba), it will have the properties of ClassesMixin.

Return type DynamicGlobalExplanation

explainer_type = 'blackbox'

Defines the Permutation Feature Importance Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **is_function** (*bool*) – Default is False. Set to True if passing function instead of model.
- **metric** (*str or function that accepts two arrays, y_true and y_pred.*) – The metric name or function to evaluate the permutation. Note that if a metric function is provided, a higher value must be better. Otherwise, take the negative of the function or set is_error_metric to True. By default, if no metric is provided, F1 Score is used for binary classification, F1 Score with micro average is used for multiclass classification and mean absolute error is used for regression.
- **metric_args** (*dict*) – Optional arguments for metric function.

- **is_error_metric** (*bool*) – If custom metric function is provided, set to True if a higher value of the metric is better.
- **explain_subset** (*list[int]*) – List of feature indexes. If specified, only selects a subset of the features in the evaluation dataset for explanation. For permutation feature importance, we can shuffle, score and evaluate on the specified indexes when this parameter is set. This argument is not supported when transformations are set.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

  The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.
- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations.
- **seed** (*int*) – Random number seed for shuffling.
- **for_classifier_use_predict_proba** (*bool*) – If specifying a model instead of a function, and the model is a classifier, set to True instead of the default False to use predict_proba instead of predict when calculating the metric.

- **show_progress** (*bool*) – Default to 'True'. Determines whether to display the explanation status bar when using PFIExplainer.
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

interpret_community.permutation.permutation_importance.**labels_decorator**(*explain_func*)
Decorate PFI explainer to throw better error message if true_labels not passed.

Parameters **explain_func** (*explanation function*) – PFI explanation function.

interpret_community.shap package

Module for SHAP-based blackbox and greybox explainers.

class interpret_community.shap.**DeepExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *nclusters=10*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, *is_classifier=None*, ***kwargs*)

Bases: *interpret_community.common.structured_model_explainer.StructuredInitModelExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)
Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (*interpret_community.common.policy.SamplingPolicy*) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)
Explain the model by using SHAP's deep explainer.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

`explainer_type = 'specific'`

An explainer for DNN models, implemented using shap's DeepExplainer, supports TensorFlow and PyTorch.

Parameters

- **model** (*PyTorch or TensorFlow model*) – The DNN model to explain.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model.

class interpret_community.shap.**GPUKernelExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *is_function=False*, *nsamples='auto'*, *features=None*, *classes=None*, *nclusters=10*, *show_progress=False*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global. :param evaluation_examples: A matrix of feature vector examples (# examples x # features) on which

to explain the model's output.

Parameters

- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the function locally by using SHAP's KernelExplainer. :param evaluation_examples: A matrix of feature vector examples (# examples x # features) on which

to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

GPU version of the Kernel Explainer for explaining black box models or functions.

Uses cuml's GPU Kernel SHAP. https://docs.rapids.ai/api/cuml/stable/api.html#shap-kernel-explainer

Characteristics of the GPU version:

- Unlike the SHAP package, `nsamples` is a parameter at the initialization of the explainer and there is a small initialization time.
- Only tabular data is supported for now, via passing the background dataset explicitly.
- Sparse data support is planned for the near future.
- Further optimizations are in progress. For example, if the background dataset has constant value columns and the observation has the same value in some entries, the number of evaluations of the function can be reduced.

Parameters

- **model** (*object*) – Function that takes a matrix of samples (n_samples, n_features) and computes the output for those samples with shape (n_samples).
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation, which will speed up the explanation process when number of features is large and the user already knows the set of interested features. The subset can be the top-k features from the model summary.
- **nsamples** (*'auto' or int*) – int (default = 2 * data.shape[1] + 2048) 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.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **show_progress** (*bool*) – Default to 'False'. Determines whether to display the explanation status bar when using shap_values from the cuML KernelExplainer.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas. If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler. Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many::

[(["col1", "col2"], my_own_transformer)

]

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations.
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

class interpret_community.shap.**KernelExplainer**(*model*, *initialization_examples*, *is_function=False*, *explain_subset=None*, *nsamples='auto'*, *features=None*, *classes=None*, *nclusters=10*, *show_progress=True*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the function locally by using SHAP's KernelExplainer.

