interpret-community

Release 0.15.2

Oct 08, 2020

Contents

The code is available from GitHub.

CHAPTER 1

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 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: 'unknown'>*, ***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** (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** (*model that implements sklearn.predict() or sklearn.predict_proba() or pipeline function that accepts a 2d ndarray*) – The model or pipeline to explain.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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.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: *interpret_community.common.chained_identity.ChainedIdentity*

The base class for explainers that create global explanations.

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: *interpret_community.common.chained_identity.ChainedIdentity*

The base class for explainers that create local explanations.

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.Unk 'un- known'>*, ***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** (*model that implements sklearn.predict or sklearn. predict_proba or function that accepts a 2d ndarray*) – The model to explain or function if is_function is True.
- **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: 'un- known'>*, ***kwargs*)

Bases: *interpret_community.common.chained_identity.ChainedIdentity*

Mixin for black box models or functions.

Parameters

- **model** (*model that implements sklearn.predict or sklearn. predict_proba or function that accepts a 2d ndarray*) – The model to explain or function if is_function is True.
- **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_serializable()

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

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_KERNEL = 'shap_kernel'

SHAP_LINEAR = 'shap_linear'

SHAP_TREE = 'shap_tree'

TABULAR = 'tabular'

class interpret_community.common.constants.**ExplainableModelType**
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', '_orig

save_properties = ['surrogate_model']

class interpret_community.common.constants.**ModelTask**
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**
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.**ShapValuesOutput**
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.explanation_utils module

Defines helpful utilities for summarizing and uploading data.

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 rewrapping the model to conform to explainer contracts.

class interpret_community.common.model_wrapper.**WrappedClassificationModel**(*model*, *eval_function*)
Bases: object

A class for wrapping a classification model.

predict(*dataset*)
Predict the output using the wrapped classification model.

Parameters **dataset** (DatasetWrapper) – The dataset to predict on.

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

Parameters **dataset** (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 trival boolean version of predict_proba

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

Parameters **dataset** (DatasetWrapper) – The dataset to predict on.

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

Parameters **dataset** (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** (DatasetWrapper) – The dataset to predict on.

predict_classes(*dataset*)
Predict the class using the wrapped PyTorch model.

Parameters **dataset** (DatasetWrapper) – The dataset to predict on.

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

Parameters **dataset** (DatasetWrapper) – The dataset to predict_proba on.

class interpret_community.common.model_wrapper.**WrappedRegressionModel**(*model*, *eval_function*)

Bases: object

A class for wrapping a regression model.

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

Parameters **dataset** (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** (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

allow_eval_sampling
Get whether to allow sampling of evaluation data.

Returns Whether to allow sampling of evaluation data.

Return type bool

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

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.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**(*m*
**

Bases: *interpret_community.common.base_explainer.BaseExplainer*

The base PureStructuredModelExplainer API for explainers used on specific models.

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

class interpret_community.common.structured_model_explainer.**StructuredInitModelExplainer**(*m*
in
ti
iz
ti
**

Bases: *interpret_community.common.base_explainer.BaseExplainer*

The base StructuredInitModelExplainer API for explainers.

Used on specific models that require initialization examples.

Parameters

- **model** (*A white box model.*) – The white box model to explain.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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 iml.datatypes.DenseData 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 iml.datatypes.DenseData or scipy.sparse.csr_matrix*) – The dataset containing timestamp columns to featurize.

Returns The transformed dataset.

Return type numpy.array or iml.datatypes.DenseData or scipy.sparse.csr_matrix

class interpret_community.dataset.dataset_wrapper.**DatasetWrapper**(*dataset*)
Bases: object

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

Parameters **dataset** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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** (*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** (*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*, ***kwargs*)
Summarizes the dataset if it hasn't been summarized yet.

dataset
Get the dataset.

Returns The underlying dataset.

Return type numpy.array or iml.datatypes.DenseData 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

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 OneHotEncoder

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 iml.datatypes.DenseData or scipy.sparse matrix

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 iml.datatypes.DenseData 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 ColumnTransformer

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 iml.datatypes.DenseData 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 *DatasetWrapper*

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 iml.datatypes.DenseData 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 function that outputs the original type

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.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: *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

id
Get the explanation ID.

Returns The explanation ID.

Return type str

method
Get the explanation method.

Returns The explanation method.

Return type str

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

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

name
Get the name of the explanation.

Returns The name of the explanation.

Return type str

selector
Get the local or global selector.

Returns The selector as a pandas dataframe of records.

Return type pd.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.

classes
Get the classes.

Returns The list of classes.

Return type list

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** (*np.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

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=N*, *is_raw=False*, *is_engineered=F*, ***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.

features

Get the feature names.

Returns The feature names.

Return type list[str]

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

is_raw

Get the raw explanation flag.

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

Return type bool

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{str: float}

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 given input feature maps.

Parameters

- **feature_maps** (*list of numpy arrays or sparse matrices 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.*) – list of feature maps from raw to generated feature
- **raw_feature_names** (*[str]*) – list of raw feature names
- **eval_data** (*np.ndarray or pd.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]]

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]

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]

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 pd.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 of numpy arrays or sparse matrices 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) – list of feature maps from raw to generated feature

- **raw_feature_names** (*[str]*) – list of raw feature names
- **eval_data** (*np.ndarray or pd.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]]

is_local_sparse

Determines whether the local importance values are sparse.

Returns True if the local importance values are sparse.

