

---

# **interpret-community**

***Release 0.27.0***

**Microsoft**

**Sep 13, 2022**



# CONTENTS

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Indices and tables</b>	<b>107</b>
	<b>Python Module Index</b>	<b>109</b>
	<b>Index</b>	<b>111</b>



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/>.



## 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 <b>any model</b> .	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 <b>any model</b> . 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 <b>trees and ensembles of trees</b> .	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 <b>linear model</b> , optionally accounting for inter-feature correlations.	Model-specific
Mimic Explainer (Global Surrogate)	Mimic explainer is based on the idea of training <b>global surrogate models</b> to mimic blackbox models. A global surrogate model is an intrinsically interpretable model that is trained to approximate the predictions of <b>any black box model</b> 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 <b>any underlying model</b> 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 explainer 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 PFIE explainer 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 PFIEExplainer 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:

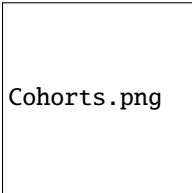
- Model Performance
- Data Explorer
- Aggregate Feature Importance
- Individual Feature Importance and What-If

---

**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.

---

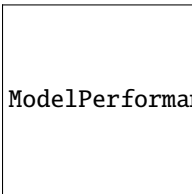
You can further create custom cohorts (subgroups of your dataset) to explore the insights across different subgroups (e.g., women vs. men). The created cohorts can contain more than one filter (e.g., age < 30 and sex = female) and will be visible from all of the four tabs. The following sections demonstrate 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.



Cohorts.png

### 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.



ModelPerformance.png

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.



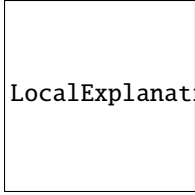
1.9.4 Individual feature importance (local explanation) and what-if

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.





LocalExplanation.png

## 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 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 Sampling-Policy 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 `Per-ClassMixin`.

**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 proba-
    bilities 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](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](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](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 meta-data 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 PFIE explainer.

**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,
                                                                                          ini-
                                                                                          tial-
                                                                                          iza-
                                                                                          tion_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 preexisting 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 preexisting 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 Sampling-Policy 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 LIMExplainer.
- **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 Sampling-Policy 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.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 decision 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 do.

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 do.

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 do.

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 do.

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 do.

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  $(\text{clip}(\text{decision\_function}(X), 1, 1) + 1) / 2$ . 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,  
 ran-  
 dom\_state=123,  
 shap\_values\_output=ShapValuesOut  
 classifica-  
 tion=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,
                                                                           ran-
                                                                           dom_state=123,
                                                                           classifica-
                                                                           tion=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 onvsrest 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  $(\text{clip}(\text{decision\_function}(X), 1, 1) + 1) / 2$ . 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,
                                                                              ran-
                                                                              dom_state=123,
                                                                              shap_values_output=ShapValue
                                                                              classifica-
                                                                              tion=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 do.

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 do.

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 do.

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 do.

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 do.

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 PFIE explainer.
- **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 PFIE explainer.

**class** `interpret_community.permutation.metric_constants.MetricConstants`(*value*)

Bases: `str`, `enum.Enum`

The metric to use for PFIE explainer.

**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 PFIE explainer for computing global explanations on black box models or functions.

The PFIE explainer 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,
                                                                           ex-
                                                                           plain_subset=None,
                                                                           features=None,
                                                                           classes=None,
                                                                           transforma-
                                                                           tions=None,
                                                                           al-
                                                                           low_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.

`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 Sampling-Policy 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 =  $2 * \text{data.shape}[1] + 2048$ ) 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 ( $10 \times \text{nclusters}$ ), those examples will be summarized with k-means where  $k = \text{nclusters}$ .
- **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.kwarg\_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[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[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[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[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[int]*) – The true labels for the provided dataset. Overwrites any existing dataset on the explanation object.
- **classes** (*numpy.ndarray* or *list[str]*) – The class names.
- **features** (*numpy.ndarray* or *list[str]*) – 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

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### i

`interpret_community`, 14  
`interpret_community.adapter`, 16  
`interpret_community.adapter.explanation_adapter`, 17  
`interpret_community.common`, 18  
`interpret_community.common.aggregate`, 18  
`interpret_community.common.base_explainer`, 19  
`interpret_community.common.blackbox_explainer`, 19  
`interpret_community.common.chained_identity`, 20  
`interpret_community.common.constants`, 21  
`interpret_community.common.error_handling`, 27  
`interpret_community.common.exception`, 27  
`interpret_community.common.explanation_utils`, 27  
`interpret_community.common.gpu_kmeans`, 27  
`interpret_community.common.metrics`, 28  
`interpret_community.common.model_summary`, 28  
`interpret_community.common.model_wrapper`, 18  
`interpret_community.common.policy`, 29  
`interpret_community.common.progress`, 30  
`interpret_community.common.serialization_utils`, 30  
`interpret_community.common.structured_model_explainer`, 30  
`interpret_community.common.warnings_suppressor`, 31  
`interpret_community.dataset`, 31  
`interpret_community.dataset.dataset_wrapper`, 31  
`interpret_community.dataset.decorator`, 31  
`interpret_community.explanation`, 32  
`interpret_community.explanation.explanation`, 32  
`interpret_community.explanation.serialization`, 40  
`interpret_community.lime`, 40  
`interpret_community.lime.lime_explainer`, 43  
`interpret_community.mimic`, 45  
`interpret_community.mimic.mimic_explainer`, 70  
`interpret_community.mimic.model_distill`, 73  
`interpret_community.mimic.models`, 48  
`interpret_community.mimic.models.explainable_model`, 59  
`interpret_community.mimic.models.lightgbm_model`, 59  
`interpret_community.mimic.models.linear_model`, 61  
`interpret_community.mimic.models.tree_model`, 67  
`interpret_community.mimic.models.tree_model_utils`, 70  
`interpret_community.mlflow`, 73  
`interpret_community.mlflow.mlflow`, 74  
`interpret_community.permutation`, 75  
`interpret_community.permutation.metric_constants`, 77  
`interpret_community.permutation.permutation_importance`, 77  
`interpret_community.shap`, 80  
`interpret_community.shap.deep_explainer`, 90  
`interpret_community.shap.gpu_kernel_explainer`, 93  
`interpret_community.shap.kernel_explainer`, 95  
`interpret_community.shap.kwargs_utils`, 98  
`interpret_community.shap.linear_explainer`, 98  
`interpret_community.shap.tree_explainer`, 100  
`interpret_community.tabular_explainer`, 104  
`interpret_community.version`, 106  
`interpret_community.widget`, 102  
`interpret_community.widget.explanation_dashboard`, 103



