
interpret-community

Release 0.23.0

Microsoft

Jan 03, 2022

CONTENTS

1 API Reference	3
2 Indices and tables	99
Python Module Index	101
Index	103

The code is available from [GitHub](#).

API REFERENCE

1.1 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.array` 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.array` 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.array` 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

1.1.1 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.array* or *scipy.sparse.csr_matrix* or *list[scipy.sparse.csr_matrix]*) – The feature importance values.
- **evaluation_examples** (*numpy.array* 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.array*) – 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.array* or *scipy.sparse.csr_matrix* or *list[scipy.sparse.csr_matrix]*) – The feature importance values.

- **evaluation_examples** (`numpy.array` 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.array`) – 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(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.array` or `scipy.sparse.csr_matrix` or `list[scipy.sparse.csr_matrix]`) – The feature importance values.
- **evaluation_examples** (`numpy.array` 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.array`) – 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.array` or `scipy.sparse.csr_matrix` or `list[scipy.sparse.csr_matrix]`) – The feature importance values.
- **evaluation_examples** (`numpy.array` 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.array`) – 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`

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'

class interpret_community.common.constants.LightGBMSerializationConstants
Bases: object

Provide internal class that defines fields used for MimicExplainer serialization.

IDENTITY = '_identity'
LOGGER = '_logger'
MODEL_STR = 'model_str'
MULTICLASS = 'multiclass'
OBJECTIVE = 'objective'
REGRESSION = 'regression'
TREE_EXPLAINER = '_tree_explainer'
```

```
enum_properties = ['_shap_values_output']
nonify_properties = ['_logger', '_tree_explainer']
save_properties = ['_lgbm']

class interpret_community.common.constants.MimicSerializationConstants
Bases: object

Provide internal class that defines fields used for MimicExplainer serialization.

ALLOW_ALL_TRANSFORMATIONS = '_allow_all_transformations'
FUNCTION = 'function'
IDENTITY = '_identity'
INITIALIZATION_EXAMPLES = 'initialization_examples'
LOGGER = '_logger'
MODEL = 'model'
ORIGINAL_EVAL_EXAMPLES = '_original_eval_examples'
PREDICT_PROBA_FLAG = 'predict_proba_flag'
RESET_INDEX = 'reset_index'
TIMESTAMP_FEATURIZER = '_timestamp_featurizer'
enum_properties = ['_shap_values_output']
nonify_properties = ['_logger', 'model', 'function', 'initialization_examples',
'_original_eval_examples', '_timestamp_featurizer']
save_properties = ['surrogate_model']

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

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

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

Classification = 'classification'
Regression = 'regression'
Unknown = 'unknown'

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

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

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

Ignore = 'ignore'
Reset = 'reset'
ResetTeacher = 'reset_teacher'
```

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

Provide constants for default values to SHAP.

INDEPENDENT = 'independent'

class interpret_community.common.constants.SKLearn
Bases: object

Provide scikit-learn related constants.

EXAMPLES = 'examples'

LABELS = 'labels'

PREDICTIONS = 'predictions'

PREDICT_PROBA = 'predict_proba'

class interpret_community.common.constants.Scipy
Bases: object

Provide scipy related constants.

CSR_FORMAT = 'csr'

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

Provide constants for the SHAP values output from the explainer.

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

DEFAULT = 'default'

PROBABILITY = 'probability'

TEACHER_PROBABILITY = 'teacher_probability'

class interpret_community.common.constants.Spacy
Bases: object

Provide spaCy related constants.

EN = 'en'

NER = 'ner'

TAGGER = 'tagger'

class interpret_community.common.constants.Tensorflow
Bases: object

Provide TensorFlow and TensorBoard related constants.

CPU0 = '/CPU:0'

TFLOG = 'tflog'
```

interpret_community.common.error_handling module

Defines error handling utilities.

interpret_community.common.exception module

Defines different types of exceptions that this package can raise.

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

An exception indicating that some scenario is not supported.

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

interpret_community.common.explanation_utils module

Defines helpful utilities for summarizing and uploading data.

interpret_community.common.gpu_kmeans module

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

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

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

`interpret_community.common.gpu_kmeans.kmeans(X, k, round_values=True)`

Summarize a dataset with k mean samples weighted by the number of data points they each represent. Parameters
———
X : numpy.array 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.

DenseData object.

interpret_community.common.metrics module

Defines metrics for validating model explanations.

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

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

Parameters

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

`interpret_community.common.metrics.ndcg(validate_order, ground_truth_order, top_values=10)`

Compute the normalized discounted cumulative gain (NDCG).

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

Parameters

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

interpret_community.common.model_summary module

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

`class interpret_community.common.model_summary.ModelSummary`

Bases: `object`

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

`add_from_get_model_summary(name, artifact_metadata_tuple)`

Update artifacts and metadata with new information.

Parameters

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

`get_artifacts()`

Get the list of artifacts.

Returns Artifact list.

Return type `list[list[dict]]`

`get_metadata_dictionary()`

Get the combined dictionary of metadata.

Returns Metadata dictionary.

Return type `dict`

interpret_community.common.model_wrapper module

Defines helpful model wrapper and utils for implicitly rewinding the model to conform to explainer contracts.

```
class interpret_community.common.model_wrapper.BaseWrappedModel(model, eval_function, examples, model_task)
```

Bases: `object`

A base class for WrappedClassificationModel and WrappedRegressionModel.

```
class interpret_community.common.model_wrapper.WrappedClassificationModel(model, eval_function, examples=None)
```

Bases: `interpret_community.common.model_wrapper.BaseWrappedModel`

A class for wrapping a classification model.

predict(dataset)

Predict the output using the wrapped classification model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict on.

predict_proba(dataset)

Predict the output probability using the wrapped model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict_proba on.

```
class interpret_community.common.model_wrapper.WrappedClassificationWithoutProbaModel(model)
```

Bases: `object`

A class for wrapping a classifier without a predict_proba method.

Note: the classifier may not output numeric values for its predictions. We generate a trivial boolean version of predict_proba

predict(dataset)

Predict the output using the wrapped regression model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict on.

predict_proba(dataset)

Predict the output probability using the wrapped model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict_proba on.

```
class interpret_community.common.model_wrapper.WrappedPytorchModel(model)
```

Bases: `object`

A class for wrapping a PyTorch model in the scikit-learn specification.

predict(dataset)

Predict the output using the wrapped PyTorch model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict on.

predict_classes(dataset)

Predict the class using the wrapped PyTorch model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict on.

predict_proba(`dataset`)
Predict the output probability using the wrapped PyTorch model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict_proba on.

```
class interpret_community.common.model_wrapper.WrappedRegressionModel(model, eval_function,
examples=None)
Bases: interpret_community.common.model_wrapper.BaseWrappedModel
```

A class for wrapping a regression model.

predict(`dataset`)
Predict the output using the wrapped regression model.

Parameters `dataset` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The dataset to predict on.

```
interpret_community.common.model_wrapper.wrap_model(model, examples, model_task)
If needed, wraps the model in a common API based on model task and prediction function contract.
```

Parameters

- `model` (`model with a predict or predict_proba function.`) – The model to evaluate on the examples.
- `examples` (`interpret_community.dataset.dataset_wrapper.DatasetWrapper`) – The model evaluation examples.
- `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.

Returns The wrapper model.

Return type `model`

interpret_community.common.policy module

Defines explanation policies.

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

Defines the sampling policy for downsampling the evaluation examples.

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

Parameters

- `allow_eval_sampling` (`bool`) – Default to ‘False’. Specify whether to allow sampling of evaluation data. If ‘True’, cluster the evaluation data and determine the optimal number of points for sampling. Set to ‘True’ to speed up the process when the evaluation data set is large and you only want to generate model summary info.

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

Return type `dict`

Returns The arguments for the sampling policy

property `allow_eval_sampling`

Get whether to allow sampling of evaluation data.

Returns Whether to allow sampling of evaluation data.

Return type `bool`

property `max_dim_clustering`

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

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

Return type `int`

property `sampling_method`

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

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

Return type `str`

interpret_community.common.progress module

Defines utilities for getting progress status for explanation.

`interpret_community.common.progress.get_tqdm(logger, show_progress)`

Get the tqdm progress bar function.

Parameters

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

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

Return type function

interpret_community.common.serialization_utils module

Defines utility functions for serialization of data.

interpret_community.common.structured_model_explainer module

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

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

Bases: *interpret_community.common.base_explainer.BaseExplainer*

The base PureStructuredModelExplainer API for explainers used on specific models.

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

```
class interpret_community.common.structured_model_explainer.StructuredInitModelExplainer(model,
                                         init-
                                         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.array* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.

interpret_community.dataset package

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

Submodules

interpret_community.dataset.dataset_wrapper module

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