Parameters **evaluation_examples** (*DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

The Kernel Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The model to explain or function if is_function is True. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **is_function** (*bool*) – Default is False. Set to True if passing function instead of a model.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation, which will speed up the explanation process when number of features is large and the user already knows the set of interested features. The subset can be the top-k features from the model summary.
- **nsamples** (*'auto' or int*) – Default to 'auto'. Number of times to re-evaluate the model when explaining each prediction. More samples lead to lower variance estimates of the feature importance values, but incur more computation cost. When 'auto' is provided, the number of samples is computed according to a heuristic rule.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **show_progress** (*bool*) – Default to 'True'. Determines whether to display the explanation status bar when using shap_values from the KernelExplainer.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations

are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations.
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

class interpret_community.shap.**LinearExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, ***kwargs*)

Bases: *interpret_community.common.structured_model_explainer.StructuredInitModelExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the model by using SHAP's linear explainer.

Parameters **evaluation_examples** (*DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'specific'

Defines the LinearExplainer for returning explanations for linear models.

Parameters

- **model** (*(coef, intercept) or sklearn.linear_model.**) – The linear model to explain as the coefficient and intercept or scikit learn model.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer,

LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations

class interpret_community.shap.**TreeExplainer**(*model*, *explain_subset=None*, *features=None*, *classes=None*, *shap_values_output=ShapValuesOutput.DEFAULT*, *transformations=None*, *allow_all_transformations=False*, ***kwargs*)

Bases: *interpret_community.common.structured_model_explainer.PureStructuredModelExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the model by using shap's tree explainer.

Parameters **evaluation_examples** (*DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'specific'

The TreeExplainer for returning explanations for tree-based models.

Parameters

- **model** (*lightgbm, xgboost or scikit-learn tree model*) – The tree model to explain.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **shap_values_output** (interpret_community.common.constants.ShapValuesOutput) – The type of the output when using TreeExplainer. Currently only types 'default' and 'probability' are supported. If 'probability' is specified, then the raw log-odds values are approximately scaled to probabilities from the TreeExplainer.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
```

(continues on next page)

(continued from previous page)

```
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations

Submodules

interpret_community.shap.deep_explainer module

Defines an explainer for DNN models.

class interpret_community.shap.deep_explainer.**DeepExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *nclusters=10*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, *is_classifier=None*, ***kwargs*)

Bases: *interpret_community.common.structured_model_explainer.StructuredInitModelExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (*interpret_community.common.policy.SamplingPolicy*) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the model by using SHAP's deep explainer.

Parameters **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'specific'

An explainer for DNN models, implemented using shap's DeepExplainer, supports TensorFlow and PyTorch.

Parameters

- **model** (*PyTorch or TensorFlow model*) – The DNN model to explain.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model.

class interpret_community.shap.deep_explainer.**logger_redirector**(*module_logger*)

Bases: object

A redirector for system error output to logger.

close()

flush()

write(*data*)

Write the given data to logger.

Parameters **data** (*str*) – The data to write to logger.

interpret_community.shap.gpu_kernel_explainer module

Defines the GPUKernelExplainer for computing explanations on black box models or functions.

class interpret_community.shap.gpu_kernel_explainer.**GPUKernelExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *is_function=False*, *nsamples='auto'*, *features=None*, *classes=None*, *nclusters=10*, *show_progress=False*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global. :param evaluation_examples: A matrix of feature vector examples (# examples x # features) on which

to explain the model's output.

Parameters

- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the function locally by using SHAP's KernelExplainer. :param evaluation_examples: A matrix of feature vector examples (# examples x # features) on which

to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

GPU version of the Kernel Explainer for explaining black box models or functions.