## INDEX

### A

- `add_explain_global_method()` (in module `pret_community.common.aggregate`), 18
- `add_from_get_model_summary()` (inter-  
`pret_community.common.model_summary.ModelSummary`  
method), 28
- `add_from_get_model_summary()` (inter-  
`pret_community.common.ModelSummary`  
method), 18
- `add_prepare_function_and_summary_method()`  
(in module `pret_community.common.blackbox_explainer`),  
20
- `ALLOW_ALL_TRANSFORMATIONS` (inter-  
`pret_community.common.constants.MimicSerializationConstants`  
attribute), 25
- `allow_eval_sampling` (inter-  
`pret_community.common.policy.SamplingPolicy`  
property), 29
- `Attributes` (class in inter-  
`pret_community.common.constants`), 21
- `AUTO` (`pret_community.common.constants.Defaults`  
attribute), 21
- `available_explanations` (inter-  
`pret_community.lime.lime_explainer.LIMEExplainer`  
attribute), 43
- `available_explanations` (inter-  
`pret_community.lime.LIMEExplainer` at-  
tribute), 40
- `available_explanations` (inter-  
`pret_community.mimic.mimic_explainer.MimicExplainer`  
attribute), 70
- `available_explanations` (inter-  
`pret_community.mimic.MimicExplainer` at-  
tribute), 45
- `available_explanations` (inter-  
`pret_community.mimic.models.DecisionTreeExplainableModel`  
attribute), 48
- `available_explanations` (inter-  
`pret_community.mimic.models.LGBMExplainableModel`  
attribute), 51
- `available_explanations` (inter-  
`pret_community.mimic.models.lightgbm_model.LGBMExplainableModel`  
attribute), 59
- `available_explanations` (inter-  
`pret_community.mimic.models.linear_model.LinearExplainableModel`  
attribute), 61
- `available_explanations` (inter-  
`pret_community.mimic.models.linear_model.SGDExplainableModel`  
attribute), 64
- `available_explanations` (inter-  
`pret_community.mimic.models.LinearExplainableModel`  
attribute), 53
- `available_explanations` (inter-  
`pret_community.mimic.models.SGDExplainableModel`  
attribute), 56
- `available_explanations` (inter-  
`pret_community.mimic.models.tree_model.DecisionTreeExplainableModel`  
attribute), 67
- `available_explanations` (inter-  
`pret_community.permutation.permutation_importance.PFIExplainer`  
attribute), 78
- `available_explanations` (inter-  
`pret_community.permutation.PFIExplainer`  
attribute), 75
- `available_explanations` (inter-  
`pret_community.shap.deep_explainer.DeepExplainer`  
attribute), 90
- `available_explanations` (inter-  
`pret_community.shap.DeepExplainer` at-  
tribute), 80
- `available_explanations` (inter-  
`pret_community.shap.gpu_kernel_explainer.GPUKernelExplainer`  
attribute), 93
- `available_explanations` (inter-  
`pret_community.shap.GPUKernelExplainer`  
attribute), 82
- `available_explanations` (inter-  
`pret_community.shap.kernel_explainer.KernelExplainer`  
attribute), 95
- `available_explanations` (inter-  
`pret_community.shap.KernelExplainer` at-  
tribute), 84
- `available_explanations` (inter-

pret\_community.shap.linear\_explainer.LinearExplainer (attribute), 98

available\_explanations (interpret\_community.shap.LinearExplainer attribute), 86

available\_explanations (interpret\_community.shap.tree\_explainer.TreeExplainer attribute), 100

available\_explanations (interpret\_community.shap.TreeExplainer attribute), 88

available\_explanations (interpret\_community.tabular\_explainer.TabularExplainer attribute), 104

available\_explanations (interpret\_community.TabularExplainer attribute), 14

AVERAGE\_PRECISION\_SCORE (interpret\_community.permutation.metric\_constants.MeanCoverage attribute), 77

## B

BASE\_VALUE (interpret\_community.common.constants.InterpretData attribute), 24

BaseExplainableModel (class in interpret\_community.mimic.models), 48

BaseExplainableModel (class in interpret\_community.mimic.models.explainable\_model), 59

BaseExplainer (class in interpret\_community.common.base\_explainer), 19

BaseExplanation (class in interpret\_community.explanation.explanation), 32

BATCH\_SIZE (interpret\_community.common.constants.ExplainParams attribute), 21

BLACKBOX (interpret\_community.common.constants.Extension attribute), 24

BlackBoxExplainer (class in interpret\_community.common.blackbox\_explainer), 19

BlackBoxMixin (class in interpret\_community.common.blackbox\_explainer), 20

## C

CATEGORICAL\_FEATURE (interpret\_community.common.constants.LightGBMParams attribute), 25

ChainedIdentity (class in interpret\_community.common.chained\_identity), 20

CLASSES (interpret\_community.common.constants.ExplainParams attribute), 21

CLASSES (interpret\_community.common.constants.ExplanationParams attribute), 24

classes (interpret\_community.explanation.explanation.ClassesMixin property), 34

ClassesMixin (class in interpret\_community.explanation.explanation), 33

CLASSIFICATION (interpret\_community.common.constants.ExplainParams attribute), 21

CLASSIFICATION (interpret\_community.common.constants.ExplainType attribute), 23

Classification (interpret\_community.common.constants.ModelTask attribute), 26

close() (interpret\_community.shap.deep\_explainer.logger\_redirector method), 92

CPU0 (interpret\_community.common.constants.Tensorflow attribute), 27

create\_global() (interpret\_community.adapter.explanation\_adapter.ExplanationAdapter method), 17

create\_global() (interpret\_community.adapter.ExplanationAdapter method), 16

create\_local() (interpret\_community.adapter.explanation\_adapter.ExplanationAdapter method), 17

create\_local() (interpret\_community.adapter.ExplanationAdapter method), 17

CSR\_FORMAT (interpret\_community.common.constants.Scipy attribute), 26

## D

Data (class in interpret\_community.common.gpu\_kmeans), 27

DATA (interpret\_community.common.constants.ExplainType attribute), 23

data() (interpret\_community.explanation.explanation.BaseExplanation method), 33

data() (interpret\_community.explanation.explanation.ExpectedValuesMixin method), 34

data() (interpret\_community.explanation.explanation.GlobalExplanation method), 35

data() (interpret\_community.explanation.explanation.LocalExplanation method), 37