```
class interpret_community.dataset.dataset_wrapper.CustomTimestampFeaturizer(features)
```

Bases: *sklearn.base.BaseEstimator*, *sklearn.base.TransformerMixin*

An estimator for featurizing timestamp columns to numeric data.

Parameters `features` (*list[str]*) – Feature column names.

`fit(X)`

Fits the CustomTimestampFeaturizer.

Parameters `X` (*numpy.array* or *pandas.DataFrame* or *scipy.sparse.csr_matrix*) –
The dataset containing timestamp columns to featurize.

transform(X)

Transforms the timestamp columns to numeric type in the given dataset.

Specifically, extracts the year, month, day, hour, minute, second and time since min timestamp in the training dataset.

Parameters `X` (`numpy.array` or `pandas.DataFrame` or `scipy.sparse.csr_matrix`) –

The dataset containing timestamp columns to featurize.

Returns The transformed dataset.

Return type `numpy.array` or `scipy.sparse.csr_matrix`

class `interpret_community.dataset.dataset_wrapper.DatasetWrapper(dataset, clear_references=False)`

Bases: `object`

A wrapper around a dataset to make dataset operations more uniform across explainers.

Parameters `dataset` (`numpy.array` or `pandas.DataFrame` or `scipy.sparse.csr_matrix`) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.

apply_indexer(column_indexer, bucket_unknown=False)

Indexes categorical string features on the dataset.

Parameters

- `column_indexer` (`sklearn.compose.ColumnTransformer`) – The transformation steps to index the given dataset.
- `bucket_unknown` (`bool`) – If true, buckets unknown values to separate categorical level.

apply_one_hot_encoder(one_hot_encoder)

One-hot-encode categorical string features on the dataset.

Parameters `one_hot_encoder` (`sklearn.preprocessing.OneHotEncoder`) – The transformation steps to one-hot-encode the given dataset.

apply_timestamp_featurizer(timestamp_featurizer)

Apply timestamp featurization on the dataset.

Parameters `timestamp_featurizer` (`CustomTimestampFeaturizer`) – The transformation steps to featurize timestamps in the given dataset.

augment_data(max_num_of_augmentations=inf)

Augment the current dataset.

Parameters `max_augment_data_size` (`int`) – number of times we stack permuted x to augment.

compute_summary(nclusters=10, use_gpu=False, **kwargs)

Summarizes the dataset if it hasn't been summarized yet.

property dataset

Get the dataset.

Returns The underlying dataset.

Return type `numpy.array` or `scipy.sparse.csr_matrix`

get_column_indexes(features, categorical_features)

Get the column indexes for the given column names.

Parameters

- **features** (`list[str]`) – The full list of existing column names.
- **categorical_features** (`list[str]`) – The list of categorical feature names to get indexes for.

Returns The list of column indexes.

Return type `list[int]`

get_features(`features=None, explain_subset=None, **kwargs`)

Get the features of the dataset if None on current kwargs.

Returns The features of the dataset if currently None on kwargs.

Return type `list`

property num_features

Get the number of features (columns) on the dataset.

Returns The number of features (columns) in the dataset.

Return type `int`

one_hot_encode(`columns`)

Indexes categorical string features on the dataset.

Parameters `columns` (`list[int]`) – Parameter specifying the subset of column indexes that may need to be one-hot-encoded.

Returns The transformation steps to one-hot-encode the given dataset.

Return type `sklearn.preprocessing.OneHotEncoder`

property original_dataset

Get the original dataset prior to performing any operations.

Note: if the original dataset was a pandas dataframe, this will return the numpy version.

Returns The original dataset.

Return type `numpy.array` or `scipy.sparse matrix`

property original_dataset_with_type

Get the original typed dataset which could be a numpy array or pandas DataFrame or pandas Series.

Returns The original dataset.

Return type `numpy.array` or `pandas.DataFrame` or `pandas.Series` or `scipy.sparse matrix`

reset_index()

Reset index to be part of the features on the dataset.

sample(`max_dim_clustering=50, sampling_method='hdbscan'`)

Sample the examples.

First does random downsampling to upper_bound rows, then tries to find the optimal downsample based on how many clusters can be constructed from the data. If sampling_method is hdbscan, uses hdbscan to cluster the data and then downsamples to that number of clusters. If sampling_method is k-means, uses different values of k, cutting in half each time, and chooses the k with highest silhouette score to determine how much to downsample the data. The danger of using only random downsampling is that we might downsample too much or too little, so the clustering approach is a heuristic to give us some idea of how much we should downsample to.

Parameters

- **max_dim_clustering** (`int`) – Dimensionality threshold for performing reduction.

- **sampling_method** (`str`) – Method to use for sampling, can be ‘hdbscan’ or ‘kmeans’.

set_index()

Undo reset_index. Set index as feature on internal dataset to be an index again.

string_index(columns=None)

Indexes categorical string features on the dataset.

Parameters `columns` (`list`) – Optional parameter specifying the subset of columns that may need to be string indexed.

Returns The transformation steps to index the given dataset.

Return type `sklearn.compose.ColumnTransformer`

property summary_dataset

Get the summary dataset without any subsetting.

Returns The original dataset or None if summary was not computed.

Return type `numpy.array` or `scipy.sparse.csr_matrix`

take_subset(explain_subset)

Take a subset of the dataset if not done before.

Parameters `explain_subset` (`list`) – A list of column indexes to take from the original dataset.

timestamp_featurizer()

Featurizes the timestamp columns.

Returns The transformation steps to featurize the timestamp columns.

Return type `interpret_community.dataset.dataset_wrapper.DatasetWrapper`

property typed_dataset

Get the dataset in the original type, pandas DataFrame or Series.

Returns The underlying dataset.

Return type `numpy.array` or `pandas.DataFrame` or `pandas.Series` or `scipy.sparse matrix`

typed_wrapper_func(dataset, keep_index_as_feature=False)

Get a wrapper function to convert the dataset to the original type, pandas DataFrame or Series.

Parameters

- **dataset** (`numpy.array` or `scipy.sparse.csr_matrix`) – The dataset to convert to original type.
- **keep_index_as_feature** (`bool`) – Whether to keep the index as a feature when converting back. Off by default to convert it back to index.

Returns A wrapper function for a given dataset to convert to original type.

Return type `numpy.array` or `scipy.sparse.csr_matrix` or `pandas.DataFrame` or `pandas.Series`

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.

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: object

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: `object`

The explanation mixin for expected values.

Parameters `expected_values` (`numpy.array`) – 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.array`) – The feature importance values in the order of the original features.
- **global_importance_rank** (`numpy.array`) – The feature indexes sorted by importance.
- **ranked_global_names** (`list[str] TODO`) – The feature names sorted by importance.
- **ranked_global_values** (`numpy.array`) – 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.array, 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.array 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.array 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.array, 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.array` 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.array*]) – 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.array`) – The feature importance values for each class in the order of the original features.
- **per_class_importance_rank** (`numpy.array`) – 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.array`) – 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.explanation.load_explanation(path)
interpret_community.explanation.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.

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

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

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

Examples for transformations that work:

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

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

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

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

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

Submodules

interpret_community.lime.lime_explainer module

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

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

Bases: *interpret_community.common.blackbox_explainer.BlackBoxExplainer*

available_explanations = ['global', 'local']

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

Parameters

- **evaluation_examples** (`numpy.array` 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** (`interpret_community.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.array` 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.array* 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.array* 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.array` 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.

Subpackages

interpret_community.mimic.models package

Module for explainable surrogate models.

```
class interpret_community.mimic.models.BaseExplainableModel(**kwargs)  
    Bases: abc.ABC, interpret_community.common.chained_identity.ChainedIdentity
```

The base class for models that can be explained.

abstract property expected_values

Abstract property to get the expected values.

abstract explain_global(kwargs)**

Abstract method to get the global feature importances from the trained explainable model.

abstract explain_local(evaluation_examples, **kwargs)

Abstract method to get the local feature importances from the trained explainable model.

static explainable_model_type()

Retrieve the model type.

```

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

abstract property model
    Abstract property to get the underlying model.

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

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

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

available_explanations = ['global', 'local']

property expected_values
    Use TreeExplainer to get the expected values.

    Returns The expected values of the decision tree tree model.

    Return type list

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

    Returns The global explanation of feature importances.

    Return type list

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

    Parameters
        • evaluation_examples (numpy.array 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.array or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.array

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.

X_idx_sorted [deprecated, default=”deprecated”] This parameter is deprecated and has no effect. It will be removed in 1.1 (renaming of 0.26).

Deprecated since version 0.24.

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.

X_idx_sorted [deprecated, default="deprecated"] This parameter is deprecated and has no effect.
It will be removed in 1.1 (renaming of 0.26).

Deprecated since version 0.24.

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

param labels The labels to train the model on.

type labels numpy.array

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.array 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.array 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.array 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.array or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.array

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.array 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.array or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

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

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

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

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

Parameters

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

Returns

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

References

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

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

Otherwise predict_proba is not supported for regression or binary classification.

Submodules

interpret_community.mimic.models.explainable_model module

Defines the base API for explainable models.

```
class interpret_community.mimic.models.explainable_model.BaseExplainableModel(**kwargs)
Bases: abc.ABC, interpret_community.common.chained_identity.ChainedIdentity
```

The base class for models that can be explained.

abstract property expected_values

Abstract property to get the expected values.

abstract explain_global(kwargs)**

Abstract method to get the global feature importances from the trained explainable model.

abstract explain_local(evaluation_examples, **kwargs)

Abstract method to get the local feature importances from the trained explainable model.

static explainable_model_type()

Retrieve the model type.

abstract fit(kwargs)**

Abstract method to fit the explainable model.

abstract property model

Abstract property to get the underlying model.

abstract predict(dataset, **kwargs)

Abstract method to predict labels using the explainable model.

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

interpret_community.mimic.models.lightgbm_model module

Defines an explainable lightgbm model.

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

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

available_explanations = ['global', 'local']

property expected_values

Use TreeExplainer to get the expected values.

Returns The expected values of the LightGBM tree model.

Return type list

explain_global(kwargs)**

Call lightgbm feature importances to get the global feature importances from the explainable model.

Returns The global explanation of feature importances.

Return type numpy.ndarray

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

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

Parameters

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

param `labels` The labels to train the model on.

type `labels` `numpy.array`

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.array 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.array or pandas.DataFrame or scipy.sparse.csr_matrix

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LogisticRegression:

Probability estimates.

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

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

Parameters

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

Returns

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

Otherwise predict_proba is not supported for regression or binary classification.

class `interpret_community.mimic.models.linear_model.LinearExplainer(*args: Any, **kwargs: Any)`

Bases: `shap.LinearExplainer`

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.array` 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.array 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.array or pandas.DataFrame or scipy.sparse.csr_matrix
param labels The labels to train the model on.
type labels numpy.array

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

Parameters

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

Returns

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

References

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

The justification for the formula in the loss="modified_hubert" 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.array 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.array or pandas.DataFrame or scipy.sparse.csr_matrix

param labels The labels to train the model on.

type labels numpy.array

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.

X_idx_sorted [deprecated, default="deprecated"] This parameter is deprecated and has no effect. It will be removed in 1.1 (renaming of 0.26).

Deprecated since version 0.24.

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.

X_idx_sorted [deprecated, default="deprecated"] This parameter is deprecated and has no effect. It will be removed in 1.1 (renaming of 0.26).

Deprecated since version 0.24.