Return type bool

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]

num_examples

Get the number of examples on the explanation.

Returns The number of examples on the explanation.

Return type int

selector

Get the local selector.

Returns The selector as a pandas dataframe of records.

Return type pd.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]]

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

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

Returns A model explanation object containing the global explanation.

Return type *GlobalExplanation*

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

Parameters

- **evaluation_examples** (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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The model to explain or function if is_function is True.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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: '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** (SamplingPolicy) – Optional policy for sampling the evaluation examples. See documentation on SamplingPolicy for more information.
- **include_local** (*bool*) – Include the local explanations in the returned global explanation. If include_local is False, will stream the local explanations to aggregate to global.
- **batch_size** (*int*) – If include_local is False, specifies the batch size for aggregating local explanations to global.

Returns A model explanation object containing the global explanation.

Return type *GlobalExplanation*

explain_local(*evaluation_examples*)

Explain the function locally by using LIME.

Parameters

- **evaluation_examples** (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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The model to explain or function if is_function is True.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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: 'default'>*, *categorical_features=None*, *model_task=<ModelTask.Unknown: 'unknown'>*, *reset_index=<ResetIndex.Ignore: '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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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: *interpret_community.common.chained_identity.ChainedIdentity*

The base class for models that can be explained.

expected_values
Abstract property to get the expected values.

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

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

static explainable_model_type(*self*)
Retrieve the model type.

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

model
Abstract property to get the underlying model.

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

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

class interpret_community.mimic.models.**LGBMExplainableModel**(*multiclass=False*, *random_state=123*, *shap_values_output=<ShapValuesOutput.DE 'default'>*, *classification=True*, ***kwargs*)

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

available_explanations = ['global', 'local']

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 or scipy array*) – 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 (*self*)
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.

param dataset The dataset to train the model on.

type dataset numpy or scipy array

param labels The labels to train the model on.

type labels numpy or scipy array

If multiclass=True, uses the parameters for LGBMClassifier: Build a gradient boosting model from the training set (X, y).

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input feature matrix.

y [arraylike of shape = [n_samples]] The target values (class labels in classification, real numbers in regression).

sample_weight [arraylike of shape = [n_samples] or None, optional (default=None)] Weights of training data.

init_score [arraylike of shape = [n_samples] or None, optional (default=None)] Init score of training data.

eval_set [list or None, optional (default=None)] A list of (X, y) tuple pairs to use as validation sets.

eval_names [list of strings or None, optional (default=None)] Names of eval_set.

eval_sample_weight [list of arrays or None, optional (default=None)] Weights of eval data.

eval_class_weight [list or None, optional (default=None)] Class weights of eval data.

eval_init_score [list of arrays or None, optional (default=None)] Init score of eval data.

eval_metric [string, list of strings, callable or None, optional (default=None)] If string, it should be a builtin evaluation metric to use. If callable, it should be a custom evaluation metric, see note below for more details. In either case, the `metric` from the model parameters will be evaluated and used as well. Default: 'l2' for LGBMRegressor, 'logloss' for LGBMClassifier, 'ndcg' for LGBMRanker.

early_stopping_rounds [int or None, optional (default=None)] Activates early stopping. The model will train until the validation score stops improving. Validation score needs to improve at least every `early_stopping_rounds` round(s) to continue training. Requires at least one validation data and one metric. If there's more than one, will check all of them. But the training data is ignored anyway. To check only the first metric, set the `first_metric_only` parameter to `True` in additional parameters `kwargs` of the model constructor.

verbose [bool or int, optional (default=True)] Requires at least one evaluation data. If True, the eval metric on the eval set is printed at each boosting stage. If int, the eval metric on the eval set is printed at every `verbose` boosting stage. The last boosting stage or the boosting stage found by using `early_stopping_rounds` is also printed.

Example

With `verbose` = 4 and at least one item in `eval_set`, an evaluation metric is printed every 4 (instead of 1) boosting stages.

feature_name [list of strings or 'auto', optional (default='auto')] Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.

categorical_feature [list of strings or int, or 'auto', optional (default='auto')] Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas unordered categorical columns are used. All values in categorical features should be less than int32 max value (2147483647). Large values could be memory consuming. Consider using consecutive integers starting from zero. All negative values in categorical features will be treated as missing values. The output cannot be monotonically constrained with respect to a categorical feature.

callbacks [list of callback functions or None, optional (default=None)] List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.

init_model [string, Booster, LGBMModel or None, optional (default=None)] Filename of LightGBM model, Booster instance or LGBMModel instance used for continue training.

Returns

self [object] Returns self.

Note

Custom eval function expects a callable with following signatures: `func(y_true, y_pred)`, `func(y_true, y_pred, weight)` or `func(y_true, y_pred, weight, group)` and returns (eval_name, eval_result, is_higher_better) or list of (eval_name, eval_result, is_higher_better):

y_true [arraylike of shape = [n_samples]] The target values.

y_pred [arraylike of shape = [n_samples] or shape = [n_samples * n_classes] (for multiclass task)] The predicted values.

weight [arraylike of shape = [n_samples]] The weight of samples.

group [arraylike] Group/query data, used for ranking task.

eval_name [string] The name of evaluation function (without whitespaces).

eval_result [float] The eval result.

is_higher_better [bool] Is eval result higher better, e.g. AUC is `is_higher_better`.