dcg() (in module interpret\_community.common.metrics), 28

DecisionTreeExplainableModel (class in interpret\_community.mimic.models), 48

**DecisionTreeExplainableModel** (class in `interpret_community.mimic.models.tree_model`), 67  
**DeepExplainer** (class in `interpret_community.shap`), 80  
**DeepExplainer** (class in `interpret_community.shap.deep_explainer`), 90  
**DEFAULT** (`interpret_community.common.constants.ShapValueExpected` attribute), 26  
**DEFAULT\_BATCH\_SIZE** (`interpret_community.common.constants.Defaults` attribute), 21  
**Defaults** (class in `interpret_community.common.constants`), 21  
**DenseData** (class in `interpret_community.common.gpu_kmeans`), 27  
**DNNFramework** (class in `interpret_community.common.constants`), 21  
**Dynamic** (class in `interpret_community.common.constants`), 21  
**E**  
**EN** (`interpret_community.common.constants.Spacy` attribute), 27  
**enum\_properties** (`interpret_community.common.constants.LightGBMSerializableConstants` attribute), 25  
**enum\_properties** (`interpret_community.common.constants.MimicSerializationConstants` attribute), 25  
**EVAL\_DATA** (`interpret_community.common.constants.ExplainParams` attribute), 21  
**EVAL\_Y\_PRED** (`interpret_community.common.constants.ExplainParams` attribute), 21  
**EVAL\_Y\_PRED\_PROBA** (`interpret_community.common.constants.ExplainParams` attribute), 21  
**EXAMPLES** (`interpret_community.common.constants.SKLearn` attribute), 26  
**EXPECTED\_VALUE** (`interpret_community.common.constants.Attributes` attribute), 21  
**EXPECTED\_VALUES** (`interpret_community.common.constants.ExplainParams` attribute), 21  
**EXPECTED\_VALUES** (`interpret_community.common.constants.ExplanationParams` attribute), 24  
**expected\_values** (`interpret_community.explanation.explanation.ExpectedValuesMixin` property), 34  
**expected\_values** (`interpret_community.mimic.models.BaseExplainableModel` property), 48  
**expected\_values** (`interpret_community.mimic.models.DecisionTreeExplainableModel` property), 48  
**expected\_values** (`interpret_community.mimic.models.explainable_model.BaseExplainableModel` property), 59  
**expected\_values** (`interpret_community.mimic.models.LGBMExplainableModel` property), 52  
**expected\_values** (`interpret_community.mimic.models.lightgbm_model.LGBMExplainableModel` property), 59  
**expected\_values** (`interpret_community.mimic.models.linear_model.LinearExplainableModel` property), 61  
**expected\_values** (`interpret_community.mimic.models.linear_model.SGDExplainableModel` property), 64  
**expected\_values** (`interpret_community.mimic.models.LinearExplainableModel` property), 53  
**expected\_values** (`interpret_community.mimic.models.SGDExplainableModel` property), 56  
**expected\_values** (`interpret_community.mimic.models.tree_model.DecisionTreeExplainableModel` property), 67  
**ExpectedValuesMixin** (class in `interpret_community.explanation.explanation`), 34  
**EXPLAIN** (`interpret_community.common.constants.ExplainType` attribute), 23  
**explain\_global()** (`interpret_community.common.base_explainer.GlobalExplainer` method), 19  
**explain\_global()** (`interpret_community.lime.lime_explainer.LIMEExplainer` method), 43  
**explain\_global()** (`interpret_community.lime.LIMEExplainer` method), 41  
**explain\_global()** (`interpret_community.mimic.mimic_explainer.MimicExplainer` method), 70  
**explain\_global()** (`interpret_community.mimic.MimicExplainer` method), 45  
**explain\_global()** (`interpret_community.mimic.models.BaseExplainableModel` method), 48  
**explain\_global()** (`interpret_community.mimic.models.DecisionTreeExplainableModel` method), 48  
**explain\_global()** (inter-

Method	Module	Method	Module
<code>pret_community.mimic.models.explainable_model.BaseExplainer</code>	<code>pret_community.mimic.models.explainable_model.BaseExplainer</code>	<code>pret_community.shap.tree_explainer.TreeExplainer</code>	<code>pret_community.shap.tree_explainer.TreeExplainer</code>
<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.LGBMExplainer method), 52</code>	<code>explain_global()</code>	<code>(inter-pret_community.shap.TreeExplainer method), 88</code>
<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.lightgbm_model.LGBMExplainer method), 59</code>	<code>explain_global()</code>	<code>(inter-pret_community.shap.tabular_explainer.TabularExplainer method), 104</code>
<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.linear_model.LinearExplainer method), 61</code>	<code>explain_global()</code>	<code>(inter-pret_community.shap.TabularExplainer method), 14</code>
<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.linear_model.SGDExplainer method), 64</code>	<code>explain_global()</code>	<code>(inter-pret_community.common.base_explainer.LocalExplainer method), 19</code>
<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.LinearExplainer method), 53</code>	<code>explain_global()</code>	<code>(inter-pret_community.lime.lime_explainer.LIMEExplainer method), 43</code>
<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.SGDExplainer method), 56</code>	<code>explain_global()</code>	<code>(inter-pret_community.lime.LIMEExplainer method), 41</code>
<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.tree_model.DecisionTreeExplainer method), 67</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.mimic_explainer.MimicExplainer method), 71</code>
<code>explain_global()</code>	<code>(inter-pret_community.permutation.permutation_importance.PFIExplainer method), 78</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.MimicExplainer method), 46</code>
<code>explain_global()</code>	<code>(inter-pret_community.permutation.PFIExplainer method), 75</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.BaseExplainer method), 48</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.deep_explainer.DeepExplainer method), 90</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.DecisionTreeExplainer method), 49</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.DeepExplainer method), 80</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.explainable_model.BaseExplainer method), 59</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.gpu_kernel_explainer.GPUKernelExplainer method), 93</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.LGBMExplainer method), 52</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.GPUKernelExplainer method), 82</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.lightgbm_model.LGBMExplainer method), 60</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.kernel_explainer.KernelExplainer method), 95</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.linear_model.LinearExplainer method), 61</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.KernelExplainer method), 84</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.linear_model.SGDExplainer method), 64</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.linear_explainer.LinearExplainer method), 98</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.LinearExplainer method), 53</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.LinearExplainer method), 86</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.SGDExplainer method), 56</code>
<code>explain_global()</code>	<code>(inter-pret_community.shap.LinearExplainer method), 86</code>	<code>explain_global()</code>	<code>(inter-pret_community.mimic.models.SGDExplainer method), 56</code>



pret\_community.mimic.models.tree\_model.DecisionTreeExplainer, 67

pret\_community.mimic.models.lightgbm\_model.LGBMExplainer, 60

explain\_local() (inter-pret\_community.shap.deep\_explainer.DeepExplainer method), 91

explain\_local() (inter-pret\_community.shap.DeepExplainer method), 80

explain\_local() (inter-pret\_community.shap.gpu\_kernel\_explainer.GPUKernelExplainer method), 93

explain\_local() (inter-pret\_community.shap.GPUKernelExplainer method), 82

explain\_local() (inter-pret\_community.shap.kernel\_explainer.KernelExplainer method), 96

explain\_local() (inter-pret\_community.shap.KernelExplainer method), 85

explain\_local() (inter-pret\_community.shap.linear\_explainer.LinearExplainer method), 98

explain\_local() (inter-pret\_community.shap.LinearExplainer method), 87

explain\_local() (inter-pret\_community.shap.tree\_explainer.TreeExplainer method), 100

explain\_local() (inter-pret\_community.shap.TreeExplainer method), 89

explain\_local() (inter-pret\_community.tabular\_explainer.TabularExplainer method), 104

explain\_local() (inter-pret\_community.TabularExplainer method), 15

EXPLAIN\_SUBSET (inter-pret\_community.common.constants.ExplainParameters attribute), 21

explainable\_model\_type() (inter-pret\_community.mimic.models.BaseExplainerModel static method), 48

explainable\_model\_type() (inter-pret\_community.mimic.models.DecisionTreeExplainerModel static method), 49

explainable\_model\_type() (inter-pret\_community.mimic.models.explainable\_model.ExplainerModel static method), 59

explainable\_model\_type() (inter-pret\_community.mimic.models.LGBMExplainerModel static method), 52

explainable\_model\_type() (inter-pret\_community.mimic.models.linear\_model.LinearExplainerModel static method), 61

explainable\_model\_type() (inter-pret\_community.mimic.models.linear\_model.LinearExplainerModel static method), 61

explainable\_model\_type() (inter-pret\_community.mimic.models.LinearExplainerModel static method), 53

explainable\_model\_type() (inter-pret\_community.mimic.models.tree\_model.DecisionTreeExplainerModel static method), 67

EXPLAINABLE\_MODEL\_TYPE (class in inter-pret\_community.common.constants), 23

EXPLAINED\_VARIANCE\_SCORE (inter-pret\_community.permutation.metric\_constants.MetricConstants attribute), 77