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.array 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.array 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.array* 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.array` 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.array` 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.

interpret_community.mimic.model_distill module

Utilities to train a surrogate model from teacher.

interpret_community.mlflow package

Submodules

interpret_community.mlflow.mlflow module

interpret_community.permutation package

Module for permutation feature importance.

```
class interpret_community.permutation.PFIEExplainer(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.array` 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.array` or `pandas.DataFrame`) – An array of true labels used for reference to compute the evaluation metric for base case and after each permutation.

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

Return type DynamicGlobalExplanation

explainer_type = 'blackbox'

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

Parameters

- **model** (`object`) – The black box model or function (if `is_function` is True) to be explained. Also known as the teacher model. A model that implements `sklearn.predict` or `sklearn.predict_proba` or function that accepts a 2d ndarray.
- **is_function** (`bool`) – Default is False. Set to True if passing function instead of model.
- **metric** (`str or function that accepts two arrays, y_true and y_pred`) – The metric name or function to evaluate the permutation. Note that if a metric function is provided, a higher value must be better. Otherwise, take the negative of the function or set `is_error_metric` to True. By default, if no metric is provided, F1 Score is used for binary classification, F1 Score with micro average is used for multiclass classification and mean absolute error is used for regression.
- **metric_args** (`dict`) – Optional arguments for metric function.
- **is_error_metric** (`bool`) – If custom metric function is provided, set to True if a higher value of the metric is better.
- **explain_subset** (`list[int]`) – List of feature indexes. If specified, only selects a subset of the features in the evaluation dataset for explanation. For permutation feature importance, we can shuffle, score and evaluate on the specified indexes when this parameter is set. This argument is not supported when transformations are set.
- **features** (`list[str]`) – A list of feature names.
- **classes** (`list[str]`) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **transformations** (`sklearn.compose.ColumnTransformer or list[tuple]`) – `sklearn.compose.ColumnTransformer` or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: <https://github.com/scikit-learn-contrib/sklearn-pandas>.

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

Examples for transformations that work:

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

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

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

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

- **allow_all_transformations** (`bool`) – Allow many to many and many to one transformations.
- **seed** (`int`) – Random number seed for shuffling.
- **for_classifier_use_predict_proba** (`bool`) – If specifying a model instead of a function, and the model is a classifier, set to True instead of the default False to use predict_proba instead of predict when calculating the metric.
- **show_progress** (`bool`) – Default to ‘True’. Determines whether to display the explanation status bar when using PFIExplainer.
- **model_task** (`str`) – Optional parameter to specify whether the model is a classification or regression model. In most cases, the type of the model can be inferred based on the shape of the output, where a classifier has a predict_proba method and outputs a 2 dimensional array, while a regressor has a predict method and outputs a 1 dimensional array.

Submodules

interpret_community.permutation.metric_constants module

Defines metric constants for PFIExplainer.

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

The metric to use for PFIExplainer.

AVERAGE_PRECISION_SCORE = 'average_precision_score'
EXPLAINED_VARIANCE_SCORE = 'explained_variance_score'
F1_SCORE = 'f1_score'
FBETA_SCORE = 'fbeta_score'
MEAN_ABSOLUTE_ERROR = 'mean_absolute_error'
MEAN_SQUARED_ERROR = 'mean_squared_error'
MEAN_SQUARED_LOG_ERROR = 'mean_squared_log_error'
MEDIAN_ABSOLUTE_ERROR = 'median_absolute_error'
PRECISION_SCORE = 'precision_score'
R2_SCORE = 'r2_score'
RECALL_SCORE = 'recall_score'
```

interpret_community.permutation.permutation_importance module

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

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

```
class interpret_community.permutation.permutation_importance.PFIExplainer(model,
                           is_function=False,
                           metric=None,
                           metric_args=None,
                           is_error_metric=False,
                           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.array` 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.array` 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.array` 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.array* 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.array* 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  
]  
[
```

(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
- **model_task** (*str*) – Optional parameter to specify whether the model is a classification or regression model.

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

Bases: interpret_community.common.blackbox_explainer.BlackBoxExplainer

available_explanations = ['global', 'local']

explain_global(evaluation_examples, sampling_policy=None, include_local=True, batch_size=100)
    Explain the model globally by aggregating local explanations to global. :param evaluation_examples: A matrix of feature vector examples (# examples x # features) on which
        to explain the model's output.
```

Parameters

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

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

Return type DynamicGlobalExplanation

explain_local(*evaluation_examples*)

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

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

Return type DynamicLocalExplanation

explainer_type = 'blackbox'

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

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

Characteristics of the GPU version:

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

Parameters

- **model** (*object*) – Function that takes a matrix of samples (n_samples, n_features) and computes the output for those samples with shape (n_samples).
- **initialization_examples** (*numpy.array* 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.array` 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.array` 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.array` 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.array` 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.array 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

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

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

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

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

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

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

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

interpret_community.shap.kernel_explainer module

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

```
class interpret_community.shap.kernel_explainer.KernelExplainer(model, initialization_examples,  
                                                               is_function=False,  
                                                               explain_subset=None,  
                                                               nsamples='auto', features=None,  
                                                               classes=None, nclusters=10,  
                                                               show_progress=True,  
                                                               transformations=None, al-  
                                                               low_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.array` 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.array` or `pandas.DataFrame` or `scipy.sparse.csr_matrix`) – A matrix of feature vector examples (# examples x # features) for initializing the explainer.
- **is_function** (`bool`) – Default is False. Set to True if passing function instead of a model.
- **explain_subset** (`list[int]`) – List of feature indices. If specified, only selects a subset of the features in the evaluation dataset for explanation, which will speed up the explanation process when number of features is large and the user already knows the set of interested features. The subset can be the top-k features from the model summary.
- **nsamples** ('`auto`' or `int`) – Default to ‘auto’. Number of times to re-evaluate the model when explaining each prediction. More samples lead to lower variance estimates of the feature importance values, but incur more computation cost. When ‘auto’ is provided, the number of samples is computed according to a heuristic rule.
- **features** (`list[str]`) – A list of feature names.

- **classes** (`list[str]`) – Class names as a list of strings. The order of the class names should match that of the model output. Only required if explaining classifier.
- **nclusters** (`int`) – Number of means to use for approximation. A dataset is summarized with nclusters mean samples weighted by the number of data points they each represent. When the number of initialization examples is larger than (10 x nclusters), those examples will be summarized with k-means where k = nclusters.
- **show_progress** (`bool`) – Default to ‘True’. Determines whether to display the explanation status bar when using `shap_values` from the KernelExplainer.
- **transformations** (`sklearn.compose.ColumnTransformer or list[tuple]`) – `sklearn.compose.ColumnTransformer` or a list of tuples describing the column name and transformer. When transformations are provided, explanations are of the features before the transformation. The format for a list of transformations is same as the one here: <https://github.com/scikit-learn-contrib/sklearn-pandas>.

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

Examples for transformations that work:

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

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

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

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

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

interpret_community.shap.kwargs_utils module

Defines utilities for handling kwargs on SHAP-based explainers.

interpret_community.shap.linear_explainer module

Defines the LinearExplainer for returning explanations for linear models.

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

Bases: [interpret_community.common.structured_model_explainer.StructuredInitModelExplainer](#)

available_explanations = ['global', 'local']

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

Explain the model globally by aggregating local explanations to global.

Parameters

- **evaluation_examples** ([numpy.array](#) 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.array* 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.array` 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.array or list[][]*) – A matrix of feature vector examples (# examples x # features), the same samples used to build the explanation. Overwrites any existing dataset on the explanation object. Must have fewer than 10000 rows and fewer than 1000 columns.
- **datasetX** (*numpy.array or list[][]*) – Alias of the dataset parameter. If dataset is passed, this will have no effect. Must have fewer than 10000 rows and fewer than 1000 columns.
- **true_y** (*numpy.array or list[]*) – The true labels for the provided dataset. Overwrites any existing dataset on the explanation object.
- **classes** (*numpy.array or list[]*) – The class names.
- **features** (*numpy.array 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.array` or `list[][]`) – A matrix of feature vector examples (# examples x # features), the same samples used to build the explanation. Overwrites any existing dataset on the explanation object. Must have fewer than 10000 rows and fewer than 1000 columns.
- **datasetX** (`numpy.array` or `list[][]`) – Alias of the dataset parameter. If dataset is passed, this will have no effect. Must have fewer than 10000 rows and fewer than 1000 columns.
- **true_y** (`numpy.array` or `list[]`) – The true labels for the provided dataset. Overwrites any existing dataset on the explanation object.
- **classes** (`numpy.array` or `list[]`) – The class names.
- **features** (`numpy.array` 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.

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

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

Examples for transformations that work:

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

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

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

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

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

`interpret_community.version` module

**CHAPTER
TWO**

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

interpret_community, 3
interpret_community.adapter, 5
interpret_community.adapter.explanation_adapter, 6
interpret_community.common, 7
interpret_community.common.aggregate, 7
interpret_community.common.base_explainer, 8
interpret_community.common.blackbox_explainer, 8
interpret_community.common.chained_identity, 9
interpret_community.common.constants, 9
interpret_community.common.error_handling, 16
interpret_community.common.exception, 16
interpret_community.common.explanation_utils, 16
interpret_community.common.gpu_kmeans, 16
interpret_community.common.metrics, 16
interpret_community.common.model_summary, 17
interpret_community.common.model_wrapper, 18
interpret_community.common.policy, 19
interpret_community.common.progress, 20
interpret_community.common.serialization_utils, 21
interpret_community.common.structured_model_explainer, 21
interpret_community.dataset, 21
interpret_community.dataset.dataset_wrapper, 21
interpret_community.dataset.decorator, 25
interpret_community.explanation, 25
interpret_community.explanation.explanation, 25
interpret_community.lime, 33
interpret_community.lime.lime_explainer, 35
interpret_community.mimic, 38
interpret_community.mimic.mimic_explainer, 63
interpret_community.mimic.model_distill, 66
interpret_community.mimic.models, 40
interpret_community.mimic.models.explainable_model, 51
interpret_community.mimic.models.lightgbm_model, 52
interpret_community.mimic.models.linear_model, 53
interpret_community.mimic.models.tree_model, 59
interpret_community.mimic.models.tree_model_utils, 63
interpret_community.permutation, 66
interpret_community.permutation.metric_constants, 68
interpret_community.permutation.permutation_importance, 69
interpret_community.shap, 71
interpret_community.shap.deep_explainer, 81
interpret_community.shap.gpu_kernel_explainer, 84
interpret_community.shap.kernel_explainer, 86
interpret_community.shap.kwargs_utils, 89
interpret_community.shap.linear_explainer, 89
interpret_community.shap.tree_explainer, 91
interpret_community.tabular_explainer, 95
interpret_community.version, 97
interpret_community.widget, 93
interpret_community.widget.explanation_dashboard,

