interpret-community

Release 0.31.0

Microsoft

Jan 12, 2024

CONTENTS

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

The code is available from GitHub.

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

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

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

CHAPTER

ONE

INSTALLATION

The package can be installed from pypi with:

```
pip install interpret-community
```

1.1 Overview

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

Interpret-Community

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

1.2 Getting Started

The package can be installed from pypi with:

```
pip install interpret-community
```

You can use Anaconda to simplify package and environment management.

To setup on your local machine:

1.3 Supported Models

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

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

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

1.4 Supported Explainers

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

Table 1: Explainers

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

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

Table 2: TabularExplainer

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

1.5 Example Notebooks

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

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

1.6 Use Interpret-Community

1.6.1 Interpretability in training

1. Train your model

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

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

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

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

```
  from interpret.ext.blackbox import TabularExplainer

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

or

```
from interpret.ext.blackbox import MimicExplainer

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

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

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

or

```
from interpret.ext.blackbox import PFIExplainer

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

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

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

1.7 Importance Values

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

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

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

Some feature importance values also have useful properties.

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

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

1.7.1 Overall (Global) feature importance values

Get the aggregate feature importance values.

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

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

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


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

1.7.2 Instance-level (Local) feature importance values

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

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

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

or

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

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

1.8 Raw feature transformations

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

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

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

```
from sklearn.compose import ColumnTransformer

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

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

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

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

(continues on next page)

(continued from previous page)

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


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

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

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

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

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

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

transformations = numeric_transformations + categorical_transformations

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

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

1.9 Visualizations

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

```
pip install raiwidgets
```

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

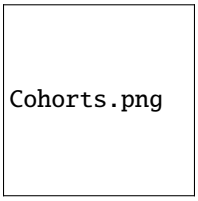
```
from raiwidgets import ExplanationDashboard

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

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

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

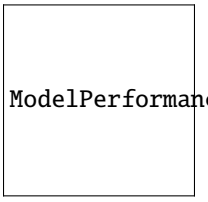
Note: Click on "Open in a new tab" on the top left corner to get a better view of the dashboard in a new tab.

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



1.9.1 Model performance

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



1.9.2 Dataset explorer

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

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

1.9.3 Aggregate feature importance (global explanation)

This view consists of two charts:

Table 3: Aggregate feature importance

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

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

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

Table 4: Individual feature importance

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

1.10 Contributing

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

1.10.1 Acceptance criteria

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

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

1.10.2 Development process

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

```
pip install -e .
```

1.10.3 Testing

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

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

The unit tests can then be run via the command:

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

Code coverage can be run locally via the command:

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

Notebook tests can also be run via the command:

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

1.10.4 Linting

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

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

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

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

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

```
isort . -c
```

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

```
isort .
```

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

1.11 API Reference

1.11.1 interpret_community package

Subpackages

interpret_community.adapter package

Submodules

interpret_community.adapter.explanation_adapter module

interpret_community.common package

Common infrastructure, class hierarchy and utilities for model explanations.

class interpret_community.common.**ModelSummary**

Bases: object

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

add_from_get_model_summary(*name*, *artifact_metadata_tuple*)

Update artifacts and metadata with new information.

Parameters

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

get_artifacts()

Get the list of artifacts.

Returns
Artifact list.

Return type
list[list[dict]]

get_metadata_dictionary()

Get the combined dictionary of metadata.

Returns
Metadata dictionary.

Return type
dict

Subpackages

interpret_community.common.model_wrapper package

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

Submodules

interpret_community.common.aggregate module

interpret_community.common.base_explainer module

Defines the base explainer API to create explanations.

class interpret_community.common.base_explainer.**BaseExplainer**(**args*, ***kwargs*)

Bases: *GlobalExplainer*, *LocalExplainer*

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

class interpret_community.common.base_explainer.**GlobalExplainer**(**args*, ***kwargs*)

Bases: ABC, *ChainedIdentity*

The base class for explainers that create global explanations.

abstract explain_global(**args*, ***kwargs*)

Abstract method to globally explain the given model.

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

Returns
A model explanation object containing the global explanation.

Return type
GlobalExplanation

class interpret_community.common.base_explainer.**LocalExplainer**(**args*, ***kwargs*)

Bases: ABC, *ChainedIdentity*

The base class for explainers that create local explanations.

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

Abstract method to explain local instances.

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

Returns
A model explanation object containing the local explanation.

Return type
LocalExplanation

interpret_community.common.blackbox_explainer module

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'

CHECK_ADDITIVITY = 'check_additivity'

CLASSES = 'classes'

CLASSIFICATION = 'classification'

EVAL_DATA = 'eval_data'

EVAL_Y_PRED = 'eval_y_predicted'

EVAL_Y_PRED_PROBA = 'eval_y_predicted_proba'