EXPLAINER (interpret\_community.common.constants.ExplainType attribute), 23

explainer\_type (inter-pret\_community.lime.lime\_explainer.LIMEExplainer attribute), 43

explainer\_type (inter-pret\_community.lime.LIMEExplainer attribute), 41

explainer\_type (inter-pret\_community.mimic.mimic\_explainer.MimicExplainer attribute), 71

explainer\_type (inter-pret\_community.mimic.MimicExplainer attribute), 46

explainer\_type (inter-pret\_community.mimic.models.DecisionTreeExplainerModel attribute), 49

explainer\_type (inter-pret\_community.mimic.models.LGBMExplainerModel attribute), 52

explainer\_type (inter-pret\_community.mimic.models.lightgbm\_model.LGBMExplainerModel attribute), 60

explainer\_type (inter-pret\_community.mimic.models.linear\_model.LinearExplainerModel attribute), 61

explainer\_type (inter-pret\_community.mimic.models.linear\_model.SGDExplainerModel attribute), 64

EXPLAINER\_MODEL\_TYPE (class in inter-pret\_community.mimic.models.LinearExplainerModel static method), 54

EXPLAINER\_MODEL\_TYPE (class in inter-pret\_community.mimic.models.SGDExplainerModel static method), 56

EXPLAINER\_MODEL\_TYPE (class in inter-pret\_community.mimic.models.tree\_model.DecisionTreeExplainerModel static method), 67

**explainer\_type** (interpret\_community.permutation.permutation\_importance.ExplanationAdapter (class in interpret\_community.adapter), 16  
 attribute), 78

**explainer\_type** (interpret\_community.permutation.PFIExplainer (class in interpret\_community.adapter.explanation\_adapter), 17  
 attribute), 75

**explainer\_type** (interpret\_community.shap.deep\_explainer.DeepExplainer (class in interpret\_community.widget), 102  
 attribute), 91

**explainer\_type** (interpret\_community.shap.DeepExplainer (class in interpret\_community.widget.explanation\_dashboard), 103  
 attribute), 81

**explainer\_type** (interpret\_community.shap.gpu\_kernel\_explainer.GPUKernelExplainer (class in interpret\_community.common.constants), 23  
 attribute), 93

**explainer\_type** (interpret\_community.shap.GPUKernelExplainer (class in interpret\_community.common.constants), 24  
 attribute), 83

**explainer\_type** (interpret\_community.shap.kernel\_explainer.KernelExplainer (class in interpret\_community.common.constants), 24  
 attribute), 96

**explainer\_type** (interpret\_community.shap.KernelExplainer (class in interpret\_community.common.constants), 24  
 attribute), 85

**explainer\_type** (interpret\_community.shap.linear\_explainer.LinearExplainer (class in interpret\_community.explanation.explanation), 34  
 attribute), 98

**explainer\_type** (interpret\_community.shap.LinearExplainer (class in interpret\_community.explanation.explanation), 34  
 attribute), 87

**explainer\_type** (interpret\_community.shap.tree\_explainer.TreeExplainer (class in interpret\_community.explanation.explanation), 34  
 attribute), 100

**explainer\_type** (interpret\_community.shap.TreeExplainer (class in interpret\_community.explanation.explanation), 34  
 attribute), 89

**explainer\_type** (interpret\_community.tabular\_explainer.TabularExplainer (class in interpret\_community.explanation.explanation), 34  
 attribute), 105

**explainer\_type** (interpret\_community.TabularExplainer (class in interpret\_community.explanation.explanation), 34  
 attribute), 15

**ExplainParams** (class in interpret\_community.common.constants), 21

**ExplainType** (class in interpret\_community.common.constants), 23

**EXPLANATION\_CLASS\_DIMENSION** (interpret\_community.common.constants.ExplainParams (class in interpret\_community.common.constants), 21  
 attribute), 24

**EXPLANATION\_ID** (interpret\_community.common.constants.ExplainParams (class in interpret\_community.common.constants), 21  
 attribute), 21

**EXPLANATION\_TYPE** (interpret\_community.common.constants.ExplainParams (class in interpret\_community.common.constants), 21  
 attribute), 24

**EXPLAIN\_ADAPTER** (class in interpret\_community.adapter), 16

**EXPLANATION\_ADAPTER** (class in interpret\_community.adapter.explanation\_adapter), 17

**EXPLANATION\_DASHBOARD** (class in interpret\_community.widget), 102

**EXPLANATION\_DASHBOARD** (class in interpret\_community.widget.explanation\_dashboard), 103

**EXPLANATION\_PARAMS** (class in interpret\_community.common.constants), 23

**EXPLAIN\_PARAMS** (class in interpret\_community.common.constants), 23

**EXPLAIN\_TYPE** (class in interpret\_community.common.constants), 23

**EXTRA** (interpret\_community.common.constants.ExplainParams (class in interpret\_community.common.constants), 21  
 attribute), 24

**F**

**FBETA\_SCORE** (interpret\_community.permutation.metric\_constants.MetricConstants (class in interpret\_community.permutation.metric\_constants), 77  
 attribute), 77

**FBETA\_SCORE** (interpret\_community.permutation.metric\_constants.MetricConstants (class in interpret\_community.permutation.metric\_constants), 77  
 attribute), 77

**FEATURE\_LIST** (interpret\_community.common.constants.ExplainParams (class in interpret\_community.common.constants), 21  
 attribute), 24

**FEATURE\_IMPORTANCE\_EXPLANATION** (class in interpret\_community.explanation.explanation), 34

**FEATURE\_IMPORTANCE\_EXPLANATION** (class in interpret\_community.explanation.explanation), 34

**FEATURES** (interpret\_community.common.constants.ExplainParams (class in interpret\_community.common.constants), 21  
 attribute), 21

**features** (interpret\_community.explanation.explanation.FeatureImportance (class in interpret\_community.explanation.explanation), 34  
 property), 34

**fit()** (interpret\_community.mimic.models.BaseExplainableModel (class in interpret\_community.mimic.models), 48  
 method), 48

**fit()** (interpret\_community.mimic.models.DecisionTreeExplainableModel (class in interpret\_community.mimic.models), 49  
 method), 49

**fit()** (interpret\_community.mimic.models.explainable\_model.BaseExplainableModel (class in interpret\_community.mimic.models.explainable\_model), 59  
 method), 59

**fit()** (interpret\_community.mimic.models.LGBMExplainableModel (class in interpret\_community.mimic.models), 52  
 method), 52

**fit()** (interpret\_community.mimic.models.lightgbm\_model.LGBMExplainableModel (class in interpret\_community.mimic.models), 60  
 method), 60

**fit()** (interpret\_community.mimic.models.linear\_model.LinearExplainableModel (class in interpret\_community.mimic.models), 62  
 method), 62

**fit()** (interpret\_community.mimic.models.linear\_model.SGDExplainableModel (class in interpret\_community.mimic.models), 65  
 method), 65

**fit()** (interpret\_community.mimic.models.LinearExplainableModel (class in interpret\_community.mimic.models), 54  
 method), 54

**fit()** (interpret\_community.mimic.models.SGDExplainableModel (class in interpret\_community.mimic.models), 56  
 method), 56