INDEX

A

add_explain_global_method() (in module <code>interpret_community.common.aggregate</code>), 7	<code>pret_community.mimic.mimic_explainer.MimicExplainer</code> attribute), 63
add_from_get_model_summary() (interpret_community.common.model_summary. <code>ModelSummary</code>) (method), 17	<code>available_explanations</code> (interpret_community.mimic. <code>MimicExplainer</code> attribute), 38
add_from_get_model_summary() (interpret_community.common. <code>ModelSummary</code>) (method), 7	<code>available_explanations</code> (interpret_community.mimic.models. <code>DecisionTreeExplainableModel</code> attribute), 41
add_prepare_function_and_summary_method() (in module interpret_community.common.blackbox_explainer), 9	<code>available_explanations</code> (interpret_community.mimic.models. <code>LGBMExplainableModel</code> attribute), 44
ALLOW_ALL_TRANSFORMATIONS (interpret_community.common.constants. <code>MimicSerializationConstants</code>) (attribute), 14	<code>available_explanations</code> (interpret_community.mimic.models. <code>linear_model.LinearExplainableModel</code> attribute), 52
allow_eval_sampling (interpret_community.common.policy. <code>SamplingPolicy</code>) (property), 20	<code>available_explanations</code> (interpret_community.mimic.models. <code>linear_model.SGDExplainableModel</code> attribute), 53
apply_indexer() (interpret_community.dataset.dataset_wrapper. <code>DatasetWrapper</code>) (method), 22	<code>available_explanations</code> (interpret_community.mimic.models. <code>LinearExplainableModel</code> attribute), 57
apply_one_hot_encoder() (interpret_community.dataset.dataset_wrapper. <code>DatasetWrapper</code>) (method), 22	<code>available_explanations</code> (interpret_community.mimic.models. <code>SGDExplainableModel</code> attribute), 46
apply_timestamp_featurizer() (interpret_community.dataset.dataset_wrapper. <code>DatasetWrapper</code>) (method), 22	<code>available_explanations</code> (interpret_community.mimic.models. <code>tree_model.DecisionTreeExplainableModel</code> attribute), 48
Attributes (class in interpret_community.common.constants), 9	<code>available_explanations</code> (interpret_community.mimic.models. <code>SGDExplainableModel</code> attribute), 59
augment_data() (interpret_community.dataset.dataset_wrapper. <code>DatasetWrapper</code>) (method), 22	<code>available_explanations</code> (interpret_community.permutation.permutation_importance. <code>PFIExplainer</code> attribute), 69
AUTO (interpret_community.common.constants. <code>Defaults</code>) (attribute), 10	<code>available_explanations</code> (interpret_community.permutation. <code>PFIExplainer</code> attribute), 66
available_explanations (interpret_community.lime.lime_explainer. <code>LIMEExplainer</code>) (attribute), 35	<code>available_explanations</code> (interpret_community.shap.deep_explainer. <code>DeepExplainer</code> attribute), 81
available_explanations (interpret_community.lime. <code>LIMEExplainer</code>) (attribute), 33	<code>available_explanations</code> (interpret_community.shap. <code>DeepExplainer</code> attribute), 71
available_explanations (interpret_community.shap. <code>DeepExplainer</code>)	<code>available_explanations</code> (interpret_community.shap. <code>DeepExplainer</code> attribute), 71

available_explanations	(interpretation), 73	BlackBoxExplainer	(class in interpret_community.common.blackbox_explainer), 8
available_explanations	(interpretation), 86	BlackBoxMixin	(class in interpret_community.common.blackbox_explainer), 8
available_explanations	(interpretation), 75	CATEGORICAL_FEATURE	(interpretation), 13
available_explanations	(interpretation), 89	ChainedIdentity	(class in interpret_community.common.chained_identity), 9
available_explanations	(interpretation), 78	CLASSES	(interpret_community.common.constants.ExplainParams attribute), 10
available_explanations	(interpretation), 91	CLASSES	(interpret_community.common.constants.ExplanationParams attribute), 12
available_explanations	(interpretation), 79	ClassesMixin	(class in interpret_community.explanation.explanation), 26
available_explanations	(interpretation), 95	CLASSIFICATION	(interpret_community.common.constants.ExplainParams attribute), 10
available_explanations	(interpretation), 3	CLASSIFICATION	(interpret_community.common.constants.ExplainType attribute), 11
AVERAGE_PRECISION_SCORE	(interpretation), 68	Classification	(interpret_community.common.constants.ModelTask attribute), 14
B		close()	(interpret_community.shap.deep_explainer.logger_redirector method), 83
BASE_VALUE (interpret_community.common.constants.Interpretation)	(attribute), 13	create_summary()	(interpret_community.dataset.dataset_wrapper.DatasetWrapper method), 22
BaseExplainableModel	(class in interpret_community.mimic.models), 40	CPU0	(interpret_community.common.constants.Tensorflow attribute), 15
BaseExplainableModel	(class in interpret_community.mimic.models.explainable_model), 51	create_global()	(interpret_community.adapter.explanation_adapter.ExplanationAdapter method), 6
BaseExplainer	(class in interpret_community.common.base_explainer), 8	create_global()	(interpret_community.adapter.ExplanationAdapter method), 5
BaseExplanation	(class in interpret_community.explanation.explanation), 25	create_local()	(interpret_community.adapter.explanation_adapter.ExplanationAdapter method), 6
BaseWrappedModel	(class in interpret_community.common.model_wrapper), 18	create_local()	(interpret_community.adapter.ExplanationAdapter method), 5
BATCH_SIZE (interpret_community.common.constants.ExplainParam)	(attribute), 10	CSR_FORMAT (interpret_community.common.constants.Scipy attribute), 15	
BLACKBOX (interpret_community.common.constants.Extension)	(attribute), 12	CustomTimestampFeaturizer	(class in interpret_community.common.extension.CustomTimestampFeaturizer), 12

pret_community.dataset.dataset_wrapper), 21

D

Data (*class in interpret_community.common.gpu_kmeans*), **EVAL_Y_PRED_PROBA** (*interpret_community.common.constants.ExplainParams attribute*), 10
DATA (*interpret_community.common.constants.ExplainType attribute*), 11
data() (*interpret_community.explanation.explanation.BaseExplanation attribute*), 15
data() (*interpret_community.explanation.explanation.ExpectedValueMixin attribute*), 27
data() (*interpret_community.explanation.explanation.GlobalExplainer attribute*), 28
data() (*interpret_community.explanation.explanation.LocalExplainer attribute*), 29
dataset (*interpret_community.dataset.dataset_wrapper.DatasetWrapper attribute*), 12
DatasetWrapper (*class in interpret_community.dataset.dataset_wrapper*), 22
dcg() (*in module interpret_community.common.metrics*), 16
DecisionTreeExplainableModel (*class in interpret_community.mimic.models*), 41
DecisionTreeExplainableModel (*class in interpret_community.mimic.models.tree_model*), 59
DeepExplainer (*class in interpret_community.shap*), 71
DeepExplainer (*class in interpret_community.shap.deep_explainer*), 81
DEFAULT (*interpret_community.common.constants.ShapValuesOutput attribute*), 15
DEFAULT_BATCH_SIZE (*interpret_community.common.constants.Defaults attribute*), 10
Defaults (*class in interpret_community.common.constants*), 9
DenseData (*class in interpret_community.common.gpu_kmeans*), 16
DNNFramework (*class in interpret_community.common.constants*), 9
Dynamic (*class in interpret_community.common.constants*), 10

E

EN (*interpret_community.common.constants.Spacy attribute*), 15
enum_properties (*interpret_community.common.constants.LightGBMSerializationConstants attribute*), 13
enum_properties (*interpret_community.common.constants.MimicSerializationConstants attribute*), 14

EVAL_DATA (*interpret_community.common.constants.ExplainParams attribute*), 10
EVAL_Y_PRED (*interpret_community.common.constants.ExplainParams attribute*), 10
EXAMPLES (*interpret_community.common.constants.SKLearnAttribute*), 9
EXPECTED_VALUES (*interpret_community.common.constants.ExplainParams attribute*), 10
EXPECTED_VALUES (*interpret_community.common.constants.ExplainParams attribute*), 10
expected_values (*interpret_community.explanation.explanation.ExpectedValueMixin property*), 27
expected_values (*interpret_community.mimic.models.BaseExplainableModel property*), 40
expected_values (*interpret_community.mimic.models.DecisionTreeExplainableModel property*), 41
expected_values (*interpret_community.mimic.models.explainable_model.BaseExplainableModel property*), 51
expected_values (*interpret_community.mimic.models.lightgbm_model.LGBMExplainableModel property*), 44
expected_values (*interpret_community.mimic.models.linear_model.LinearExplainableModel property*), 52
expected_values (*interpret_community.mimic.models.linear_model.LinearExplainableModel property*), 53
expected_values (*interpret_community.mimic.models.linear_model.SGDExplainableModel property*), 57
expected_values (*interpret_community.mimic.models.LinearExplainableModel property*), 46
expected_values (*interpret_community.mimic.models.SGDExplainableModel property*), 48
expected_values (*interpret_community.mimic.models.tree_model.DecisionTreeExplainableModel property*), 59
ExpectedValuesMixin (*class in interpret_community.explanation.explanation*), 26

```
EXPLAIN(interpret_community.common.constants.ExplainType      pret_community.shap.deep_explainer.DeepExplainer  
attribute), 11                                         method), 81  
explain_global()                                         (interpret_community.common.base_explainer.GlobalExplainer pret_community.shap.DeepExplainer method),  
method), 8                                              71  
explain_global()                                         (interpret_community.lime.lime_explainer.LIMEExplainer pret_community.shap.gpu_kernel_explainer.GPUKernelExplaine  
method), 35                                              method), 84  
explain_global()                                         (interpret_community.lime.LIMEExplainer method), 33          (interpret_community.shap.gpu_kernel_explainer.GPUKernelExplaine  
method), 73  
explain_global()                                         (interpret_community.mimic.mimic_explainer.MimicExplainer pret_community.shap.kernel_explainer.KernelExplainer  
method), 63                                              method), 86  
explain_global()                                         (interpret_community.mimic.MimicExplainer pret_community.shap.KernelExplainer  
method), 38                                              method), 75  
explain_global()                                         (interpret_community.mimic.models.BaseExplainableModel pret_community.shap.linear_explainer.LinearExplainer  
method), 40                                              method), 89  
explain_global()                                         (interpret_community.mimic.models.DecisionTreeExplainableMopret_community.shap.LinearExplainer  
method), 41                                              method), 78  
explain_global()                                         (interpret_community.mimic.models.explainable_model.BaseExplaipret_community.shap.tree_explainer.TreeExplainer  
nableModel), 51                                              method), 91  
explain_global()                                         (interpret_community.mimic.models.LGBMExplainableModel pret_community.shap.TreeExplainer  
method), 44                                              method), 79  
explain_global()                                         (interpret_community.mimic.models.lightgbm_model.LGBMExplaipret_community.shap.tabular_explainer.TabularExplainer  
nableModel), 52                                              method), 95  
explain_global()                                         (interpret_community.mimic.models.linear_model.LinearExplaipret_community.shap.tabular_explainer.TabularExplainer  
nableModel), 54                                              method), 3  
explain_global()                                         (interpret_community.mimic.models.linear_model.SGDExplainabpret_community.common.base_explainer.LocalExplainer  
leModel), 57                                              method), 8  
explain_global()                                         (interpret_community.mimic.models.LinearExplainableModel pret_community.lime.lime_explainer.LIMEExplainer  
method), 46                                              method), 36  
explain_global()                                         (interpret_community.mimic.models.SGDExplainableModel pret_community.lime.LIMEExplainer  
method), 49                                              method), 33  
explain_global()                                         (interpret_community.mimic.models.tree_model.DecisionTreeExplaipret_community.mimic.mimic_explainer.MimicExplainer  
nableModel), 60                                              method), 64  
explain_global()                                         (interpret_community.mimic.models.PFIExplaipret_community.mimic.MimicExplainer  
nableModel), 69                                              method), 38  
explain_global()                                         (interpret_community.permutation.permutation_importance.PFIExplaipret_community.mimic.MimicExplainer  
nableModel), 66                                              method), 40  
explain_global()                                         (interpret_community.permutation.PFIExplainer pret_community.mimic.models.BaseExplainableModel  
method), 66                                              method), 40  
explain_global()                                         (interpret_community.shap.deep_explainer.DeepExplainer  
method), 81
```

```

pret_community.mimic.models.DecisionTreeExplainableModel.explain_local()
method), 41                                         (inter- explain_local()          (inter-
                                                    pret_community.mimic.models.explainable_model.BaseExplainerMethod.tabular_explainer.TabularExplainer
method), 51                                         method), 95

explain_local()                                     (inter- explain_local()          (inter-
                                                    pret_community.mimic.models.LGBMExplainableModel   pret_community.TabularExplainer    method),
method), 44                                         3

explain_local()                                     (inter- EXPLAIN_SUBSET        (inter-
                                                    pret_community.mimic.models.lightgbm_model.LGBMExplainerMethod.common.constants.ExplainParams
method), 52                                         attribute), 10

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.mimic.models.linear_model.LinearExplainerMethod.mimic.models.BaseExplainableModel
method), 54                                         static method), 40

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.mimic.models.linear_model.SGDExplainableModel   pret_community.mimic.models.DecisionTreeExplainableModel
method), 57                                         static method), 41

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.mimic.models.LinearExplainableModel   pret_community.mimic.models.explainable_model.BaseExplainableModel
method), 46                                         static method), 51

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.mimic.models.SGDExplainableModel   pret_community.mimic.models.LGBMExplainableModel
method), 49                                         static method), 45

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.mimic.models.tree_model.DecisionTreeExplainerMethod.mimic.models.lightgbm_model.LGBMExplainableModel
method), 60                                         static method), 52

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.shap.deep_explainer.DeepExplainer   pret_community.mimic.models.linear_model.LinearExplainableModel
method), 82                                         static method), 54

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.shap.DeepExplainer method),   pret_community.mimic.models.LinearExplainableModel
71                                                 static method), 46

explain_local()                                     (inter- explainable_model_type() (inter-
                                                    pret_community.shap.gpu_kernel_explainer.GPUKernelExplainer   pret_community.mimic.models.tree_model.DecisionTreeExplainableModel
method), 84                                         static method), 60

explain_local()                                     (inter- ExplainableModelType   (class   in   inter-
                                                    pret_community.shap.GPUKernelExplainer   pret_community.common.constants), 12
method), 73                                         EXPLAINED_VARIANCE_SCORE (inter-
                                                    pret_community.permutation.metric_constants.MetricConstants

explain_local()                                     (inter- KernelExplainer     (attribute), 68
                                                    pret_community.shap.kernel_explainer.KernelExplainer   attribute), 11

explain_local()                                     (inter- KernelExplainer     (attribute), 36
                                                    pret_community.shap.KernelExplainer   attribute), 11

explain_local()                                     (inter- LinearExplainer     (attribute), 34
                                                    pret_community.shap.linear_explainer.LinearExplainer   attribute), 34

explain_local()                                     (inter- LinearExplainer     (attribute), 64
                                                    pret_community.shap.LinearExplainer   attribute), 64

explain_local()                                     (inter- TreeExplainer      (attribute), 38
                                                    pret_community.shap.tree_explainer.TreeExplainer   attribute), 38

