aix360

Release 0.1

Jul 31, 2023

Package Reference:

The AI Explainbability 360 toolkit is an open-source library that supports interpretability and explainability of data and machine learning models. The AI Explainability 360 Python package includes a comprehensive set of algorithms that cover different dimensions of explanations along with proxy explainability metrics.

For more information and installation instructions, see our GitHub page.

CHAPTER 1

Algorithms

1.1 Local White Box Explainers

1.1.1 Contrastive Explainers

1.1.2 SHAP Explainers

class aix360.algorithms.shap.shap_wrapper.**GradientExplainer**(**argv*, ***kwargs*)

This class wraps the source class GradientExplainer available in the SHAP library. Additional variables or functions from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize shap kernelexplainer object.

explain_instance(**argv*, ***kwargs*)

Explain one or more input instances.

set_params(**argv*, ***kwargs*)

Optionally, set parameters for the explainer.

class aix360.algorithms.shap.shap_wrapper.**DeepExplainer**(**argv*, ***kwargs*)

This class wraps the source class DeepExplainer available in the SHAP library. Additional variables or functions from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize shap kernelexplainer object.

explain_instance(**argv*, ***kwargs*)

Explain one or more input instances.

set_params(**argv*, ***kwargs*)

Optionally, set parameters for the explainer.

class aix360.algorithms.shap.shap_wrapper.**TreeExplainer**(**argv*, ***kwargs*)

This class wraps the source class TreeExplainer available in the SHAP library. Additional variables or functions

from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize shap kernelexplainer object.

explain_instance(**argv*, ***kwargs*)
Explain one or more input instances.

set_params(**argv*, ***kwargs*)
Optionally, set parameters for the explainer.

class aix360.algorithms.shap.shap_wrapper.**LinearExplainer**(**argv*, ***kwargs*)
This class wraps the source class Linearexplainer available in the SHAP library. Additional variables or functions from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize shap kernelexplainer object.

explain_instance(**argv*, ***kwargs*)
Explain one or more input instances.

set_params(**argv*, ***kwargs*)
Optionally, set parameters for the explainer.

1.1.3 Matching Explainers

class aix360.algorithms.matching.order_constraints.**OTMatchingExplainer**(*deactivate_bounds: bool = False*, *error_limit: float = 0.001*)
OTMatchingExplainer provides explainations for a matching that satisfies the transport polytope constraints. Given a matching, it produces a set of alternative matchings, where each alternate contrasts with the provided instance by a sparse set of salient matchings.[1].

This is akin to a search engine providing alternative suggestions relevant to a search string. OTMatchingExplainer aims to provide the same for matchings.

References

Initialize the OTMatchingExplainer

explain_instance(*matching: numpy.ndarray, costs: numpy.ndarray, constraints: Tuple[numpy.ndarray, numpy.ndarray], num_alternate_matchings: int = 1, search_thresholds: Tuple[float, float] = (0.5, 0.5), search_node_limit: int = 20, search_depth_limit: int = 1, search_match_pos_filter: Optional[List[Tuple[int, int]]] = None*)
Explain the matching

Parameters

- **matching** (*numpy 2d array*) – the matching to be explained.

[1] Fabian Lim, Laura Wynter, Shiau Hong Lim, "Order Constraints in Optimal Transport", 2022

- **costs** – (numpy 2d array): the (non-negative) matching costs used to obtain above matching.
- **constraints** (*numpy array, numpy array*) – the transport polytope row/column constraints.
- **num_alternate_matchings** (*int*) – the number of alternate matchings to return back.
- **search_node_limit** (*int*) – stop the search when this many nodes have been encountered.
- **search_depth_limit** (*int*) – do not progress beyond this tree level in the search
- **search_match_pos_filter** (*(int,int) array or None*) – if specified, this is a whitelist of positions (i,j) of candidate match positions
- **search_thresholds** (*float, float*) – thresholds used to pick the candidate match positions to search over.

Returns list of AlternateMatching explanations.

set_params(**args*, ***kwargs*)
Set parameters for the explainer.