**fit()** (interpret\_community.mimic.models.tree\_model.DecisionTreeExplainableModel (class in interpret\_community.mimic.models), 68  
 method), 68

**flush()** (interpret\_community.shap.deep\_explainer.logger\_redirector (class in interpret\_community.shap.deep\_explainer), 92  
 method), 92



FUNCTION (*interpret\_community.common.constants.ExplainType* method), 37  
 attribute), 23  
 FUNCTION (*interpret\_community.common.constants.MimicSerializationConstants* method), 36  
 attribute), 25  
**G**  
 get\_artifacts() (interpret\_community.common.model\_summary.ModelSummary method), 29  
 get\_artifacts() (interpret\_community.common.ModelSummary method), 18  
 get\_explanation() (in module interpret\_community.mlflow), 73  
 get\_explanation() (in module interpret\_community.mlflow.mlflow), 74  
 get\_feature\_importance\_dict() (interpret\_community.explanation.explanation.GlobalExplanation method), 35  
 get\_local\_importance\_rank() (interpret\_community.explanation.explanation.LocalExplanation method), 37  
 get\_metadata\_dictionary() (interpret\_community.common.model\_summary.ModelSummary method), 29  
 get\_metadata\_dictionary() (interpret\_community.common.ModelSummary method), 18  
 get\_private() (interpret\_community.common.constants.ExplainParams class method), 22  
 get\_ranked\_global\_names() (interpret\_community.explanation.explanation.GlobalExplanation method), 35  
 get\_ranked\_global\_values() (interpret\_community.explanation.explanation.GlobalExplanation method), 35  
 get\_ranked\_local\_names() (interpret\_community.explanation.explanation.LocalExplanation method), 37  
 get\_ranked\_local\_values() (interpret\_community.explanation.explanation.LocalExplanation method), 37  
 get\_ranked\_per\_class\_names() (interpret\_community.explanation.explanation.PerClassMixin method), 39  
 get\_ranked\_per\_class\_values() (interpret\_community.explanation.explanation.PerClassMixin method), 39  
 get\_raw\_explanation() (interpret\_community.explanation.explanation.GlobalExplanation method), 36  
 get\_raw\_explanation() (interpret\_community.explanation.explanation.LocalExplanation method), 36  
 get\_raw\_feature\_importances() (interpret\_community.explanation.explanation.GlobalExplanation method), 36  
 get\_raw\_feature\_importances() (interpret\_community.explanation.explanation.LocalExplanation method), 38  
 get\_serializable() (interpret\_community.common.constants.ExplainParams class method), 22  
 get\_surrogate\_model\_replication\_measure() (interpret\_community.mimic.mimic\_explainer.MimicExplainer method), 73  
 get\_surrogate\_model\_replication\_measure() (interpret\_community.mimic.MimicExplainer method), 47  
 get\_tqdm() (in module interpret\_community.common.progress), 30  
 GLASSBOX (interpret\_community.common.constants.Extension attribute), 24  
 GLOBAL (interpret\_community.common.constants.ExplainType attribute), 23  
 GLOBAL (interpret\_community.common.constants.Extension attribute), 24  
 GLOBAL\_EXPLANATION (interpret\_community.common.constants.Dynamic attribute), 21  
 GLOBAL\_FEATURE\_IMPORTANCE (interpret\_community.common.constants.InterpretData attribute), 24  
 GLOBAL\_IMPORTANCE\_NAMES (interpret\_community.common.constants.ExplainParams attribute), 21  
 GLOBAL\_IMPORTANCE\_RANK (interpret\_community.common.constants.ExplainParams attribute), 21  
 global\_importance\_rank (interpret\_community.explanation.explanation.GlobalExplanation property), 36  
 GLOBAL\_IMPORTANCE\_VALUES (interpret\_community.common.constants.ExplainParams attribute), 22  
 global\_importance\_values (interpret\_community.explanation.explanation.GlobalExplanation property), 36  
 GLOBAL\_NAMES (interpret\_community.common.constants.ExplainParams attribute), 22  
 GLOBAL\_RANK (interpret\_community.common.constants.ExplainParams attribute), 22  
 GLOBAL\_VALUES (interpret\_community.common.constants.ExplainParams attribute), 22  
 GlobalExplainer (class in interpret\_community.common.base\_explainer),

19		interpret_community.common
GlobalExplanation	(class in interpret_community.explanation.explanation), 35	module, 18
GPUKernelExplainer	(class in interpret_community.shap), 82	interpret_community.common.aggregate module, 18
GPUKernelExplainer	(class in interpret_community.shap.gpu_kernel_explainer), 93	interpret_community.common.base_explainer module, 19
GREYBOX	(interpret_community.common.constants.Extension attribute), 24	interpret_community.common.blackbox_explainer module, 19
H		interpret_community.common.chained_identity module, 20
HAN	(interpret_community.common.constants.ExplainType attribute), 23	interpret_community.common.constants module, 21
HDBSCAN	(interpret_community.common.constants.Defaults attribute), 21	interpret_community.common.error_handling module, 27
I		interpret_community.common.exception module, 27
ID	(interpret_community.common.constants.ExplainParams attribute), 22	interpret_community.common.explanation_utils module, 27
id	(interpret_community.explanation.explanation.BaseExplanation property), 33	interpret_community.common.gpu_kmeans module, 27
IDENTITY	(interpret_community.common.constants.LightGBMSerializationConstants attribute), 25	interpret_community.common.metrics module, 28
IDENTITY	(interpret_community.common.constants.MimicSerializationConstants attribute), 25	interpret_community.common.model_summary module, 28
Ignore	(interpret_community.common.constants.ResetIndex attribute), 26	interpret_community.common.model_wrapper module, 18
INCLUDE_LOCAL	(interpret_community.common.constants.ExplainParams attribute), 22	interpret_community.common.policy module, 29
INDEPENDENT	(interpret_community.common.constants.SHAP Defaults attribute), 26	interpret_community.common.progress module, 30
init_aggregator_decorator()	(in module interpret_community.common.aggregate), 19	interpret_community.common.serialization_utils module, 30
init_blackbox_decorator()	(in module interpret_community.common.blackbox_explainer), 20	interpret_community.common.structured_model_explainer module, 30
INIT_DATA	(interpret_community.common.constants.ExplainParams attribute), 22	interpret_community.common.warnings_suppressor module, 31
init_tabular_decorator()	(in module interpret_community.dataset.decorator), 31	interpret_community.dataset module, 31
INITIALIZATION_EXAMPLES	(interpret_community.common.constants.MimicSerializationConstants attribute), 25	interpret_community.dataset.dataset_wrapper module, 31
INTERCEPT	(interpret_community.common.constants.InterpretData attribute), 24	interpret_community.dataset.decorator module, 31
interpret_community		interpret_community.explanation module, 32
interpret_community.adapter		interpret_community.explanation.explanation module, 32
interpret_community.adapter.explanation_adapter		interpret_community.explanation.serialization module, 40
		interpret_community.lime module, 40
		interpret_community.lime.lime_explainer module, 43
		interpret_community.mimic module, 45