Uses cuml's GPU Kernel SHAP. https://docs.rapids.ai/api/cuml/stable/api.html#shap-kernel-explainer

Characteristics of the GPU version:

- Unlike the SHAP package, `nsamples` is a parameter at the initialization of the explainer and there is a small initialization time.
- Only tabular data is supported for now, via passing the background dataset explicitly.
- Sparse data support is planned for the near future.
- Further optimizations are in progress. For example, if the background dataset has constant value columns and the observation has the same value in some entries, the number of evaluations of the function can be reduced.

Parameters

- **model** (*object*) – Function that takes a matrix of samples (n_samples, n_features) and computes the output for those samples with shape (n_samples).
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation, which will speed up the explanation process when number of features is large and the user already knows the set of interested features. The subset can be the top-k features from the model summary.
- **nsamples** (*'auto' or int*) – int (default = 2 * data.shape[1] + 2048) 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.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **show_progress** (*bool*) – Default to 'False'. Determines whether to display the explanation status bar when using shap_values from the cuML KernelExplainer.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas. If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer,

KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler. Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

An example of a transformation that would raise an error since it cannot be interpreted as one to many

[(“col1”, “col2”], my_own_transformer)

]

The last example would not work since the interpret-community package can’t determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations.
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

interpret_community.shap.kernel_explainer module

Defines the KernelExplainer for computing explanations on black box models or functions.

class interpret_community.shap.kernel_explainer.**KernelExplainer**(*model*, *initialization_examples*, *is_function=False*, *explain_subset=None*, *nsamples='auto'*, *features=None*, *classes=None*, *nclusters=10*, *show_progress=True*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, ***kwargs*)

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the function locally by using SHAP's KernelExplainer.

Parameters **evaluation_examples** (*DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

The Kernel Explainer for explaining black box models or functions.

Parameters

- **model** (*object*) – The model to explain or function if is_function is True. A model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **is_function** (*bool*) – Default is False. Set to True if passing function instead of a model.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation, which will speed up the explanation process when number of features is large and the user already knows the set of interested features. The subset can be the top-k features from the model summary.
- **nsamples** (*'auto' or int*) – Default to 'auto'. Number of times to re-evaluate the model when explaining each prediction. More samples lead to lower variance estimates of the feature importance values, but incur more computation cost. When 'auto' is provided, the number of samples is computed according to a heuristic rule.
- **features** (*list[str]*) – A list of feature names.

- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **nclusters** (*int*) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **show_progress** (*bool*) – Default to 'True'. Determines whether to display the explanation status bar when using shap_values from the KernelExplainer.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

  The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.
- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations.
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

interpret_community.shap.kwargs_utils module

Defines utilities for handling kwargs on SHAP-based explainers.

interpret_community.shap.linear_explainer module

Defines the LinearExplainer for returning explanations for linear models.

class interpret_community.shap.linear_explainer.**LinearExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, ***kwargs*)

Bases: *interpret_community.common.structured_model_explainer.StructuredInitModelExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (*interpret_community.common.policy.SamplingPolicy*) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the model by using SHAP's linear explainer.

Parameters **evaluation_examples** (*DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'specific'

Defines the LinearExplainer for returning explanations for linear models.

Parameters

- **model** (*(coef, intercept) or sklearn.linear_model.**) – The linear model to explain as the coefficient and intercept or scikit learn model.
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

  The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.
- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations

interpret_community.shap.tree_explainer module

Defines the TreeExplainer for returning explanations for tree-based models.

class interpret_community.shap.tree_explainer.**TreeExplainer**(*model*, *explain_subset=None*, *features=None*, *classes=None*, *shap_values_output=ShapValuesOutput.DEFAULT*, *transformations=None*, *allow_all_transformations=False*, ***kwargs*)

Bases: *interpret_community.common.structured_model_explainer.PureStructuredModelExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (*interpret_community.common.policy.SamplingPolicy*) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation which also has the properties of LocalExplanation and ExpectedValuesMixin. If the model is a classifier, it will have the properties of PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Explain the model by using shap's tree explainer.

Parameters **evaluation_examples** (*DatasetWrapper*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation which also has the properties of ExpectedValuesMixin. If the model is a classifier, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'specific'

The TreeExplainer for returning explanations for tree-based models.