For binary task, the y_pred is probability of positive class (or margin in case of custom `objective`). For multiclass task, the y_pred is group by class_id first, then group by row_id. If you want to get ith row y_pred in jth class, the access way is y_pred[j * num_data + i].

Otherwise, if multiclass=False, uses the parameters for LGBMRegressor: Build a gradient boosting model from the training set (X, y).

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input feature matrix.

y [arraylike of shape = [n_samples]] The target values (class labels in classification, real numbers in regression).

sample_weight [arraylike of shape = [n_samples] or None, optional (default=None)] Weights of training data.

init_score [arraylike of shape = [n_samples] or None, optional (default=None)] Init score of training data.

eval_set [list or None, optional (default=None)] A list of (X, y) tuple pairs to use as validation sets.

eval_names [list of strings or None, optional (default=None)] Names of eval_set.

eval_sample_weight [list of arrays or None, optional (default=None)] Weights of eval data.

eval_init_score [list of arrays or None, optional (default=None)] Init score of eval data.

eval_metric [string, list of strings, callable or None, optional (default=None)] If string, it should be a builtin evaluation metric to use. If callable, it should be a custom evaluation metric, see note below for more details. In either case, the `metric` from the model parameters will be evaluated and used as well. Default: 'l2' for LGBMRegressor, 'logloss' for LGBMClassifier, 'ndcg' for LGBMRanker.

early_stopping_rounds [int or None, optional (default=None)] Activates early stopping. The model will train until the validation score stops improving. Validation score needs to improve at least every `early_stopping_rounds` round(s) to continue training. Requires at least one validation data and one metric. If there's more than one, will check all of them. But the training data is ignored anyway. To check only the first metric, set the

first_metric_only parameter to True in additional parameters kwargs of the model constructor.

verbose [bool or int, optional (default=True)] Requires at least one evaluation data. If True, the eval metric on the eval set is printed at each boosting stage. If int, the eval metric on the eval set is printed at every verbose boosting stage. The last boosting stage or the boosting stage found by using early_stopping_rounds is also printed.

Example

With verbose = 4 and at least one item in eval_set, an evaluation metric is printed every 4 (instead of 1) boosting stages.

feature_name [list of strings or 'auto', optional (default='auto')] Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.

categorical_feature [list of strings or int, or 'auto', optional (default='auto')] Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify feature_name as well). If 'auto' and data is pandas DataFrame, pandas unordered categorical columns are used. All values in categorical features should be less than int32 max value (2147483647). Large values could be memory consuming. Consider using consecutive integers starting from zero. All negative values in categorical features will be treated as missing values. The output cannot be monotonically constrained with respect to a categorical feature.

callbacks [list of callback functions or None, optional (default=None)] List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.

init_model [string, Booster, LGBMModel or None, optional (default=None)] Filename of LightGBM model, Booster instance or LGBMModel instance used for continue training.

Returns

self [object] Returns self.

Note

Custom eval function expects a callable with following signatures: func(y_true, y_pred), func(y_true, y_pred, weight) or func(y_true, y_pred, weight, group) and returns (eval_name, eval_result, is_higher_better) or list of (eval_name, eval_result, is_higher_better):

y_true [arraylike of shape = [n_samples]] The target values.

y_pred [arraylike of shape = [n_samples] or shape = [n_samples * n_classes] (for multiclass task)] The predicted values.

weight [arraylike of shape = [n_samples]] The weight of samples.

group [arraylike] Group/query data, used for ranking task.

eval_name [string] The name of evaluation function (without whitespaces).

eval_result [float] The eval result.

is_higher_better [bool] Is eval result higher better, e.g. AUC is is_higher_better.

For binary task, the y_pred is probability of positive class (or margin in case of custom objective). For multiclass task, the y_pred is group by class_id first, then group by row_id. If you want to get ith row y_pred in jth class, the access way is y_pred[j * num_data + i].

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.

param dataset The dataset to predict on.

type dataset numpy or scipy array

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LGBMClassifier: Return the predicted value for each sample.

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input features matrix.

raw_score [bool, optional (default=False)] Whether to predict raw scores.

start_iteration [int, optional (default=0)] Start index of the iteration to predict. If <= 0, starts from the first iteration.

num_iteration [int or None, optional (default=None)] Total number of iterations used in the prediction. If None, if the best iteration exists and start_iteration <= 0, the best iteration is used; otherwise, all iterations from `start_iteration` are used (no limits). If <= 0, all iterations from `start_iteration` are used (no limits).

pred_leaf [bool, optional (default=False)] Whether to predict leaf index.

pred_contrib [bool, optional (default=False)] Whether to predict feature contributions.

Note: If you want to get more explanations for your model's predictions using SHAP values, like SHAP interaction values, you can install the shap package (https://github.com/slundberg/shap). Note that unlike the shap package, with `pred_contrib` we return a matrix with an extra column, where the last column is the expected value.

---

kwargs Other parameters for the prediction.

Returns

predicted_result [arraylike of shape = [n_samples] or shape = [n_samples, n_classes]] The predicted values.

X_leaves [arraylike of shape = [n_samples, n_trees] or shape = [n_samples, n_trees * n_classes]] If `pred_leaf=True`, the predicted leaf of every tree for each sample.

X_SHAP_values [arraylike of shape = [n_samples, n_features + 1] or shape = [n_samples, (n_features + 1) * n_classes] or list with n_classes length of such objects] If `pred_contrib=True`, the feature contributions for each sample.