interpret\_community.mimic.mimic\_explainer module, 70  
 interpret\_community.mimic.model\_distill module, 73  
 interpret\_community.mimic.models module, 48  
 interpret\_community.mimic.models.explainable\_model module, 59  
 interpret\_community.mimic.models.lightgbm\_model module, 59  
 interpret\_community.mimic.models.linear\_model module, 61  
 interpret\_community.mimic.models.tree\_model module, 67  
 interpret\_community.mimic.models.tree\_model\_utils module, 70  
 interpret\_community.mlflow module, 73  
 interpret\_community.mlflow.mlflow module, 74  
 interpret\_community.permutation module, 75  
 interpret\_community.permutation.metric\_constants module, 77  
 interpret\_community.permutation.permutation\_importance module, 77  
 interpret\_community.shap module, 80  
 interpret\_community.shap.deep\_explainer module, 90  
 interpret\_community.shap.gpu\_kernel\_explainer module, 93  
 interpret\_community.shap.kernel\_explainer module, 95  
 interpret\_community.shap.kwarg\_utils module, 98  
 interpret\_community.shap.linear\_explainer module, 98  
 interpret\_community.shap.tree\_explainer module, 100  
 interpret\_community.tabular\_explainer module, 104  
 interpret\_community.version module, 106  
 interpret\_community.widget module, 102  
 interpret\_community.widget.explanation\_dashboard module, 103  
 InterpretData (class in interpret\_community.common.constants), 24  
 IS\_ENG (interpret\_community.common.constants.ExplainParams attribute), 22  
 IS\_ENG (interpret\_community.common.constants.ExplainType attribute), 23  
 is\_engineered (interpret\_community.explanation.explanation.FeatureImportanceExplainer property), 34  
 IS\_LOCAL\_SPARSE (interpret\_community.common.constants.ExplainParams attribute), 22  
 is\_local\_sparse (interpret\_community.explanation.explanation.LocalExplanation property), 38  
 IS\_RAW (interpret\_community.common.constants.ExplainParams attribute), 22  
 IS\_RAW (interpret\_community.common.constants.ExplainType attribute), 23  
 is\_raw (interpret\_community.explanation.explanation.FeatureImportanceExplainer property), 35  
**K**  
 KernelExplainer (class in interpret\_community.shap), 84  
 KernelExplainer (class in interpret\_community.shap.kernel\_explainer), 95  
 kmeans() (in module interpret\_community.common.gpu\_kmeans), 27  
**L**  
 LABELS (interpret\_community.common.constants.SKLearn attribute), 26  
 labels\_decorator() (in module interpret\_community.permutation.permutation\_importance), 80  
 LGBMExplainableModel (class in interpret\_community.mimic.models), 51  
 LGBMExplainableModel (class in interpret\_community.mimic.models.lightgbm\_model), 59  
 LightGBMParams (class in interpret\_community.common.constants), 24  
 LightGBMSerializationConstants (class in interpret\_community.common.constants), 25  
 LIME (interpret\_community.common.constants.ExplainType attribute), 23  
 LIMEExplainer (class in interpret\_community.lime), 40  
 LIMEExplainer (class in interpret\_community.lime.lime\_explainer), 43  
 LINEAR\_EXPLAINABLE\_MODEL\_TYPE (interpret\_community.common.constants.ExplainableModelType attribute), 23  
 LinearExplainableModel (class in interpret\_community.mimic.models), 53  
 LinearExplainableModel (class in interpret\_community.mimic.models.linear\_model), 61

LinearExplainer (class in interpret\_community.mimic.models.linear\_model), 64  
 LinearExplainer (class in interpret\_community.shap), 86  
 LinearExplainer (class in interpret\_community.shap.linear\_explainer), 98  
 load\_explanation() (in module interpret\_community.explanation), 32  
 load\_explanation() (in module interpret\_community.explanation.serialization), 40  
 LOCAL (interpret\_community.common.constants.ExplainType attribute), 23  
 LOCAL (interpret\_community.common.constants.Extension attribute), 24  
 LOCAL\_EXPLANATION (interpret\_community.common.constants.Dynamic attribute), 21  
 LOCAL\_EXPLANATION (interpret\_community.common.constants.ExplainParams attribute), 22  
 LOCAL\_FEATURE\_IMPORTANCE (interpret\_community.common.constants.InterpretData attribute), 24  
 LOCAL\_IMPORTANCE\_VALUES (interpret\_community.common.constants.ExplainParams attribute), 22  
 local\_importance\_values (interpret\_community.explanation.explanation.LocalExplanation property), 38  
 LocalExplainer (class in interpret\_community.common.base\_explainer), 19  
 LocalExplanation (class in interpret\_community.explanation.explanation), 37  
 log\_explanation() (in module interpret\_community.mlflow), 73  
 log\_explanation() (in module interpret\_community.mlflow.mlflow), 74  
 LOGGER (interpret\_community.common.constants.LightGBMModelConstants attribute), 25  
 LOGGER (interpret\_community.common.constants.MimicSerializationConstants attribute), 25  
 logger\_redirector (class in interpret\_community.shap.deep\_explainer), 92  
**M**  
 MAX\_DIM (interpret\_community.common.constants.Defaults attribute), 21  
 max\_dim\_clustering (interpret\_community.common.policy.SamplingPolicy property), 30  
 MEAN\_ABSOLUTE\_ERROR (interpret\_community.permutation.metric\_constants.MetricConstants attribute), 77  
 MEAN\_SQUARED\_ERROR (interpret\_community.permutation.metric\_constants.MetricConstants attribute), 77  
 MEAN\_SQUARED\_LOG\_ERROR (interpret\_community.permutation.metric\_constants.MetricConstants attribute), 77  
 MEDIAN\_ABSOLUTE\_ERROR (interpret\_community.permutation.metric\_constants.MetricConstants attribute), 77  
 METHOD (interpret\_community.common.constants.ExplainParams attribute), 22  
 METHOD (interpret\_community.common.constants.ExplainType attribute), 23  
 method (interpret\_community.explanation.explanation.BaseExplanation property), 33  
 MetricConstants (class in interpret\_community.permutation.metric\_constants), 77  
 MIMIC (interpret\_community.common.constants.ExplainType attribute), 23  
 MimicExplainer (class in interpret\_community.mimic), 45  
 MimicExplainer (class in interpret\_community.mimic.mimic\_explainer), 70  
 MimicSerializationConstants (class in interpret\_community.common.constants), 25  
 MIN\_DATA\_IN\_LEAF (interpret\_community.common.constants.LightGBMParams attribute), 25  
 MLI (interpret\_community.common.constants.InterpretData attribute), 24  
 MODEL (interpret\_community.common.constants.ExplainType attribute), 23  
 MODEL (interpret\_community.common.constants.MimicSerializationConstants attribute), 25  
 model (interpret\_community.mimic.models.BaseExplainableModel property), 48  
 model (interpret\_community.mimic.models.DecisionTreeExplainableModel property), 50  
 model (interpret\_community.mimic.models.explainable\_model.BaseExplainableModel property), 59  
 model (interpret\_community.mimic.models.LGBMExplainableModel property), 53  
 model (interpret\_community.mimic.models.lightgbm\_model.LGBMExplainableModel property), 60  
 model (interpret\_community.mimic.models.linear\_model.LinearExplainableModel property), 63  
 model (interpret\_community.mimic.models.linear\_model.SGDExplainableModel property), 65