```

explainer_type (interpret_community.mimic.models.DecisionTreeExplainableModel.explainer_type, 41) (interpret_community.shap.tree_explainer.TreeExplainer.attribute), 91

explainer_type (interpret_community.mimic.models.LGBMExplainableModel.explainer_type, 45) (interpret_community.shap.TreeExplainer.attribute), 80

explainer_type (interpret_community.mimic.models.lightgbm_model.LGBMExplainer.explainer_type, 52) (interpret_community.tabular_explainer.TabularExplainer.attribute), 96

explainer_type (interpret_community.mimic.models.linear_model.LinearExplainer.explainer_type, 54) (interpret_community.tabular_explainer.TabularExplainer.attribute), 4

explainer_type (interpret_community.mimic.models.linear_model.SGDExplainableModel.explainer_type, 57) (interpret_community.common.constants.ExplainType.attribute), 10

explainer_type (interpret_community.mimic.models.LinearExplainableModel.explainer_type, 46) (interpret_community.common.constants.Explanation_CLASS_DIMENSION.attribute), 11

explainer_type (interpret_community.mimic.models.SGDExplainableModel.explainer_type, 49) (interpret_community.common.constants.Explanation_ID.attribute), 13

explainer_type (interpret_community.mimic.models.tree_model.DecisionTreeExplainableModel.explainer_type, 60) (interpret_community.common.constants.ExPLANATION_TYPE.attribute), 13

explainer_type (interpret_community.permutation.permutation_importance.ExPLAIN_ADAPTER.explainer_type, 69) (interpret_community.adapter.ExplainParams.attribute), 10

explainer_type (interpret_community.permutation.PFIEExplainer.explainer_type, 66) (interpret_community.adapter.ExplanationAdapter.attribute), 5

explainer_type (interpret_community.shap.deep_explainer.DeepExplainer.explainer_type, 82) (interpret_community.widget.ExplanationDashboard.attribute), 93

explainer_type (interpret_community.shap.DeepExplainer.explainer_type, 72) (interpret_community.widget.ExplanationDashboard.attribute), 94

explainer_type (interpret_community.shap.gpu_kernel_explainer.GPUExplainer.explainer_type, 84) (interpret_community.common.constants.ExplanationParams.attribute), 12

explainer_type (interpret_community.shap.GPUKernelExplainer.explainer_type, 74) (interpret_community.common.constants.ExTRA(attribute), 13)

explainer_type (interpret_community.shap.kernel_explainer.KernelExplainer.explainer_type, 87) (interpret_community.permutation.metric_constants.MetricConsistencyScore.attribute), 68

explainer_type (interpret_community.shap.KernelExplainer.explainer_type, 76) (interpret_community.permutation.metric_constants.MetricFBetaScore.attribute), 68

explainer_type (interpret_community.shap.linear_explainer.LinearExplainer.explainer_type, 89) (interpret_community.common.constants.ExplanationImportanceList.attribute), 13

explainer_type (interpret_community.shap.LinearExplainer.explainer_type, 78) (interpret_community.explanation.ExplanationImportanceExplanation.attribute), 27

explainer_type (interpret_community.shap.LinearExplainer.explainer_type, 78) (interpret_community.common.constants.Features.attribute), 10

F

features (*interpret_community.explanation.explanation.Features*) (interpret-property), 27
fit() (*interpret_community.dataset.dataset_wrapper.CustomTimestampFilterizer* method), 21
fit() (*interpret_community.mimic.models.BaseExplainableModel* method), 40
fit() (*interpret_community.mimic.models.DecisionTreeExplainer*) (interpret-method), 42
fit() (*interpret_community.mimic.models.explainable_model.BaseExplainerModel* method), 51
fit() (*interpret_community.mimic.models.LGBMExplainableModel* method), 45
fit() (*interpret_community.mimic.models.lightgbm_model.LightGBMExplainer*) (interpret-method), 53
fit() (*interpret_community.mimic.models.linear_model.LinearExplainerModel* method), 54
fit() (*interpret_community.mimic.models.SGDExplainableModel* method), 57
fit() (*interpret_community.mimic.models.LinearExplainerModel*) (interpret-method), 46
fit() (*interpret_community.mimic.models.SGDExplainableModel* method), 49
fit() (*interpret_community.mimic.models.tree_model.DecisionTreeExplainer*) (interpret-method), 60
flush() (*interpret_community.shap.deep_explainer.logger*) (interpret-method), 83
FUNCTION (*interpret_community.common.constants.ExplainType* attribute), 11
FUNCTION (*interpret_community.common.constants.MimicSerialization* attribute), 14

G
get_artifacts() (interpret-method), 17
get_artifacts() (interpret-method), 7
get_column_indexes() (interpret-method), 22
get_feature_importance_dict() (interpret-method), 28
get_features() (interpret-method), 23
get_local_importance_rank() (interpret-method), 30
get_metadata_dictionary() (interpret-method), 17

get_private() (*interpret_community.common.ModelSummary* class method), 11
get_ranked_global_names() (*interpret_community.explanation.explanation.GlobalExplanation* method), 28
get_ranked_global_values() (interpret-method), 20
get_ranked_local_values() (interpret-method), 20
get_ranked_per_class_names() (interpret-method), 30
get_ranked_per_class_values() (interpret-method), 32
get_raw_explanation() (interpret-method), 28
get_raw_feature_importances() (interpret-method), 30
get_raw_feature_importances() (interpret-method), 29
get_raw_feature_importances() (interpret-method), 31
get_serializable() (interpret-method), 11
get_tdm() (in module *interpret_community.common.progress*), 20
GLASSBOX (*interpret_community.common.constants.ExtensionAttribute*), 12
GLOBAL (*interpret_community.common.constants.ExplainTypeAttribute*), 11
GLOBAL (*interpret_community.common.constants.ExtensionAttribute*), 12
GLOBAL_EXPLANATION (interpret-method), 10
GLOBAL FEATURE IMPORTANCE (interpret-method), 13
GLOBAL IMPORTANCE NAMES (interpret-method), 13


```

    module, 21
interpret_community.common.structured_model_explainer module, 91
    module, 21
interpret_community.dataset module, 95
    module, 21
interpret_community.dataset.dataset_wrapper module, 97
    module, 21
interpret_community.dataset.decorator module, 98
    module, 25
interpret_community.explanation module, 99
    module, 25
interpret_community.explanation.explanation module, 100
    module, 25
interpret_community.lime module, 101
    module, 33
interpret_community.lime.lime_explainer module, 102
    module, 35
interpret_community.mimic module, 103
    module, 38
interpret_community.mimic.mimic_explainer module, 104
    module, 63
interpret_community.mimic.model_distill module, 105
    module, 66
interpret_community.mimic.models module, 106
    module, 40
interpret_community.mimic.models.explainable_model module, 107
    module, 51
interpret_community.mimic.models.lightgbm_model module, 108
    module, 52
interpret_community.mimic.models.linear_model module, 109
    module, 53
interpret_community.mimic.models.tree_model module, 110
    module, 59
interpret_community.mimic.models.tree_model_utils module, 111
    module, 63
interpret_community.permutation module, 112
    module, 66
interpret_community.permutation.metric_constants module, 113
    module, 68
interpret_community.permutation.permutation_importance module, 114
    module, 69
interpret_community.shap module, 115
    module, 71
interpret_community.shap.deep_explainer module, 116
    module, 81
interpret_community.shap.gpu_kernel_explainer module, 117
    module, 84
interpret_community.shap.kernel_explainer module, 118
    module, 86
interpret_community.shap.kwargs_utils module, 119
    module, 89
interpret_community.shap.linear_explainer module, 120
    module, 89
interpret_community.shap.tree_explainer module, 121
    module, 91
interpret_community.tabular_explainer module, 95
    module, 21
interpret_community.version module, 97
    module, 21
interpret_community.widget module, 98
    module, 21
interpret_community.widget.explanation_dashboard module, 99
    module, 94
InterpretData (class in interpret_community.common.constants), 12
IS_ENG(interpret_community.common.constants.ExplainParams attribute), 10
IS_ENG(interpret_community.common.constants.ExplainType attribute), 12
is_engineered (interpret_community.explanation.explanation.FeatureImportanceExplainer property), 27
IS_LOCAL_SPARSE (interpret_community.common.constants.ExplainParams attribute), 10
is_local_sparse (interpret_community.explanation.explanation.LocalExplanation property), 31
IS_RAW(interpret_community.common.constants.ExplainParams attribute), 10
IS_RAW(interpret_community.common.constants.ExplainType attribute), 12
is_raw(interpret_community.explanation.explanation.FeatureImportanceExplainer property), 27
K
KernelExplainer (class in interpret_community.shap), 75
    module, 75
KernelExplainer (class in interpret_community.shap.kernel_explainer), 86
    module, 86
kmeans() (in interpret_community.common.gpu_kmeans), 16
    module, 16
L
LABELS (interpret_community.common.constants.SKLearn attribute), 15
labels_decorator() (in interpret_community.permutation.permutation_importance), 71
    module, 71
LGBMExplainableModel (class in interpret_community.mimic.models), 44
    module, 44
LGBMExplainableModel (class in interpret_community.mimic.models.lightgbm_model), 52
    module, 52
LightGBMParams (class in interpret_community.common.constants), 13
    module, 13