```
    EXPECTED_VALUES = 'expected_values'
    EXPLAIN_SUBSET = 'explain_subset'
    EXPLANATION_ID = 'explanation_id'
    FEATURES = 'features'
    GLOBAL_IMPORTANCE_NAMES = 'global_importance_names'
    GLOBAL_IMPORTANCE_RANK = 'global_importance_rank'
    GLOBAL_IMPORTANCE_VALUES = 'global_importance_values'
    GLOBAL_NAMES = 'global_names'
    GLOBAL_RANK = 'global_rank'
    GLOBAL_VALUES = 'global_values'
    ID = 'id'
    INCLUDE_LOCAL = 'include_local'
    INIT_DATA = 'init_data'
    IS_ENG = 'is_engineered'
    IS_LOCAL_SPARSE = 'is_local_sparse'
    IS_RAW = 'is_raw'
    LOCAL_EXPLANATION = 'local_explanation'
    LOCAL_IMPORTANCE_VALUES = 'local_importance_values'
    METHOD = 'method'
    MODEL_ID = 'model_id'
    MODEL_TASK = 'model_task'
    MODEL_TYPE = 'model_type'
    NUM_CLASSES = 'num_classes'
    NUM_EXAMPLES = 'num_examples'
    NUM_FEATURES = 'num_features'
    PER_CLASS_NAMES = 'per_class_names'
    PER_CLASS_RANK = 'per_class_rank'
    PER_CLASS_VALUES = 'per_class_values'
    PROBABILITIES = 'probabilities'
    SAMPLING_POLICY = 'sampling_policy'
    SHAP_VALUES_OUTPUT = 'shap_values_output'
```

classmethod get_private(*explain_param*)

Return the private version of the ExplainParams property.

Parameters

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

Returns

The private version of the property.

Return type

str

classmethod get_serializable()

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

Parameters

cls (ExplainParams) – ExplainParams input class.

Returns

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

Return type

set[str]

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

Bases: object

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

CLASSIFICATION = 'classification'

DATA = 'data_type'

EXPLAIN = 'explain_type'

EXPLAINER = 'explainer'

FUNCTION = 'function'

GLOBAL = 'global'

HAN = 'han'

IS_ENG = 'is_engineered'

IS_RAW = 'is_raw'

LIME = 'lime'

LOCAL = 'local'

METHOD = 'method'

MIMIC = 'mimic'

MODEL = 'model_type'

MODEL_CLASS = 'model_class'

MODEL_TASK = 'model_task'

PFI = 'pfi'

REGRESSION = 'regression'

SHAP = 'shap'

SHAP_DEEP = 'shap_deep'

SHAP_GPU_KERNEL = 'shap_gpu_kernel'

SHAP_KERNEL = 'shap_kernel'

SHAP_LINEAR = 'shap_linear'

SHAP_TREE = 'shap_tree'

TABULAR = 'tabular'

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

Bases: str, Enum

Provide constants for the explainable model type.

LINEAR_EXPLAINABLE_MODEL_TYPE = 'linear_explainable_model_type'

TREE_EXPLAINABLE_MODEL_TYPE = 'tree_explainable_model_type'

class interpret_community.common.constants.**ExplanationParams**

Bases: object

Provide constants for explanation parameters.

CLASSES = 'classes'

EXPECTED_VALUES = 'expected_values'

class interpret_community.common.constants.**Extension**

Bases: object

Provide constants for extensions to interpret package.

BLACKBOX = 'blackbox'

GLASSBOX = 'model'

GLOBAL = 'global'

GREYBOX = 'specific'

LOCAL = 'local'

class interpret_community.common.constants.**InterpretData**

Bases: object

Provide Data and Visualize constants for interpret core.

BASE_VALUE = 'Base Value'

EXPLANATION_CLASS_DIMENSION = 'explanation_class_dimension'

EXPLANATION_TYPE = 'explanation_type'

EXTRA = 'extra'

FEATURE_LIST = 'feature_list'

GLOBAL_FEATURE_IMPORTANCE = 'global_feature_importance'

INTERCEPT = 'intercept'

LOCAL_FEATURE_IMPORTANCE = 'local_feature_importance'

MLI = 'mli'

MULTICLASS = 'multiclass'

NAMES = 'names'

OVERALL = 'overall'

PERF = 'perf'

SCORES = 'scores'

SINGLE = 'single'

SPECIFIC = 'specific'

TYPE = 'type'

UNIVARIATE = 'univariate'

VALUE = 'value'

VALUES = 'values'

class interpret_community.common.constants.**LightGBMParams**

Bases: object

Provide constants for LightGBM.

CATEGORICAL_FEATURE = 'categorical_feature'

MIN_DATA_IN_LEAF = 'min_data_in_leaf'

class interpret_community.common.constants.**LightGBMSerializationConstants**

Bases: object

Provide internal class that defines fields used for MimicExplainer serialization.

IDENTITY = '_identity'

LOGGER = '_logger'

MODEL_STR = 'model_str'

MULTICLASS = 'multiclass'

OBJECTIVE = 'objective'

REGRESSION = 'regression'

TREE_EXPLAINER = '_tree_explainer'