`model` (`interpret_community.mimic.models.LinearExplainableModel` property), 55  
`model` (`interpret_community.mimic.models.SGDExplainableModel` property), 57  
`model` (`interpret_community.mimic.models.tree_model.DecisionTreeExplainableModel` property), 69  
`MODEL_CLASS` (`interpret_community.common.constants.ExplainType` attribute), 23  
`MODEL_ID` (`interpret_community.common.constants.ExplainParameter` attribute), 22  
`MODEL_STR` (`interpret_community.common.constants.LightGBMSerializationConstants` attribute), 25  
`MODEL_TASK` (`interpret_community.common.constants.ExplainParameter` attribute), 22  
`MODEL_TASK` (`interpret_community.common.constants.ExplainType` attribute), 23  
`model_task` (`interpret_community.explanation.explanation.BaseExplanation` property), 33  
`MODEL_TYPE` (`interpret_community.common.constants.ExplainParameter` attribute), 22  
`model_type` (`interpret_community.explanation.explanation.BaseExplanation` property), 33  
`ModelSummary` (class in `interpret_community.common`), 18  
`ModelSummary` (class in `interpret_community.common.model_summary`), 28  
`ModelTask` (class in `interpret_community.common.constants`), 25  
`module`  
`interpret_community`, 14  
`interpret_community.adapter`, 16  
`interpret_community.adapter.explanation_adapter`, 17  
`interpret_community.common`, 18  
`interpret_community.common.aggregate`, 18  
`interpret_community.common.base_explainer`, 19  
`interpret_community.common.blackbox_explainer`, 19  
`interpret_community.common.chained_identity`, 20  
`interpret_community.common.constants`, 21  
`interpret_community.common.error_handling`, 27  
`interpret_community.common.exception`, 27  
`interpret_community.common.explanation_utils`, 27  
`interpret_community.common.gpu_kmeans`, 27  
`interpret_community.common.metrics`, 28  
`interpret_community.common.model_summary`, 28  
`interpret_community.common.model_wrapper`, 18  
`interpret_community.common.policy`, 29  
`interpret_community.common.progress`, 30  
`interpret_community.common.serialization_utils`, 30  
`interpret_community.common.structured_model_explainer`, 30  
`interpret_community.common.warnings_suppressor`, 31  
`interpret_community.dataset`, 31  
`interpret_community.dataset.dataset_wrapper`, 31  
`interpret_community.dataset.decorator`, 31  
`interpret_community.explanation`, 32  
`interpret_community.explanation.explanation`, 32  
`interpret_community.explanation.serialization`, 40  
`interpret_community.lime`, 40  
`interpret_community.lime.lime_explainer`, 43  
`interpret_community.mimic`, 45  
`interpret_community.mimic.mimic_explainer`, 70  
`interpret_community.mimic.model_distill`, 73  
`interpret_community.mimic.models`, 48  
`interpret_community.mimic.models.explainable_model`, 59  
`interpret_community.mimic.models.lightgbm_model`, 59  
`interpret_community.mimic.models.linear_model`, 61  
`interpret_community.mimic.models.tree_model`, 67  
`interpret_community.mimic.models.tree_model_utils`, 70  
`interpret_community.mlflow`, 73  
`interpret_community.mlflow.mlflow`, 74  
`interpret_community.permutation`, 75  
`interpret_community.permutation.metric_constants`, 77  
`interpret_community.permutation.permutation_importance`, 77  
`interpret_community.shap`, 80  
`interpret_community.shap.deep_explainer`, 90  
`interpret_community.shap.gpu_kernel_explainer`, 93  
`interpret_community.shap.kernel_explainer`, 95  
`interpret_community.shap.kwarg_utils`, 98  
`interpret_community.shap.linear_explainer`, 98  
`interpret_community.shap.tree_explainer`,



100  
 interpret\_community.tabular\_explainer, 104  
 interpret\_community.version, 106  
 interpret\_community.widget, 102  
 interpret\_community.widget.explanation\_dashboard, 103  
 MULTICLASS (interpret\_community.common.constants.InterpretData attribute), 24  
 MULTICLASS (interpret\_community.common.constants.LightGBMSerializationConstants attribute), 25  
**N**  
 name (interpret\_community.explanation.explanation.BaseExplanation property), 33  
 NAMES (interpret\_community.common.constants.InterpretData attribute), 24  
 ndcg() (in module interpret\_community.common.metrics), 28  
 NER (interpret\_community.common.constants.Spacy attribute), 27  
 nonify\_properties (interpret\_community.common.constants.LightGBMSerializationConstants attribute), 25  
 nonify\_properties (interpret\_community.common.constants.MimicSerializationConstants attribute), 25  
 NUM\_CLASSES (interpret\_community.common.constants.ExplainParams attribute), 22  
 num\_classes (interpret\_community.explanation.explanation.ClassesMixin property), 34  
 NUM\_EXAMPLES (interpret\_community.common.constants.ExplainParams attribute), 22  
 num\_examples (interpret\_community.explanation.explanation.ExplanationLocalExplanation property), 38  
 NUM\_FEATURES (interpret\_community.common.constants.ExplainParams attribute), 22  
 num\_features (interpret\_community.explanation.explanation.FeatureImportanceExplanation property), 35  
**O**  
 OBJECTIVE (interpret\_community.common.constants.LightGBMSerializationConstants attribute), 25  
 ORIGINAL\_EVAL\_EXAMPLES (interpret\_community.common.constants.MimicSerializationConstants attribute), 25  
 OVERALL (interpret\_community.common.constants.InterpretData attribute), 24  
**P**  
 PER\_CLASS\_NAMES (interpret\_community.common.constants.ExplainParams attribute), 22  
 PER\_CLASS\_RANK (interpret\_community.common.constants.ExplainParams attribute), 22  
 per\_class\_rank (interpret\_community.explanation.explanation.PerClassMixin property), 39  
 PER\_CLASS\_VALUES (interpret\_community.common.constants.ExplainParams attribute), 22  
 PER\_CLASS\_VALUES (interpret\_community.explanation.explanation.PerClassMixin property), 39  
 PerClassMixin (class in interpret\_community.explanation.explanation), 39  
 PERF (interpret\_community.common.constants.InterpretData attribute), 24  
 PFI (interpret\_community.common.constants.ExplainType attribute), 23  
 PFIExplainer (class in interpret\_community.permutation), 75  
 PFIExplainer (class in interpret\_community.permutation.permutation\_importance), 77  
 PRECISION\_SCORE (interpret\_community.permutation.metric\_constants.MetricConstants attribute), 77  
 predict() (interpret\_community.mimic.models.BaseExplainableModel method), 48  
 predict() (interpret\_community.mimic.models.DecisionTreeExplainableModel method), 50  
 predict() (interpret\_community.mimic.models.explainable\_model.BaseExplainableModel method), 59  
 predict() (interpret\_community.mimic.models.LGBMExplainableModel method), 53  
 predict() (interpret\_community.mimic.models.lightgbm\_model.LGBMExplainableModel method), 60  
 predict() (interpret\_community.mimic.models.linear\_model.LinearExplainableModel method), 63  
 predict() (interpret\_community.mimic.models.linear\_model.SGDExplainableModel method), 65  
 predict() (interpret\_community.mimic.models.LinearExplainableModel method), 55  
 predict() (interpret\_community.mimic.models.SGDExplainableModel method), 57  
 predict() (interpret\_community.mimic.models.tree\_model.DecisionTreeExplainableModel method), 69  
 PREDICT\_PROBA (interpret\_community.common.constants.SKLearn attribute), 26  
 predict\_proba() (interpret\_community.mimic.models.BaseExplainableModel method), 48  
 predict\_proba() (interpret\_community.mimic.models.LinearExplainableModel method), 63  
 predict\_proba() (interpret\_community.mimic.models.SGDExplainableModel method), 57  
 predict\_proba() (interpret\_community.mimic.models.tree\_model.DecisionTreeExplainableModel method), 69