Otherwise, if multiclass=False, uses the parameters for LGBMRegressor: Return the predicted value for each sample.

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input features matrix.

raw_score [bool, optional (default=False)] Whether to predict raw scores.

start_iteration [int, optional (default=0)] Start index of the iteration to predict. If <= 0, starts from the first iteration.

num_iteration [int or None, optional (default=None)] Total number of iterations used in the prediction. If None, if the best iteration exists and start_iteration <= 0, the best iteration is used; otherwise, all iterations from `start_iteration` are used (no limits). If <= 0, all iterations from `start_iteration` are used (no limits).

pred_leaf [bool, optional (default=False)] Whether to predict leaf index.

pred_contrib [bool, optional (default=False)] Whether to predict feature contributions.

Note: If you want to get more explanations for your model's predictions using SHAP values, like SHAP interaction values, you can install the shap package (https://github.com/slundberg/shap). Note that unlike the shap package, with `pred_contrib` we return a matrix with an extra column, where the last column is the expected value.

---

kwargs Other parameters for the prediction.

Returns

predicted_result [arraylike of shape = [n_samples] or shape = [n_samples, n_classes]] The predicted values.

X_leaves [arraylike of shape = [n_samples, n_trees] or shape = [n_samples, n_trees * n_classes]] If `pred_leaf=True`, the predicted leaf of every tree for each sample.

X_SHAP_values [arraylike of shape = [n_samples, n_features + 1] or shape = [n_samples, (n_features + 1) * n_classes] or list with n_classes length of such objects] If `pred_contrib=True`, the feature contributions for each sample.

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

Call lightgbm predict_proba to predict probabilities using the explainable model.

param dataset The dataset to predict probabilities on.

type dataset numpy or scipy array

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LGBMClassifier: Return the predicted probability for each class for each sample.

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input features matrix.

raw_score [bool, optional (default=False)] Whether to predict raw scores.

start_iteration [int, optional (default=0)] Start index of the iteration to predict. If <= 0, starts from the first iteration.

num_iteration [int or None, optional (default=None)] Total number of iterations used in the prediction. If None, if the best iteration exists and start_iteration <= 0, the best iteration is used; otherwise, all iterations from `start_iteration` are used (no limits). If <= 0, all iterations from `start_iteration` are used (no limits).

pred_leaf [bool, optional (default=False)] Whether to predict leaf index.

pred_contrib [bool, optional (default=False)] Whether to predict feature contributions.

> **Note:** If you want to get more explanations for your model's predictions using SHAP values, like SHAP interaction values, you can install the shap package (https://github.com/slundberg/shap). Note that unlike the shap package, with `pred_contrib` we return a matrix with an extra column, where the last column is the expected value.

kwargs Other parameters for the prediction.

Returns

predicted_probability [arraylike of shape = [n_samples, n_classes]] The predicted probability for each class for each sample.

X_leaves [arraylike of shape = [n_samples, n_trees * n_classes]] If `pred_leaf=True`, the predicted leaf of every tree for each sample.

X_SHAP_values [arraylike of shape = [n_samples, (n_features + 1) * n_classes] or list with n_classes length of such objects] If `pred_contrib=True`, the feature contributions for each sample.

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']

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 or scipy array*) – 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 or scipy array

param labels The labels to train the model on.

type labels numpy or scipy 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 : 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 : returns an instance of self.

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 or scipy array

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

Returns

C [array, shape [n_samples]] Predicted class label per 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)

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 or scipy array

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, http://www.research.ibm.com/people/z/zadrozny/kdd2002Transf.pdf

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.

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']

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 or scipy array*) – 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(*self*)

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 or scipy array

param labels The labels to train the model on.

type labels numpy or scipy 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 : returns an instance of self.

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 or scipy array

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

Returns

C [array, shape [n_samples]] Predicted class label per sample.

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

Predict using the linear model.

Parameters

X [array_like 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 or scipy array

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.**DecisionTreeExplainableModel**(*multiclass=False*, *random_state=123*, *shap_values_output=<ShapValue 'default'>*, *classification=True*, ***kwargs*)

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

available_explanations = ['global', 'local']

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 or scipy array*) – 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 (*self*)
Retrieve the model type.

Returns Tree explainable model type.

Return type *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 or scipy array

param labels The labels to train the model on.

type labels numpy or scipy 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 [arraylike of shape (n_samples, n_features), default=None] The indexes of the sorted training input samples. If many tree are grown on the same dataset, this allows the ordering to be cached between trees. If None, the data will be sorted here. Don't use this parameter unless you know what to do.

Returns

self [DecisionTreeClassifier] Fitted estimator.

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

Parameters

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

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

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

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

X_idx_sorted [arraylike of shape (n_samples, n_features), default=None] The indexes of the sorted training input samples. If many tree are grown on the same dataset, this allows the ordering to be cached between trees. If None, the data will be sorted here. Don't use this parameter unless you know what to do.

Returns

self [DecisionTreeRegressor] Fitted estimator.

`model`

Retrieve the underlying model.

Returns The decision tree model, either classifier or regressor.

Return type Union[DecisionTreeClassifier, DecisionTreeRegressor]

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

Call tree predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy or scipy array

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 or scipy array

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.

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: *interpret_community.common.chained_identity.ChainedIdentity*

The base class for models that can be explained.

expected_values
Abstract property to get the expected values.

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

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

static explainable_model_type(*self*)
Retrieve the model type.