enum_properties = ['_shap_values_output']

nonify_properties = ['_logger', '_tree_explainer']

save_properties = ['_lgbm']

class interpret_community.common.constants.MimicSerializationConstants

Bases: object

Provide internal class that defines fields used for MimicExplainer serialization.

ALLOW_ALL_TRANSFORMATIONS = '_allow_all_transformations'

FUNCTION = 'function'

IDENTITY = '_identity'

INITIALIZATION_EXAMPLES = 'initialization_examples'

LOGGER = '_logger'

MODEL = 'model'

ORIGINAL_EVAL_EXAMPLES = '_original_eval_examples'

PREDICT_PROBA_FLAG = 'predict_proba_flag'

RESET_INDEX = 'reset_index'

TIMESTAMP_FEATURIZER = '_timestamp_featurizer'

enum_properties = ['_shap_values_output']

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

save_properties = ['surrogate_model']

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

Bases: str, Enum

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

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

Classification = 'classification'

Regression = 'regression'

Unknown = 'unknown'

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

Bases: str, Enum

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

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

Ignore = 'ignore'

Reset = 'reset'

ResetTeacher = 'reset_teacher'

class interpret_community.common.constants.**SHAPDefaults**

Bases: object

Provide constants for default values to SHAP.

INDEPENDENT = 'independent'

class interpret_community.common.constants.**SKLearn**

Bases: object

Provide scikit-learn related constants.

EXAMPLES = 'examples'

LABELS = 'labels'

PREDICTIONS = 'predictions'

PREDICT_PROBA = 'predict_proba'

class interpret_community.common.constants.**Scipy**

Bases: object

Provide scipy related constants.

CSR_FORMAT = 'csr'

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

Bases: str, Enum

Provide constants for the SHAP values output from the explainer.

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

DEFAULT = 'default'

PROBABILITY = 'probability'

TEACHER_PROBABILITY = 'teacher_probability'

class interpret_community.common.constants.**Spacy**

Bases: object

Provide spaCy related constants.

EN = 'en'

NER = 'ner'

TAGGER = 'tagger'

class interpret_community.common.constants.**Tensorflow**

Bases: object

Provide TensorFlow and TensorBoard related constants.

CPU0 = '/CPU:0'

TFLOG = 'tflog'

interpret_community.common.error_handling module

Defines error handling utilities.

interpret_community.common.exception module

Defines different types of exceptions that this package can raise.

exception interpret_community.common.exception.**ScenarioNotSupportedException**

Bases: Exception

An exception indicating that some scenario is not supported.

Parameters

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

interpret_community.common.explanation_utils module

interpret_community.common.gpu_kmeans module

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

class interpret_community.common.gpu_kmeans.**Data**

Bases: object

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

Bases: *Data*

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

Summarize a dataset with k mean samples weighted by the number of data points they each represent.

Parameters

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

Returns

DenseData object.

Return type

DenseData

interpret_community.common.metrics module

Defines metrics for validating model explanations.

interpret_community.common.metrics.**dcg**(*validate_order*, *ground_truth_order_relevance*, *top_values=10*)

Compute the discounted cumulative gain (DCG).

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

Parameters

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

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

Compute the normalized discounted cumulative gain (NDCG).

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

Parameters

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

interpret_community.common.model_summary module

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

class interpret_community.common.model_summary.**ModelSummary**

Bases: object