Parameters

- **model** (*lightgbm, xgboost or scikit-learn tree model*) – The tree model to explain.

- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation. The subset can be the top-k features from the model summary.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **shap_values_output** (interpret_community.common.constants.ShapValuesOutput) – The type of the output when using TreeExplainer. Currently only types 'default' and 'probability' are supported. If 'probability' is specified, then the raw log-odds values are approximately scaled to probabilities from the TreeExplainer.
- **transformations** (*sklearn.compose.ColumnTransformer* or *list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If you are using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. You can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

  The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.
- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations

interpret_community.widget package

Module for Explanation Dashboard widget.

class interpret_community.widget.**ExplanationDashboard**(*explanation*, *model=None*, ***, *dataset=None*, *true_y=None*, *classes=None*, *features=None*, *port=None*, *datasetX=None*, *trueY=None*, *locale=None*, *public_ip=None*, *with_credentials=False*, *use_cdn=None*)

Bases: `object`

DEPRECATED Explanation Dashboard class, please use the Explanation Dashboard from raiwidgets package instead.

Since this class is deprecated it will no longer display the widget. Please install raiwidgets from pypi by running: pip install –upgrade raiwidgets The dashboard can be run with the same parameters in the new namespace: from raiwidgets import ExplanationDashboard

Parameters

- **explanation** (*ExplanationMixin*) – An object that represents an explanation.
- **model** (*object*) – An object that represents a model. It is assumed that for the classification case it has a method of predict_proba() returning the prediction probabilities for each class and for the regression case a method of predict() returning the prediction value.
- **dataset** (*numpy.ndarray or list[][]*) – A matrix of feature vector examples (# examples x # features), the same samples used to build the explanation. Overwrites any existing dataset on the explanation object. Must have fewer than 10000 rows and fewer than 1000 columns.
- **datasetX** (*numpy.ndarray or list[][]*) – Alias of the dataset parameter. If dataset is passed, this will have no effect. Must have fewer than 10000 rows and fewer than 1000 columns.
- **true_y** (*numpy.ndarray or list[]*) – The true labels for the provided dataset. Overwrites any existing dataset on the explanation object.
- **classes** (*numpy.ndarray or list[]*) – The class names.
- **features** (*numpy.ndarray or list[]*) – Feature names.
- **port** (*int*) – The port to use on locally hosted service.
- **use_cdn** (*bool*) – Deprecated. Whether to load latest dashboard script from cdn, fall back to local script if False. .. deprecated:: 0.15.2

  Deprecated since 0.15.2, cdn has been removed. Setting parameter to True or False will trigger warning.
- **public_ip** (*str*) – Optional. If running on a remote vm, the external public ip address of the VM.
- **with_credentials** (*bool*) – Optional. If running on a remote vm, sets up CORS policy both on client and server.

Submodules

interpret_community.widget.explanation_dashboard module

Defines the DEPRECATED Explanation dashboard class.

class interpret_community.widget.explanation_dashboard.**ExplanationDashboard**(*explanation*, *model=None*, *, *dataset=None*, *true_y=None*, *classes=None*, *features=None*, *port=None*, *datasetX=None*, *trueY=None*, *locale=None*, *public_ip=None*, *with_credentials=False*, *use_cdn=None*)

Bases: object

DEPRECATED Explanation Dashboard class, please use the Explanation Dashboard from raiwidgets package instead.

Since this class is deprecated it will no longer display the widget. Please install raiwidgets from pypi by running: pip install –upgrade raiwidgets The dashboard can be run with the same parameters in the new namespace: from raiwidgets import ExplanationDashboard