```

LightGBMSerializationConstants (class in interpret_community.common.constants), 13
LIME (interpret_community.common.constants.ExplainType attribute), 12
LIMEExplainer (class in interpret_community.lime), 33
LIMEExplainer (class in interpret_community.lime.lime_explainer), 35
LINEAR_EXPLAINABLE_MODEL_TYPE (interpret_community.common.constants.ExplainableModelType attribute), 12
LinearExplainableModel (class in interpret_community.mimic.models), 46
LinearExplainableModel (class in interpret_community.mimic.models.linear_model), 53
LinearExplainer (class in interpret_community.mimic.models.linear_model), 56
LinearExplainer (class in interpret_community.shap), 77
LinearExplainer (class in interpret_community.shap.linear_explainer), 89
load_explanation() (in module interpret_community.explanation.explanation), 33
LOCAL (interpret_community.common.constants.ExplainType attribute), 12
LOCAL (interpret_community.common.constants.Extension attribute), 12
LOCAL_EXPLANATION (interpret_community.common.constants.Dynamic attribute), 10
LOCAL_EXPLANATION (interpret_community.common.constants.ExplainParams attribute), 10
LOCAL_FEATURE_IMPORTANCE (interpret_community.common.constants.InterpretData attribute), 13
LOCAL_IMPORTANCE_VALUES (interpret_community.common.constants.ExplainParams attribute), 11
local_importance_values (interpret_community.explanation.explanation.LocalExplanation property), 31
LocalExplainer (class in interpret_community.common.base_explainer), 8
LocalExplanation (class in interpret_community.explanation.explanation), 29
LOGGER (interpret_community.common.constants.LightGBMSerializationConstants attribute), 13
LOGGER (interpret_community.common.constants.MimicSerializationConstants property), 45
attribute), 14
logger_redirector (class in interpret_community.shap.deep_explainer), 83