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

add_from_get_model_summary(*name*, *artifact_metadata_tuple*)

Update artifacts and metadata with new information.

Parameters

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

get_artifacts()

Get the list of artifacts.

Returns

Artifact list.

Return type
list[list[dict]]

`get_metadata_dictionary()`

Get the combined dictionary of metadata.

Returns
Metadata dictionary.

Return type
dict

interpret_community.common.policy module

Defines explanation policies.

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

Bases: *ChainedIdentity*

Defines the sampling policy for downsampling the evaluation examples.

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

Parameters

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

Return type
dict

Returns
The arguments for the sampling policy

property allow_eval_sampling

Get whether to allow sampling of evaluation data.

Returns
Whether to allow sampling of evaluation data.

Return type
bool

property max_dim_clustering

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

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

Return type
int

property sampling_method

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

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

Return type
str

interpret_community.common.progress module

Defines utilities for getting progress status for explanation.

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

Get the tqdm progress bar function.

Parameters

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

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

Return type
function

interpret_community.common.serialization_utils module

interpret_community.common.structured_model_explainer module

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

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

Bases: *BaseExplainer*

The base PureStructuredModelExplainer API for explainers used on specific models.

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

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

Bases: *BaseExplainer*

The base StructuredInitModelExplainer API for explainers.

Used on specific models that require initialization examples.

Parameters

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

interpret_community.common.warnings_suppressor module

Suppresses warnings on imports.

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

Bases: object

Context manager to suppress warnings from shap.

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

Bases: object

Context manager to suppress warnings from tensorflow.

interpret_community.dataset package

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

Subpackages

interpret_community.dataset.dataset_wrapper package

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

Submodules

interpret_community.dataset.decorator module

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

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

Decorate a constructor to wrap initialization examples in a DatasetWrapper.

Provided for convenience for tabular data explainers.

Parameters

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

interpret_community.dataset.decorator.**tabular_decorator**(*explain_func*)

Decorate an explanation function to wrap evaluation examples in a DatasetWrapper.

Parameters

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

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

interpret_community.explanation package

Submodules

interpret_community.explanation.explanation module

interpret_community.explanation.serialization module

interpret_community.lime package

Submodules

interpret_community.lime.lime_explainer module

interpret_community.mimic package

Subpackages

interpret_community.mimic.models package

Submodules

interpret_community.mimic.models.explainable_model module

interpret_community.mimic.models.lightgbm_model module

interpret_community.widget.explanation_dashboard module

Submodules

interpret_community.tabular_explainer module

interpret_community.version module

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

F

G

T

U

V

W