Parameters

- **explanation** (*ExplanationMixin*) – An object that represents an explanation.
- **model** (*object*) – An object that represents a model. It is assumed that for the classification case it has a method of predict_proba() returning the prediction probabilities for each class and for the regression case a method of predict() returning the prediction value.
- **dataset** (*numpy.ndarray or list[][]*) – A matrix of feature vector examples (# examples x # features), the same samples used to build the explanation. Overwrites any existing dataset on the explanation object. Must have fewer than 10000 rows and fewer than 1000 columns.
- **datasetX** (*numpy.ndarray or list[][]*) – Alias of the dataset parameter. If dataset is passed, this will have no effect. Must have fewer than 10000 rows and fewer than 1000 columns.
- **true_y** (*numpy.ndarray or list[]*) – The true labels for the provided dataset. Overwrites any existing dataset on the explanation object.
- **classes** (*numpy.ndarray or list[]*) – The class names.
- **features** (*numpy.ndarray or list[]*) – Feature names.
- **port** (*int*) – The port to use on locally hosted service.
- **use_cdn** (*bool*) – Deprecated. Whether to load latest dashboard script from cdn, fall back to local script if False. .. deprecated:: 0.15.2

  Deprecated since 0.15.2, cdn has been removed. Setting parameter to True or False will trigger warning.

- **public_ip** (*str*) – Optional. If running on a remote vm, the external public ip address of the VM.
- **with_credentials** (*bool*) – Optional. If running on a remote vm, sets up CORS policy both on client and server.

Submodules

interpret_community.tabular_explainer module

Defines the tabular explainer meta-api for returning the best explanation result based on the given model.

class interpret_community.tabular_explainer.**TabularExplainer**(*model*, *initialization_examples*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *model_task=ModelTask.Unknown*, *use_gpu=False*, ***kwargs*)

Bases: *interpret_community.common.base_explainer.BaseExplainer*

available_explanations = ['global', 'local']

explain_global(*evaluation_examples*, *sampling_policy=None*, *include_local=True*, *batch_size=100*)

Globally explains the black box model or function.

Parameters

- **evaluation_examples** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.
- **sampling_policy** (interpret_community.common.policy.SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object. It is guaranteed to be a GlobalExplanation. If SHAP is used for the explanation, it will also have the properties of a LocalExplanation and the ExpectedValuesMixin. If the model does classification, it will have the properties of the PerClassMixin.

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

Locally explains the black box model or function.

Parameters **evaluation_examples** (*numpy.ndarray* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) on which to explain the model's output.

Returns A model explanation object. It is guaranteed to be a LocalExplanation. If SHAP is used for the explanation, it will also have the properties of the ExpectedValuesMixin. If the model does classification, it will have the properties of the ClassesMixin.

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

The tabular explainer meta-api for returning the best explanation result based on the given model.

Parameters

- **model** (*object*) – The model or pipeline to explain. A model that implements sklearn.predict() or sklearn.predict_proba() or pipeline function that accepts a 2d ndarray
- **initialization_examples** (*numpy.ndarray or pandas.DataFrame or scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **explain_subset** (*list[int]*) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation, which will speed up the explanation process when number of features is large and the user already knows the set of interested features. The subset can be the top-k features from the model summary. This argument is not supported when transformations are set.
- **features** (*list[str]*) – A list of feature names.
- **classes** (*list[str]*) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (*sklearn.compose.ColumnTransformer or list[tuple]*) – sklearn.compose.ColumnTransformer or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: https://github.com/scikit-learn-contrib/sklearn-pandas.

  If the user is using a transformation that is not in the list of sklearn.preprocessing transformations that are supported by the interpret-community package, then this parameter cannot take a list of more than one column as input for the transformation. A user can use the following sklearn.preprocessing transformations with a list of columns since these are already one to many or one to one: Binarizer, KBinsDiscretizer, KernelCenterer, LabelEncoder, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler.

  Examples for transformations that work:

```
[
    (["col1", "col2"], sklearn_one_hot_encoder),
    (["col3"], None) #col3 passes as is
]
[
    (["col1"], my_own_transformer),
    (["col2"], my_own_transformer),
]
```

  An example of a transformation that would raise an error since it cannot be interpreted as one to many:

```
[
    (["col1", "col2"], my_own_transformer)
]
```

The last example would not work since the interpret-community package can't determine whether my_own_transformer gives a many to many or one to many mapping when taking a sequence of columns.

- **allow_all_transformations** (*bool*) – Allow many to many and many to one transformations

interpret_community.version module

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

A

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W