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

model
Abstract property to get the underlying model.

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

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

interpret_community.mimic.models.lightgbm_model module

Defines an explainable lightgbm model.

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

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

available_explanations = ['global', 'local']

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 or scipy array*) – 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(*self*)

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.

param dataset The dataset to train the model on.

type dataset numpy or scipy array

param labels The labels to train the model on.

type labels numpy or scipy array

If multiclass=True, uses the parameters for LGBMClassifier: Build a gradient boosting model from the training set (X, y).

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input feature matrix.

y [arraylike of shape = [n_samples]] The target values (class labels in classification, real numbers in regression).

sample_weight [arraylike of shape = [n_samples] or None, optional (default=None)] Weights of training data.

init_score [arraylike of shape = [n_samples] or None, optional (default=None)] Init score of training data.

eval_set [list or None, optional (default=None)] A list of (X, y) tuple pairs to use as validation sets.

eval_names [list of strings or None, optional (default=None)] Names of eval_set.

eval_sample_weight [list of arrays or None, optional (default=None)] Weights of eval data.

eval_class_weight [list or None, optional (default=None)] Class weights of eval data.

eval_init_score [list of arrays or None, optional (default=None)] Init score of eval data.

eval_metric [string, list of strings, callable or None, optional (default=None)] If string, it should be a builtin evaluation metric to use. If callable, it should be a custom evaluation metric, see note below for more details. In either case, the `metric` from the model parameters will be evaluated and used as well. Default: 'l2' for LGBMRegressor, 'logloss' for LGBMClassifier, 'ndcg' for LGBMRanker.

early_stopping_rounds [int or None, optional (default=None)] Activates early stopping. The model will train until the validation score stops improving. Validation score needs to improve at least every `early_stopping_rounds` round(s) to continue training. Requires at least one validation data and one metric. If there's more than one, will check all of them. But the training data is ignored anyway. To check only the first metric, set the `first_metric_only` parameter to `True` in additional parameters `kwargs` of the model constructor.

verbose [bool or int, optional (default=True)] Requires at least one evaluation data. If True, the eval metric on the eval set is printed at each boosting stage. If int, the eval metric on the eval

set is printed at every `verbose` boosting stage. The last boosting stage or the boosting stage found by using `early_stopping_rounds` is also printed.

Example

With `verbose` = 4 and at least one item in `eval_set`, an evaluation metric is printed every 4 (instead of 1) boosting stages.

feature_name [list of strings or 'auto', optional (default='auto')] Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.

categorical_feature [list of strings or int, or 'auto', optional (default='auto')] Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas unordered categorical columns are used. All values in categorical features should be less than int32 max value (2147483647). Large values could be memory consuming. Consider using consecutive integers starting from zero. All negative values in categorical features will be treated as missing values. The output cannot be monotonically constrained with respect to a categorical feature.

callbacks [list of callback functions or None, optional (default=None)] List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.

init_model [string, Booster, LGBMModel or None, optional (default=None)] Filename of LightGBM model, Booster instance or LGBMModel instance used for continue training.

Returns

self [object] Returns self.

Note

Custom eval function expects a callable with following signatures: `func(y_true, y_pred)`, `func(y_true, y_pred, weight)` or `func(y_true, y_pred, weight, group)` and returns (eval_name, eval_result, is_higher_better) or list of (eval_name, eval_result, is_higher_better):

y_true [arraylike of shape = [n_samples]] The target values.

y_pred [arraylike of shape = [n_samples] or shape = [n_samples * n_classes] (for multiclass task)] The predicted values.

weight [arraylike of shape = [n_samples]] The weight of samples.

group [arraylike] Group/query data, used for ranking task.

eval_name [string] The name of evaluation function (without whitespaces).

eval_result [float] The eval result.

is_higher_better [bool] Is eval result higher better, e.g. AUC is `is_higher_better`.

For binary task, the y_pred is probability of positive class (or margin in case of custom `objective`). For multiclass task, the y_pred is group by class_id first, then group by row_id. If you want to get ith row y_pred in jth class, the access way is y_pred[j * num_data + i].

Otherwise, if multiclass=False, uses the parameters for LGBMRegressor: Build a gradient boosting model from the training set (X, y).

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input feature matrix.

y [arraylike of shape = [n_samples]] The target values (class labels in classification, real numbers in regression).

sample_weight [arraylike of shape = [n_samples] or None, optional (default=None)] Weights of training data.

init_score [arraylike of shape = [n_samples] or None, optional (default=None)] Init score of training data.

eval_set [list or None, optional (default=None)] A list of (X, y) tuple pairs to use as validation sets.

eval_names [list of strings or None, optional (default=None)] Names of eval_set.

eval_sample_weight [list of arrays or None, optional (default=None)] Weights of eval data.

eval_init_score [list of arrays or None, optional (default=None)] Init score of eval data.

eval_metric [string, list of strings, callable or None, optional (default=None)] If string, it should be a builtin evaluation metric to use. If callable, it should be a custom evaluation metric, see note below for more details. In either case, the `metric` from the model parameters will be evaluated and used as well. Default: 'l2' for LGBMRegressor, 'logloss' for LGBMClassifier, 'ndcg' for LGBMRanker.

early_stopping_rounds [int or None, optional (default=None)] Activates early stopping. The model will train until the validation score stops improving. Validation score needs to improve at least every `early_stopping_rounds` round(s) to continue training. Requires at least one validation data and one metric. If there's more than one, will check all of them. But the training data is ignored anyway. To check only the first metric, set the `first_metric_only` parameter to `True` in additional parameters `kwargs` of the model constructor.

verbose [bool or int, optional (default=True)] Requires at least one evaluation data. If True, the eval metric on the eval set is printed at each boosting stage. If int, the eval metric on the eval set is printed at every `verbose` boosting stage. The last boosting stage or the boosting stage found by using `early_stopping_rounds` is also printed.