M

MAX_DIM (interpret_community.common.constants.Defaults attribute), 10
max_dim_clustering (interpret_community.common.policy.SamplingPolicy property), 20
MEAN_ABSOLUTE_ERROR (interpret_community.permutation.metric_constants.MetricConstants attribute), 68
MEAN_SQUARED_ERROR (interpret_community.permutation.metric_constants.MetricConstants attribute), 68
MEAN_SQUARED_LOG_ERROR (interpret_community.permutation.metric_constants.MetricConstants attribute), 68
MEDIAN_ABSOLUTE_ERROR (interpret_community.permutation.metric_constants.MetricConstants attribute), 68
METHOD (interpret_community.common.constants.ExplainParams attribute), 11
METHOD (interpret_community.common.constants.ExplainType attribute), 12
method (interpret_community.explanation.explanation.BaseExplanation property), 26
MetricConstants (class in interpret_community.permutation.metric_constants), 68
MIMIC (interpret_community.common.constants.ExplainType attribute), 12
MimicExplainer (class in interpret_community.mimic), 38
MimicExplainer (class in interpret_community.mimic.mimic_explainer), 63
MimicSerializationConstants (class in interpret_community.common.constants), 14
MLI (interpret_community.common.constants.InterpretData attribute), 13
MODEL (interpret_community.common.constants.ExplainType attribute), 12
MODEL (interpret_community.common.constants.MimicSerializationConstants attribute), 14
model (interpret_community.mimic.models.BaseExplainableModel property), 41
model (interpret_community.mimic.models.DecisionTreeExplainableModel property), 43
model (interpret_community.mimic.models.explainable_model.BaseExplainableModel property), 43
model (interpret_community.mimic.models.LGBMExplainableModel property), 45

```

model (interpret_community.mimic.models.lightgbm_model.LGBMExplainableModel.common.gpu_kmeans, 16
      property), 53
model (interpret_community.mimic.models.linear_model.LinearExplainerModel.common.model_summary,
      property), 55
model (interpret_community.mimic.models.linear_model.SGDExplainerModel.common.model_wrapper,
      property), 58
model (interpret_community.mimic.models.LinearExplainableModel.interpret_community.common.policy, 19
      property), 47
model (interpret_community.mimic.models.SGDExplainableModel.interpret_community.common.serialization_utils,
      property), 50
model (interpret_community.mimic.models.tree_model.DecisionTreeExplainerModel.common.structured_model_explainer,
      property), 61
MODEL_CLASS (interpret_community.common.constants.ExplainType.interpret_community.dataset, 21
      attribute), 12
MODEL_ID (interpret_community.common.constants.ExplainParams 21
      attribute), 11
MODEL_STR (interpret_community.common.constants.LightGBMSentimentCommunity.explanation, 25
      attribute), 13
MODEL_TASK (interpret_community.common.constants.ExplainParams 25
      attribute), 11
MODEL_TASK (interpret_community.common.constants.ExplainType.interpret_community.lime.lime_explainer,
      attribute), 12
model_task (interpret_community.explanation.BaseExplainer.interpret_community.mimic, 38
      property), 26
MODEL_TYPE (interpret_community.common.constants.ExplainParams 63
      attribute), 11
model_type (interpret_community.explanation.explanation.BaseExplanation.interpret_community.mimic.models, 40
      property), 26
ModelSummary (class in interpret_community.common), 7
ModelSummary (class in interpret_community.common.model_summary), 17
ModelTask (class in interpret_community.common.constants), 14
module
    interpret_community, 3
    interpret_community.adapter, 5
    interpret_community.adapter.explanation_adapteinterpret_community.permutation, 66
        6
    interpret_community.common, 7
    interpret_community.common.aggregate, 7
    interpret_community.common.base_explainer, 8
    interpret_community.common.chained_identity, 9
    interpret_community.common.constants, 9
    interpret_community.common.error_handling, 16
    interpret_community.common.exception, 16
    interpret_community.common.explanation_utils, 16
        17
        18
        19
        20
        21
        22
        23
        24
        25
        26
        27
        28
        29
        30
        31
        32
        33
        34
        35
        36
        37
        38
        39
        40
        41
        42
        43
        44
        45
        46
        47
        48
        49
        50
        51
        52
        53
        54
        55
        56
        57
        58
        59
        60
        61
        62
        63
        64
        65
        66
        67
        68
        69
        70
        71
        72
        73
        74
        75
        76
        77
        78
        79
        80
        81
        82
        83
        84
        85
        86
        87
        88
        89
        90
        91
        92
        93
        94
        95
        96
        97
        98
        99
        100
        101
        102
        103
        104
        105
        106
        107
        108
        109
        110
        111
        112
        113
        114
        115
        116
        117
        118
        119
        120
        121
        122
        123
        124
        125
        126
        127
        128
        129
        130
        131
        132
        133
        134
        135
        136
        137
        138
        139
        140
        141
        142
        143
        144
        145
        146
        147
        148
        149
        150
        151
        152
        153
        154
        155
        156
        157
        158
        159
        160
        161
        162
        163
        164
        165
        166
        167
        168
        169
        170
        171
        172
        173
        174
        175
        176
        177
        178
        179
        180
        181
        182
        183
        184
        185
        186
        187
        188
        189
        190
        191
        192
        193
        194
        195
        196
        197
        198
        199
        200
        201
        202
        203
        204
        205
        206
        207
        208
        209
        210
        211
        212
        213
        214
        215
        216
        217
        218
        219
        220
        221
        222
        223
        224
        225
        226
        227
        228
        229
        230
        231
        232
        233
        234
        235
        236
        237
        238
        239
        240
        241
        242
        243
        244
        245
        246
        247
        248
        249
        250
        251
        252
        253
        254
        255
        256
        257
        258
        259
        260
        261
        262
        263
        264
        265
        266
        267
        268
        269
        270
        271
        272
        273
        274
        275
        276
        277
        278
        279
        280
        281
        282
        283
        284
        285
        286
        287
        288
        289
        290
        291
        292
        293
        294
        295
        296
        297
        298
        299
        300
        301
        302
        303
        304
        305
        306
        307
        308
        309
        310
        311
        312
        313
        314
        315
        316
        317
        318
        319
        320
        321
        322
        323
        324
        325
        326
        327
        328
        329
        330
        331
        332
        333
        334
        335
        336
        337
        338
        339
        340
        341
        342
        343
        344
        345
        346
        347
        348
        349
        350
        351
        352
        353
        354
        355
        356
        357
        358
        359
        360
        361
        362
        363
        364
        365
        366
        367
        368
        369
        370
        371
        372
        373
        374
        375
        376
        377
        378
        379
        380
        381
        382
        383
        384
        385
        386
        387
        388
        389
        390
        391
        392
        393
        394
        395
        396
        397
        398
        399
        400
        401
        402
        403
        404
        405
        406
        407
        408
        409
        410
        411
        412
        413
        414
        415
        416
        417
        418
        419
        420
        421
        422
        423
        424
        425
        426
        427
        428
        429
        430
        431
        432
        433
        434
        435
        436
        437
        438
        439
        440
        441
        442
        443
        444
        445
        446
        447
        448
        449
        450
        451
        452
        453
        454
        455
        456
        457
        458
        459
        460
        461
        462
        463
        464
        465
        466
        467
        468
        469
        470
        471
        472
        473
        474
        475
        476
        477
        478
        479
        480
        481
        482
        483
        484
        485
        486
        487
        488
        489
        490
        491
        492
        493
        494
        495
        496
        497
        498
        499
        500
        501
        502
        503
        504
        505
        506
        507
        508
        509
        510
        511
        512
        513
        514
        515
        516
        517
        518
        519
        520
        521
        522
        523
        524
        525
        526
        527
        528
        529
        530
        531
        532
        533
        534
        535
        536
        537
        538
        539
        540
        541
        542
        543
        544
        545
        546
        547
        548
        549
        550
        551
        552
        553
        554
        555
        556
        557
        558
        559
        5510
        5511
        5512
        5513
        5514
        5515
        5516
        5517
        5518
        5519
        5520
        5521
        5522
        5523
        5524
        5525
        5526
        5527
        5528
        5529
        5530
        5531
        5532
        5533
        5534
        5535
        5536
        5537
        5538
        5539
        5540
        5541
        5542
        5543
        5544
        5545
        5546
        5547
        5548
        5549
        5550
        5551
        5552
        5553
        5554
        5555
        5556
        5557
        5558
        5559
        55510
        55511
        55512
        55513
        55514
        55515
        55516
        55517
        55518
        55519
        55520
        55521
        55522
        55523
        55524
        55525
        55526
        55527
        55528
        55529
        55530
        55531
        55532
        55533
        55534
        55535
        55536
        55537
        55538
        55539
        55540
        55541
        55542
        55543
        55544
        55545
        55546
        55547
        55548
        55549
        55550
        55551
        55552
        55553
        55554
        55555
        55556
        55557
        55558
        55559
        55560
        55561
        55562
        55563
        55564
        55565
        55566
        55567
        55568
        55569
        55570
        55571
        55572
        55573
        55574
        55575
        55576
        55577
        55578
        55579
        55580
        55581
        55582
        55583
        55584
        55585
        55586
        55587
        55588
        55589
        55590
        55591
        55592
        55593
        55594
        55595
        55596
        55597
        55598
        55599
        555100
        555101
        555102
        555103
        555104
        555105
        555106
        555107
        555108
        555109
        555110
        555111
        555112
        555113
        555114
        555115
        555116
        555117
        555118
        555119
        555120
        555121
        555122
        555123
        555124
        555125
        555126
        555127
        555128
        555129
        555130
        555131
        555132
        555133
        555134
        555135
        555136
        555137
        555138
        555139
        555140
        555141
        555142
        555143
        555144
        555145
        555146
        555147
        555148
        555149
        555150
        555151
        555152
        555153
        555154
        555155
        555156
        555157
        555158
        555159
        555160
        555161
        555162
        555163
        555164
        555165
        555166
        555167
        555168
        555169
        555170
        555171
        555172
        555173
        555174
        555175
        555176
        555177
        555178
        555179
        555180
        555181
        555182
        555183
        555184
        555185
        555186
        555187
        555188
        555189
        555190
        555191
        555192
        555193
        555194
        555195
        555196
        555197
        555198
        555199
        555200
        555201
        555202
        555203
        555204
        555205
        555206
        555207
        555208
        555209
        555210
        555211
        555212
        555213
        555214
        555215
        555216
        555217
        555218
        555219
        555220
        555221
        555222
        555223
        555224
        555225
        555226
        555227
        555228
        555229
        555230
        555231
        555232
        555233
        555234
        555235
        555236
        555237
        555238
        555239
        555240
        555241
        555242
        555243
        555244
        555245
        555246
        555247
        555248
        555249
        555250
        555251
        555252
        555253
        555254
        555255
        555256
        555257
        555258
        555259
        555260
        555261
        555262
        555263
        555264
        555265
        555266
        555267
        555268
        555269
        555270
        555271
        555272
        555273
        555274
        555275
        555276
        555277
        555278
        555279
        555280
        555281
        555282
        555283
        555284
        555285
        555286
        555287
        555288
        555289
        555290
        555291
        555292
        555293
        555294
        555295
        555296
        555297
        555298
        555299
        555300
        555301
        555302
        555303
        555304
        555305
        555306
        555307
        555308
        555309
        555310
        555311
        555312
        555313
        555314
        555315
        555316
        555317
        555318
        555319
        555320
        555321
        555322
        555323
        555324
        555325
        555326
        555327
        555328
        555329
        555330
        555331
        555332
        555333
        555334
        555335
        555336
        555337
        555338
        555339
        555340
        555341
        555342
        555343
        555344
        555345
        555346
        555347
        555348
        555349
        555350
        555351
        555352
        555353
        555354
        555355
        555356
        555357
        555358
        555359
        555360
        555361
        555362
        555363
        555364
        555365
        555366
        555367
        555368
        555369
        555370
        555371
        555372
        555373
        555374
        555375
        555376
        555377
        555378
        555379
        555380
        555381
        555382
        555383
        555384
        555385
        555386
        555387
        555388
        555389
        555390
        555391
        555392
        555393
        555394
        555395
        555396
        555397
        555398
        555399
        555400
        555401
        555402
        555403
        555404
        555405
        555406
        555407
        555408
        555409
        555410
        555411
        555412
        555413
        555414
        555415
        555416
        555417
        555418
        555419
        555420
        555421
        555422
        555423
        555424
        555425
        555426
        555427
        555428
        555429
        555430
        555431
        555432
        555433
        555434
        555435
        555436
        555437
        555438
        555439
        555440
        555441
        555442
        555443
        555444
        555445
        555446
        555447
        555448
        555449
        555450
        555451
        555452
        555453
        555454
        555455
        555456
        555457
        555458
        555459
        555460
        555461
        555462
        555463
        555464
        555465
        555466
        555467
        555468
        555469
        555470
        555471
        555472
        555473
        555474
        555475
        555476
        555477
        555478
        555479
        555480
        555481
        555482
        555483
        555484
        555485
        555486
        555487
        555488
        555489
        555490
        555491
        555492
        555493
        555494
        555495
        555496
        555497
        555498
        555499
        555500
        555501
        555502
        555503
        555504
        555505
        555506
        555507
        555508
        555509
        555510
        555511
        555512
        555513
        555514
        555515
        555516
        555517
        555518
        555519
        555520
        555521
        555522
        555523
        555524
        555525
        555526
        555527
        555528
        555529
        555530
        555531
        555532
        555533
        555534
        555535
        555536
        555537
        555538
        555539
        555540
        555541
        555542
        555543
        555544
        555545
        555546
        555547
        555548
        555549
        555550
        555551
        555552
        555553
        555554
        555555
        555556
        555557
        555558
        555559
        555560
        555561
        555562
        555563
        555564
        555565
        555566
        555567
        555568
        555569
        555570
        555571
        555572
        555573
        555574
        555575
        555576
        555577
        555578
        555579
        555580
        555581
        555582
        555583
        555584
        555585
        555586
        555587
        555588
        555589
        555590
        555591
        555592
        555593
        555594
        555595
        555596
        555597
        555598
        555599
        5555100
        5555101
        5555102
        5555103
        5555104
        5555105
        5555106
        5555107
        5555108
        5555109
        5555110
        5555111
        5555112
        5555113
        5555114
        5555115
        5555116
        5555117
        5555118
        5555119
        55551110
        55551111
        55551112
        55551113
        55551114
        55551115
        55551116
        55551117
        55551118
        55551119
        555511110
        555511111
        555511112
        555511113
        555511114
        555511115
        555511116
        555511117
        555511118
        555511119
        5555111110
        5555111111
        5555111112
        5555111113
        5555111114
        5555111115
        5555111116
        5555111117
        5555111118
        5555111119
        55551111110
        55551111111
        55551111112
        55551111113
        55551111114
        55551111115
        55551111116
        55551111117
        55551111118
        55551111119
        555511111110
        555511111111
        555511111112
        555511111113
        555511111114
        555511111115
        555511111116
        555511111117
        555511111118
        555511111119
        5555111111110
        5555111111111
        5555111111112
        5555111111113
        5555111111114
        5555111111115
        5555111111116
        5555111111117
        5555111111118
        5555111111119
        55551111111110
        55551111111111
        55551111111112
        55551111111113
        55551111111114
        55551111111115
        55551111111116
        55551111111117
        55551111111118
        55551111111119
        555511111111110
        555511111111111
        555511111111112
        555511111111113
        555511111111114
        555511111111115
        555511111111116
        555511111111117
        555511111111118
        555511111111119
        5555111111111110
        5555111111111111
        5555111111111112
        5555111111111113
        5555111111111114
        5555111111111115
        5555111111111116
        5555111111111117
        5555111111111118
        5555111111111119
        55551111111111110
        55551111111111111
        55551111111111112
        55551111111111113
        55551111111111114
        55551111111111115
        55551111111111116
        55551111111111117
        55551111111111118
        55551111111111119
        555511111111111110
        555511111111111111
        555511111111111112
        555511111111111113
        555511111111111114
        555511111111111115
        555511111111111116
        555511111111111117
        555511111111111118
        555511111111111119
        5555111111111111110
        5555111111111111111
        5555111111111111112
        55551111111
```