`pret_community.mimic.models.DecisionTreeExplainerModel` (class in `pret_community.common.constants.ResetIndex` attribute), 26  
`pret_community.mimic.models.explainable_model.BaseExplainerModel` (class in `pret_community.common.constants.MimicSerializationConstants` attribute), 26  
`pret_community.mimic.models.LGBMExplainerModel` (class in `pret_community.common.constants.ResetIndex` attribute), 26  
`pret_community.mimic.models.lightgbm_model.LGBMExplainerModel` (class in `pret_community.common.constants.ResetIndex` attribute), 26  
`pret_community.mimic.models.linear_model.LinearExplainerModel` (class in `pret_community.common.constants.SAMPLING_POLICY` attribute), 30  
`pret_community.mimic.models.linear_model.SGDExplainerModel` (class in `pret_community.common.constants.SAMPLING_POLICY` attribute), 22  
`pret_community.mimic.models.LinearExplainerModel` (class in `pret_community.common.constants.SAMPLING_POLICY` attribute), 29  
`pret_community.mimic.models.SGDExplainerModel` (class in `pret_community.common.constants.SAMPLING_POLICY` attribute), 40  
`pret_community.mimic.models.tree_model.DecisionTreeExplainerModel` (class in `pret_community.common.constants.SAMPLING_POLICY` attribute), 74  
`pret_community.common.constants.MimicSerializationConstants` (class in `pret_community.common.constants` attribute), 25  
`pret_community.common.constants.LightGBMSerializationConstants` (class in `pret_community.common.constants` attribute), 25  
`pret_community.common.constants.SKLearnConstants` (class in `pret_community.common.constants` attribute), 26  
`pret_community.common.constants.ExplainParams` (class in `pret_community.common.constants` attribute), 22  
`pret_community.common.constants.ScenarioNotSupportedException` (class in `pret_community.common.constants` attribute), 27  
`pret_community.common.constants.ShapValuesOutput` (class in `pret_community.common.constants` attribute), 26  
`pret_community.common.constants.Scores` (class in `pret_community.common.constants` attribute), 24  
`pret_community.common.constants.selector` (class in `pret_community.common.constants` attribute), 33  
`pret_community.common.constants.selector` (class in `pret_community.common.constants` attribute), 36  
`pret_community.common.constants.selector` (class in `pret_community.common.constants` attribute), 38  
`pret_community.common.constants.SGDExplainerModel` (class in `pret_community.common.constants` attribute), 56  
`pret_community.common.constants.SGDExplainerModel` (class in `pret_community.common.constants` attribute), 64  
`pret_community.common.constants.SHAPSerializationConstants` (class in `pret_community.common.constants` attribute), 23  
`pret_community.common.constants.SHAP_DEEP` (class in `pret_community.common.constants` attribute), 23

**P**  
`PREDICT_PROBA_FLAG` (class in `pret_community.common.constants` attribute), 25  
`PREDICTIONS` (class in `pret_community.common.constants` attribute), 26  
`PROBABILITIES` (class in `pret_community.common.constants` attribute), 22  
`PROBABILITY` (class in `pret_community.common.constants` attribute), 26  
`PureStructuredModelExplainer` (class in `pret_community.common.structured_model_explainer` attribute), 30  
`PYTORCH` (class in `pret_community.common.constants` attribute), 21

**R**  
`R2_SCORE` (class in `pret_community.permutation.metric_constants` attribute), 77  
`RECALL_SCORE` (class in `pret_community.permutation.metric_constants` attribute), 77  
`REGRESSION` (class in `pret_community.common.constants` attribute), 23  
`REGRESSION` (class in `pret_community.common.constants` attribute), 25  
`Regression` (class in `pret_community.common.constants` attribute), 26

SHAP\_GPU\_KERNEL (interpret\_community.common.constants.ExplainType attribute), 23  
 SHAP\_KERNEL (interpret\_community.common.constants.ExplainType attribute), 23  
 SHAP\_LINEAR (interpret\_community.common.constants.ExplainType attribute), 23  
 SHAP\_TREE (interpret\_community.common.constants.ExplainType attribute), 23  
 shap\_values() (interpret\_community.mimic.models.linear\_model.LinearExplainer attribute), 64  
 SHAP\_VALUES\_OUTPUT (interpret\_community.common.constants.ExplainParams attribute), 22  
 shap\_warnings\_suppressor (class in interpret\_community.common.warnings\_suppressor), 31  
 SHAPDefaults (class in interpret\_community.common.constants), 26  
 ShapValuesOutput (class in interpret\_community.common.constants), 26  
 SINGLE (interpret\_community.common.constants.InterpretData attribute), 24  
 SKLearn (class in interpret\_community.common.constants), 26  
 Spacy (class in interpret\_community.common.constants), 27  
 SPECIFIC (interpret\_community.common.constants.InterpretData attribute), 24  
 StructuredInitModelExplainer (class in interpret\_community.common.structured\_model\_explainer), 30  
**T**  
 TABULAR (interpret\_community.common.constants.ExplainType attribute), 23  
 tabular\_decorator() (in module interpret\_community.dataset.decorator), 32  
 TabularExplainer (class in interpret\_community), 14  
 TabularExplainer (class in interpret\_community.tabular\_explainer), 104  
 TAGGER (interpret\_community.common.constants.Spacy attribute), 27  
 TEACHER\_PROBABILITY (interpret\_community.common.constants.ShapValuesOutput attribute), 26  
 Tensorflow (class in interpret\_community.common.constants), 27  
 TENSORFLOW (interpret\_community.common.constants.DNNFramework attribute), 21  
 tf\_warnings\_suppressor (class in interpret\_community.common.warnings\_suppressor), 31  
 TFLOG (interpret\_community.common.constants.Tensorflow attribute), 27  
 TIMESTAMP\_FEATURIZER (interpret\_community.common.constants.MimicSerializationConstants attribute), 25  
 TREE\_EXPLAINABLE\_MODEL\_TYPE (interpret\_community.common.constants.ExplainableModelType attribute), 23  
 TREE\_EXPLAINER (interpret\_community.common.constants.LightGBMSerializationConstants attribute), 25  
 TreeExplainer (class in interpret\_community.shap), 88  
 TreeExplainer (class in interpret\_community.shap.tree\_explainer), 100  
 TYPE (interpret\_community.common.constants.InterpretData attribute), 24  
**U**  
 UNIVARIATE (interpret\_community.common.constants.InterpretData attribute), 24  
 Unknown (interpret\_community.common.constants.ModelTask attribute), 26  
**V**  
 VALUE (interpret\_community.common.constants.InterpretData attribute), 24  
 VALUES (interpret\_community.common.constants.InterpretData attribute), 24  
 visualize() (interpret\_community.explanation.explanation.BaseExplainer method), 33  
**W**  
 wrap\_dataset() (in module interpret\_community.dataset.decorator), 32  
 write() (interpret\_community.shap.deep\_explainer.logger\_redirector method), 92