Example

With `verbose` = 4 and at least one item in `eval_set`, an evaluation metric is printed every 4 (instead of 1) boosting stages.

feature_name [list of strings or 'auto', optional (default='auto')] Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.

categorical_feature [list of strings or int, or 'auto', optional (default='auto')] Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas unordered categorical columns are used. All values in categorical features should be less than int32 max value (2147483647). Large values could be memory consuming. Consider using consecutive integers starting from zero. All negative values in categorical features will be treated as missing values. The output cannot be monotonically constrained with respect to a categorical feature.

callbacks [list of callback functions or None, optional (default=None)] List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.

init_model [string, Booster, LGBMModel or None, optional (default=None)] Filename of LightGBM model, Booster instance or LGBMModel instance used for continue training.

Returns

self [object] Returns self.

Note

Custom eval function expects a callable with following signatures: `func(y_true, y_pred)`, `func(y_true, y_pred, weight)` or `func(y_true, y_pred, weight, group)` and returns (eval_name, eval_result, is_higher_better) or list of (eval_name, eval_result, is_higher_better):

y_true [arraylike of shape = [n_samples]] The target values.

y_pred [arraylike of shape = [n_samples] or shape = [n_samples * n_classes] (for multiclass task)] The predicted values.

weight [arraylike of shape = [n_samples]] The weight of samples.

group [arraylike] Group/query data, used for ranking task.

eval_name [string] The name of evaluation function (without whitespaces).

eval_result [float] The eval result.

is_higher_better [bool] Is eval result higher better, e.g. AUC is `is_higher_better`.

For binary task, the y_pred is probability of positive class (or margin in case of custom `objective`). For multiclass task, the y_pred is group by class_id first, then group by row_id. If you want to get ith row y_pred in jth class, the access way is y_pred[j * num_data + i].

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.

param dataset The dataset to predict on.

type dataset numpy or scipy array

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LGBMClassifier: Return the predicted value for each sample.

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input features matrix.

raw_score [bool, optional (default=False)] Whether to predict raw scores.

start_iteration [int, optional (default=0)] Start index of the iteration to predict. If <= 0, starts from the first iteration.

num_iteration [int or None, optional (default=None)] Total number of iterations used in the prediction. If None, if the best iteration exists and start_iteration <= 0, the best iteration is used; otherwise, all iterations from `start_iteration` are used (no limits). If <= 0, all iterations from `start_iteration` are used (no limits).

pred_leaf [bool, optional (default=False)] Whether to predict leaf index.

pred_contrib [bool, optional (default=False)] Whether to predict feature contributions.

Note: If you want to get more explanations for your model's predictions using SHAP values, like SHAP interaction values, you can install the shap package (https://github.com/slundberg/shap). Note that unlike the shap package, with `pred_contrib` we return a matrix with an extra column, where the last column is the expected value.

---

kwargs Other parameters for the prediction.

Returns

predicted_result [arraylike of shape = [n_samples] or shape = [n_samples, n_classes]] The predicted values.

X_leaves [arraylike of shape = [n_samples, n_trees] or shape = [n_samples, n_trees * n_classes]] If `pred_leaf=True`, the predicted leaf of every tree for each sample.

X_SHAP_values [arraylike of shape = [n_samples, n_features + 1] or shape = [n_samples, (n_features + 1) * n_classes] or list with n_classes length of such objects] If `pred_contrib=True`, the feature contributions for each sample.

Otherwise, if multiclass=False, uses the parameters for LGBMRegressor: Return the predicted value for each sample.

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input features matrix.

raw_score [bool, optional (default=False)] Whether to predict raw scores.

start_iteration [int, optional (default=0)] Start index of the iteration to predict. If <= 0, starts from the first iteration.

num_iteration [int or None, optional (default=None)] Total number of iterations used in the prediction. If None, if the best iteration exists and start_iteration <= 0, the best iteration is used; otherwise, all iterations from `start_iteration` are used (no limits). If <= 0, all iterations from `start_iteration` are used (no limits).

pred_leaf [bool, optional (default=False)] Whether to predict leaf index.

pred_contrib [bool, optional (default=False)] Whether to predict feature contributions.

Note: If you want to get more explanations for your model's predictions using SHAP values, like SHAP interaction values, you can install the shap package (https://github.com/slundberg/shap). Note that unlike the shap package, with `pred_contrib` we return a matrix with an extra column, where the last column is the expected value.

---

kwargs Other parameters for the prediction.

Returns

predicted_result [arraylike of shape = [n_samples] or shape = [n_samples, n_classes]] The predicted values.

X_leaves [arraylike of shape = [n_samples, n_trees] or shape = [n_samples, n_trees * n_classes]] If `pred_leaf=True`, the predicted leaf of every tree for each sample.

X_SHAP_values [arraylike of shape = [n_samples, n_features + 1] or shape = [n_samples, (n_features + 1) * n_classes] or list with n_classes length of such objects] If `pred_contrib=True`, the feature contributions for each sample.