	ORIGINAL_EVAL_EXAMPLES	(inter-
interpret_community.tabular_explainer, 95	pret_community.common.constants.MimicSerializationConstants	attribute), 14
interpret_community.version, 97	OVERALL (interpret_community.common.constants.InterpretData	
interpret_community.widget, 93	attribute), 13	
interpret_community.widget.explanation_dashboard, 94		
MULTICLASS (interpret_community.common.constants.InterpretData attribute), 13	PER_CLASS_NAMES	(inter-
MULTICLASS (interpret_community.common.constants.LightGBMSerializationConstants attribute), 13	pret_community.common.constants.ExplainParams attribute), 11	
N	PER_CLASS_RANK	(inter-
name (interpret_community.explanation.explanation.BaseExplanation attribute), 11	pret_community.common.constants.ExplainParams	
property), 26	per_class_rank	(inter-
NAMES (interpret_community.common.constants.InterpretData attribute), 13	PER_CLASS_VALUES	
ndcg() (in module pret_community.common.metrics), 17	pret_community.explanation.explanation.PerClassMixin property), 32	
NER (interpret_community.common.constants.Spacy attribute), 15	PER_CLASS_VALUES	(inter-
nonify_properties (interpret_community.common.constants.LightGBMSerializationConstants attribute), 14	pret_community.common.constants.ExplainParams attribute), 11	
nonify_properties (interpret_community.common.constants.MimicSerializationConstants attribute), 14	per_class_values	(inter-
nonify_properties (interpret_community.common.constants.ExplainParams attribute), 14	PerClassMixin (class in interpret_community.explanation.explanation),	
NUM_CLASSES (interpret_community.common.constants.ExplainParam attribute), 13	PERF (interpret_community.common.constants.InterpretData attribute), 11	
num_classes (interpret_community.explanation.explanation.ClassesAttribute), 12	PFI (interpret_community.common.constants.ExplainType attribute), 69	
NUM_EXAMPLES (interpret_community.common.constants.ExplainParam attribute), 11	PFIExplainer (class in interpret_community.permutation), 66	
num_examples (interpret_community.explanation.explanation.LocalExplanation.permutation.permutation_importance), 31	PFIExplainer (class in interpret_community.permutation), 69	
NUM_FEATURES (interpret_community.common.constants.ExplainParam attribute), 11	PFIExplainer (class in interpret_community.permutation.metric_constants.MetricConstants attribute), 18	
num_features (interpret_community.dataset.dataset_wrapper.DatasetWrapper), 68	PREDICTION_SCORE predict() (interpret_community.common.model_wrapper.WrappedClassificationMethod), 18	
num_features (interpret_community.explanation.explanation.FeatureImportanceExplanations attribute), 27	predict() (interpret_community.common.model_wrapper.WrappedClassificationMethod), 18	
O	predict() (interpret_community.common.model_wrapper.WrappedPytorchModel), 18	
OBJECTIVE (interpret_community.common.constants.LightGBMSerializationConstants attribute), 13	predict() (interpret_community.common.model_wrapper.WrappedRegressionModel), 19	
one_hot_encode()	(interpret_community.dataset.dataset_wrapper.DatasetWrapper), 23	
original_dataset	(interpret_community.dataset.dataset_wrapper.DatasetWrapper), 23	
original_dataset_with_type	(interpret_community.dataset.dataset_wrapper.DatasetWrapper), 23	

`predict()` (*interpret_community.mimic.models.lightgbm_explainable_model*)
(interpret-community.mimic.models.tree_model.DecisionTreeExplainer method), 53
`predict()` (*interpret_community.mimic.models.linear_model.Explainer*)
(interpret-community.mimic.models.linear_model.LinearExplainer method), 55
`PREDICT_PROBA_FLAG` (*interpret-community.mimic.models.linear_model.SGDExplainer*)
(interpret-community.mimic.models.linear_model.SGDExplainer method), 58
`PREDICTIONS` (*interpret_community.common.constants.SKLearn attribute*), 14
`PREDICTIONS` (*interpret_community.mimic.models.LinearExplainer*)
(interpret-community.mimic.models.LinearExplainer method), 47
`PROBABILITIES` (*interpret-community.mimic.models.SGDExplainer*)
(interpret-community.mimic.models.SGDExplainer method), 50
`PROBABILITY` (*interpret_community.mimic.models.tree_model.DecisionTreeExplainer*)
(interpret-community.mimic.models.tree_model.DecisionTreeExplainer method), 61
`PROBABILITY` (*interpret-community.common.constants.ExplainParams attribute*), 15
`PYTORCH` (*interpret_community.common.constants.DNNFramework attribute*), 9
`PYTHON_EXPLAINER` (*interpret_community.common.model_wrapper.WrappedPytorchModelExplainer class* in *interpret_community.common.structured_model_explainer*), 21
`PYTHON_EXPLAINER` (*interpret_community.common.model_wrapper.WrappedClassificationModel*)
(interpret-community.common.model_wrapper.WrappedClassificationModel method), 18
`R2_SCORE` (*interpret_community.permutation.metric_constants.MetricConstants attribute*), 68
`RECALL_SCORE` (*interpret_community.common.model_wrapper.WrappedClassificationModel*)
(interpret-community.common.model_wrapper.WrappedClassificationModel method), 18
`REGRESSION` (*interpret_community.common.constants.ExplainType attribute*), 12
`REGRESSION` (*interpret_community.common.constants.LightGBMSerialization attribute*), 13
`Regression` (*interpret_community.common.constants.ModelTask attribute*), 14
`Reset` (*interpret_community.common.constants.ResetIndex attribute*), 14
`RESET_INDEX` (*interpret_community.common.constants.MimicSerialization attribute*), 14
`RESET_INDEX` (*interpret_community.mimic.models.explainable_model.BaseExplainableModel*)
(interpret-community.mimic.models.explainable_model.BaseExplainableModel method), 51
`pret_community.dataset.dataset_wrapper.DatasetWrapper` (*interpret-community.mimic.models.explainable_model.ResetIndex class* in *interpret-community.common.constants*), 14
`ResetTeacher` (*interpret_community.common.constants.ResetIndex attribute*), 14
`S`
`Sample` (*interpret_community.dataset.dataset_wrapper.DatasetWrapper method*), 23
`sampling_method` (*interpret-community.mimic.models.linear_model.LinearExplainer*)
(interpret-community.mimic.models.linear_model.LinearExplainer method), 56
`SamplingPolicy` (*interpret-community.common.policy.SamplingPolicy property*), 20
`SAMPLING_POLICY` (*interpret-community.mimic.models.LinearExplainer*)
(interpret-community.mimic.models.LinearExplainer method), 48
`SamplingPolicy` (*interpret-community.mimic.models.SGDExplainer*)
(interpret-community.mimic.models.SGDExplainer method), 50
`pret_community.common.constants.ExplainParams attribute), 11
pret_community.common.policy), 19`

save_explanation() (in module `interpret_community.explanation.explanation`), 33
save_properties (in module `interpret_community.common.constants.LightGBMSerializationIndexer`), 14
save_properties (in module `interpret_community.common.constants.MimicSerializationIndexer`), 14
ScenarioNotSupportedException, 16
Scipy (class in `interpret_community.common.constants`), 15
SCORES (`interpret_community.common.constants.InterpretData` attribute), 13
selector (`interpret_community.explanation.explanation.BaseExplanation` property), 26
selector (`interpret_community.explanation.explanation.GlobalExplanation` property), 29
selector (`interpret_community.explanation.explanation.LocalExplanation` property), 31
set_index() (`interpret_community.dataset.dataset_wrapper.DatasetWrapper` method), 24
SGDExplainableModel (class in `interpret_community.mimic.models`), 48
SGDExplainableModel (class in `interpret_community.mimic.models.linear_model`), 56
SHAP (`interpret_community.common.constants.ExplainType` attribute), 12
SHAP_DEEP (`interpret_community.common.constants.ExplainType` attribute), 12
SHAP_GPU_KERNEL (in module `interpret_community.common.constants.ExplainType` attribute), 12
SHAP_KERNEL (`interpret_community.common.constants.ExplainType` attribute), 12
SHAP_LINEAR (`interpret_community.common.constants.ExplainType` attribute), 12
SHAP_TREE (`interpret_community.common.constants.ExplainType` attribute), 12
shap_values() (in module `interpret_community.mimic.models.linear_model.LinearExplainer` method), 56
SHAP_VALUES_OUTPUT (in module `interpret_community.common.constants.ExplainParam` attribute), 11
SHAPDefaults (class in `interpret_community.common.constants`), 14
ShapValuesOutput (class in `interpret_community.common.constants`), 15
SINGLE (`interpret_community.common.constants.InterpretData` attribute), 13
SKLearn (class in `interpret_community.common.constants`), 15

Spacy (class in `interpret_community.common.constants`), 15
SPECIFIC (`interpret_community.common.constants.InterpretData` attribute), 13
STRUCTURED_MODEL_EXPLAINER (class in `interpret_community.common.structured_model_explainer`), 21
summary_dataset (`interpret_community.dataset.dataset_wrapper.DatasetWrapper` property), 24

TABULAR (`interpret_community.common.constants.ExplainType` attribute), 12
tabular_decorator() (in module `interpret_community.dataset.decorator`), 25
TabularExplainer (class in `interpret_community.tabular_explainer`), 3
TabularExplainer (class in `interpret_community.tabular_explainer`), 95
TAGGER (`interpret_community.common.constants.Spacy` attribute), 15
take_subset() (in module `interpret_community.dataset.dataset_wrapper.DatasetWrapper` method), 24

TEACHER_PROBABILITY (`interpret_community.common.constants.ShapValuesOutput` attribute), 15
Tensorflow (class in `interpret_community.common.constants`), 15
TENSORFLOW (`interpret_community.common.constants.DNNFramework` attribute), 9
Tensorflow (`interpret_community.common.constants.Tensorflow` attribute), 15
TIMESTAMP_FEATURIZER (`interpret_community.common.constants.MimicSerializationConstants` attribute), 14
timestamp_featurizer() (in module `interpret_community.dataset.dataset_wrapper.DatasetWrapper` method), 24
transform() (`interpret_community.dataset.dataset_wrapper.CustomTimestampExplainer` method), 22

TREE_EXPLAINABLE_MODEL_TYPE (`interpret_community.common.constants.ExplainableModelType` attribute), 12
TREE_EXPLAINER (class in `interpret_community.common.constants.LightGBMSerializationConstants` attribute), 13
TreeExplainer (class in `interpret_community.shap.tree_explainer`), 91

TYPE (*interpret_community.common.constants.InterpretData attribute*), 13
typed_dataset (*interpret_community.dataset.dataset_wrapper.DatasetWrapper property*), 24
typed_wrapper_func() (*interpret_community.dataset.dataset_wrapper.DatasetWrapper method*), 24

U

UNIVARIATE (*interpret_community.common.constants.InterpretData attribute*), 13
Unknown (*interpret_community.common.constants.ModelTask attribute*), 14

V

VALUE (*interpret_community.common.constants.InterpretData attribute*), 13
VALUES (*interpret_community.common.constants.InterpretData attribute*), 13
visualize() (*interpret_community.explanation.explanation.BaseExplanation method*), 26

W

wrap_dataset() (*in module interpret_community.dataset.decorator*), 25
wrap_model() (*in module interpret_community.common.model_wrapper*), 19
WrappedClassificationModel (*class in interpret_community.common.model_wrapper*), 18
WrappedClassificationWithoutProbaModel (*class in interpret_community.common.model_wrapper*), 18
WrappedPytorchModel (*class in interpret_community.common.model_wrapper*), 18
WrappedRegressionModel (*class in interpret_community.common.model_wrapper*), 19
write() (*interpret_community.shap.deep_explainer.logger_redirector method*), 83