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

param dataset The dataset to predict probabilities on.

type dataset numpy or scipy array

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LGBMClassifier: Return the predicted probability for each class for each sample.

Parameters

X [arraylike or sparse matrix of shape = [n_samples, n_features]] Input features matrix.

raw_score [bool, optional (default=False)] Whether to predict raw scores.

start_iteration [int, optional (default=0)] Start index of the iteration to predict. If <= 0, starts from the first iteration.

num_iteration [int or None, optional (default=None)] Total number of iterations used in the prediction. If None, if the best iteration exists and start_iteration <= 0, the best iteration is used; otherwise, all iterations from `start_iteration` are used (no limits). If <= 0, all iterations from `start_iteration` are used (no limits).

pred_leaf [bool, optional (default=False)] Whether to predict leaf index.

pred_contrib [bool, optional (default=False)] Whether to predict feature contributions.

Note: If you want to get more explanations for your model's predictions using SHAP values, like SHAP interaction values, you can install the shap package (https://github.com/slundberg/shap). Note that unlike the shap package, with `pred_contrib` we return a matrix with an extra column, where the last column is the expected value.

---

kwargs Other parameters for the prediction.

Returns

predicted_probability [arraylike of shape = [n_samples, n_classes]] The predicted probability for each class for each sample.

X_leaves [arraylike of shape = [n_samples, n_trees * n_classes]] If `pred_leaf=True`, the predicted leaf of every tree for each sample.

X_SHAP_values [arraylike of shape = [n_samples, (n_features + 1) * n_classes] or list with n_classes length of such objects] If `pred_contrib=True`, the feature contributions for each sample.

Otherwise predict_proba is not supported for regression or binary classification.

`interpret_community.mimic.models.lightgbm_model.`**lgbm_predict_decorator**(*predict*)
Decorate the predict method, temporary workaround for sparse case until TreeExplainer support is added.

Parameters **predict** (*method*) – The prediction method from lightgbm learner to be densified.

interpret_community.mimic.models.linear_model module

Defines an explainable linear model.

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

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

available_explanations = ['global', 'local']

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 or scipy array*) – 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(*self*)

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 or scipy array

param labels The labels to train the model on.

type labels numpy or scipy 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 : returns an instance of self.

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 or scipy array

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

Returns

C [array, shape [n_samples]] Predicted class label per sample.

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

Predict using the linear model.

Parameters

X [array_like 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 or scipy array

return The predictions of the model.

rtype list

If multiclass=True, uses the parameters for LogisticRegression:

Probability estimates.

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

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

Parameters

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

Returns

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

Otherwise predict_proba is not supported for regression or binary classification.

class interpret_community.mimic.models.linear_model.**LinearExplainer**(*model*, *data*, *feature_dependence='interventional*

Bases: sphinx.ext.autodoc.importer._MockObject

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 or scipy array*) – 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']

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 or scipy array*) – 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 or scipy array

param labels The labels to train the model on.

type labels numpy or scipy 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 : 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 : returns an instance of self.

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 or scipy array

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

Returns

C [array, shape [n_samples]] Predicted class label per 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)

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 or scipy array

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, http://www.research.ibm.com/people/z/zadrozny/kdd2002Transf.pdf

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

Otherwise predict_proba is not supported for regression or binary classification.

interpret_community.mimic.models.tree_model module

Defines an explainable tree model.

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

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

available_explanations = ['global', 'local']

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 or scipy array*) – 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(*self*)
Retrieve the model type.

Returns Tree explainable model type.

Return type *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 or scipy array

param labels The labels to train the model on.

type labels numpy or scipy 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 [arraylike of shape (n_samples, n_features), default=None] The indexes of the sorted training input samples. If many tree are grown on the same dataset, this allows the ordering to be cached between trees. If None, the data will be sorted here. Don't use this parameter unless you know what to do.

Returns

self [DecisionTreeClassifier] Fitted estimator.

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

Parameters

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

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

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

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

X_idx_sorted [arraylike of shape (n_samples, n_features), default=None] The indexes of the sorted training input samples. If many tree are grown on the same dataset, this allows the ordering to be cached between trees. If None, the data will be sorted here. Don't use this parameter unless you know what to do.

Returns

self [DecisionTreeRegressor] Fitted estimator.

model

Retrieve the underlying model.

Returns The decision tree model, either classifier or regressor.

Return type Union[DecisionTreeClassifier, DecisionTreeRegressor]

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

Call tree predict to predict labels using the explainable model.

param dataset The dataset to predict on.

type dataset numpy or scipy array

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 or scipy array

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: 'default'>*, *categorical_features=None*, *model_task=<ModelTask.Unknown: 'unknown'>*, *reset_index=<ResetIndex.Ignore: '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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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

Module for interaction with MLflow.

interpret_community.mlflow.**get_explanation**(*run_id*, *name*)
Download and deserialize an explanation that has been logged to MLflow.

Parameters

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

Returns The rehydrated explanation.

Return type Explanation

interpret_community.mlflow.**log_explanation**(*name*, *explanation*)
Log the explanation to MLflow using MLflow model logging.

Parameters

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

interpret_community.mlflow.**save_model**(*path*, *loader_module=None*, *data_path=None*, *conda_env=None*, *mlflow_model=None*, ***kwargs*)
Save the explanation locally using the MLflow model format.

This function is necessary for log_explanation to work properly.

Parameters

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

Returns The MLflow model representation of the explanation.

Return type mlflow.models.Model

Submodules

interpret_community.mlflow.mlflow module

interpret_community.mlflow.mlflow.**get_explanation**(*run_id*, *name*)
Download and deserialize an explanation that has been logged to MLflow.

Parameters

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

Returns The rehydrated explanation.

Return type Explanation

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

Log the explanation to MLflow using MLflow model logging.

Parameters

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

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

Save the explanation locally using the MLflow model format.

This function is necessary for log_explanation to work properly.

Parameters

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

Returns The MLflow model representation of the explanation.

Return type mlflow.models.Model

interpret_community.permutation package

Module for permutation feature importance.

class interpret_community.permutation.**PFIExplainer**(*model*, *is_function=False*, *metric=None*, *metric_args=None*, *is_error_metric=False*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations=False*, *seed=0*, *for_classifier_use_predict_proba=False*, *show_progress=True*, *model_task=<ModelTask.Unknown: '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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model.
- **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, Labe-

lEncoder, 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**

Bases: str, enum.Enum

The metric to use for PFIExplainer.

AVERAGE_PRECISION_SCORE = 'average_precision_score'

EXPLAINED_VARIANCE_SCORE = 'explained_variance_score'

F1_SCORE = 'f1_score'

FBETA_SCORE = 'fbeta_score'

MEAN_ABSOLUTE_ERROR = 'mean_absolute_error'

MEAN_SQUARED_ERROR = 'mean_squared_error'

MEAN_SQUARED_LOG_ERROR = 'mean_squared_log_error'

MEDIAN_ABSOLUTE_ERROR = 'median_absolute_error'

PRECISION_SCORE = 'precision_score'

R2_SCORE = 'r2_score'

RECALL_SCORE = 'recall_score'

interpret_community.permutation.permutation_importance module

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

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

class interpret_community.permutation.permutation_importance.**PFIExplainer**(*model*, *is_function=False*, *metric=None*, *metric_args=None*, *is_error_metric=False*, *explain_subset=None*, *features=None*, *classes=None*, *transformations=None*, *allow_all_transformations*, *seed=0*, *for_classifier_use_predi*, *show_progress=True*, *model_task=<ModelTas*, *'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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The black box model or function (if is_function is True) to be explained. Also known as the teacher model.
- **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, Labe-

lEncoder, 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: '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** (*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 iml.datatypes.DenseData 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.**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: '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** (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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The model to explain or function if is_function is True.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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.**TreeExplainer**(*model*, *explain_subset=None*, *features=None*, *classes=None*, *shap_values_output=<ShapValuesOutput.DEFAULT: '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** (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

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** (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 iml.datatypes.DenseData 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

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: '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** (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 iml.datatypes.DenseData 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.

write(*data*)

Write the given data to logger.

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

interpret_community.shap.kernel_explainer module

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

class interpret_community.shap.kernel_explainer.**KernelExplainer**(*model*, *initialization_examples*, *is_function=False*, *explain_subset=None*, *nsamples='auto'*, *features=None*, *classes=None*, *nclusters=10*, *show_progress=True*, *transformations=None*, *allow_all_transformations=False*, *model_task=<ModelTask.Unknown: '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** (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** (*model that implements sklearn.predict or sklearn.predict_proba or function that accepts a 2d ndarray*) – The model to explain or function if is_function is True.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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
]
[
```

(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. 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** (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 iml.datatypes.DenseData 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.DE '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** (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

Explanation Dashboard Class.

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.

class DashboardService(*port*, *public_ip*, *with_credentials=False*)

Bases: object

get_base_url()

run()

default_template = <Template 'inlineDashboard.html'>

env = <jinja2.environment.Environment object>

explanations = {}

model_count = 0

service = None

Submodules

interpret_community.widget.explanation_dashboard module

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

Explanation Dashboard Class.

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.

class DashboardService(*port*, *public_ip*, *with_credentials=False*)

Bases: object

get_base_url()

run()

default_template = <Template 'inlineDashboard.html'>

env = <jinja2.environment.Environment object>

explanations = {}

model_count = 0

service = None

interpret_community.widget.explanation_dashboard.**generate_inline_html**(*explanation_input_object*, *local_url*)

interpret_community.widget.explanation_dashboard_input module

Defines the Explanation dashboard class.

class interpret_community.widget.explanation_dashboard_input.**ExplanationDashboardInput**(*explanation*, *model=None*, *dataset=None*, *true_y=None*, *classes=None*, *features=None*, *predict_url=None*, *locale=None*, *with_model=False*)

Bases: `object`

Represents an explanation as all the pieces that can be serialized and passed to JavaScript.

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. Will overwrite any set on explanation object already
- **true_y** (*numpy.array or list[]*) – The true labels for the provided dataset. Will overwrite any set on explanation object already.
- **classes** (*numpy.array or list[]*) – The class names.
- **features** (*numpy.array or list[]*) – Feature names.

enable_predict_url()

on_predict(*data*)

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: 'unknown'>*, ***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** (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** (*model that implements sklearn.predict() or sklearn.predict_proba() or pipeline function that accepts a 2d ndarray*) – The model or pipeline to explain.
- **initialization_examples** (*numpy.array or pandas.DataFrame or iml.datatypes.DenseData 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 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

A

B

C

D

E

F

G

K

L

M

R

S

T

U

V

W