1.2 Local Black Box Explainers

1.2.1 SHAP Explainers

class aix360.algorithms.shap.shap_wrapper.**KernelExplainer**(**argv*, ***kwargs*)
This class wraps the source class KernelExplainer available in the SHAP library. Additional variables or functions from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize shap kernelexplainer object.

explain_instance(**argv*, ***kwargs*)
Explain one ore more input instances.

set_params(**argv*, ***kwargs*)
Optionally, set parameters for the explainer.

1.2.2 LIME Explainers

class aix360.algorithms.lime.lime_wrapper.**LimeImageExplainer**(**argv*, ***kwargs*)
This class wraps the source class LimeImageExplainer available in the LIME library. Additional variables or functions from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize lime Image explainer object

explain_instance(**argv*, ***kwargs*)
Explain one or more input instances.

set_params(**argv*, ***kwargs*)
Optionally, set parameters for the explainer.

class aix360.algorithms.lime.lime_wrapper.**LimeTabularExplainer**(**argv*, ***kwargs*)

This class wraps the source class LimeTabularExplainer available in the LIME library. Additional variables or functions from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize lime Tabular Explainer object

explain_instance(**argv*, ***kwargs*)

Explain one or more input instances.

set_params(*verbose=0*)

Optionally, set parameters for the explainer.

class aix360.algorithms.lime.lime_wrapper.**LimeTextExplainer**(**argv*, ***kwargs*)

This class wraps the source class LimeTextExplainer available in the LIME library. Additional variables or functions from the source class can also be accessed via the 'explainer' object variable that is initialized in '__init__' function of this class.

Initialize lime text explainer object.

explain_instance(**argv*, ***kwargs*)

Explain one or more input instances.

set_params(**argv*, ***kwargs*)

Optionally, set parameters for the explainer.

1.2.3 Nearest Neighbor Contrastive Explainer

class aix360.algorithms.nncontrastive.nncontrastive.**NearestNeighborContrastiveExplainer**(*mod*
Cal
=
Nor
n_c
int
=
2,
met
ric:
str
=
'eu-
clid
nei
bor
int
=
3,
em-
bed
din
Uni
aix
=
<E
bed
din
Typ
'un
su-
per-
vise
em-
bed
din
em-
bed
din
int
=
8,
cat-
e-
gor
int
=
3,
cat-
e-
gor
str
=
'oh
nu-
mer
str
=

for each data point, for the nearest neighbor search. The dimensional reduction improves the reliability of the neighborhood search. Along with dimensionality reduction, the current implementation also allows for imposing class structure-driven orientation of the embedding space. For example, in a loan application, a high-income applicant and a low-income applicant may have very different evaluation criteria. The auto-encoder uses high-income and low-income tag classes during the auto-encoder training, ensuring instances with the same tags are in close neighborhoods.

The implementation allows exemplar selection in two ways, (1) contrastive exemplar selection is guided by a model prediction, and (2) the user explicitly provides an exemplar with a different class tag than the query instance (model-free). Given a query instance, the resulting explanation is a set of nearest neighbor exemplars with different class tags than the query instances.

NearestNeighborContrastiveExplainer initialization.

Parameters

- **model** (*Callable*) – Classification Model which will be used for contrastive explanation.
- **n_classes** (*int*) – Number of classes the classification produces.
- **metric** (*str*) – Distance metric for neighborhood finding. This metric is used to find neighborhood in the embedding space. The implementation internally uses Scipy KDTree for neighborhood search. See the documentation of scipy.spatial.distance and the metrics listed in distance_metrics for more information. Defaults to euclidean.
- **neighbors** (*int*) – Number of neighbors to fetch for producing the explanation. The NearestNeighborContrastiveExplainer uses these neighbors to produce explanation. In order to understand the variety in the neighborhood profile, higher value is suggested for this parameter, which impacts the size of the explanation produced. For fast greedy explanation lower value is suggested for this parameter. Defaults to 3.
- **embedding_type** (*Union[str, EmbeddingType]*) – This parameter controls the nature of the embedding produced. It can be set to supervised (EmbeddingType.SUPERVISED) or unsupervised (EmbeddingType.UNSUPERVISED). The unsupervised embedding ensures data distribution compliance, while supervised embedding allow imposing further structural constraints to the embedding by provided business tags during the model fit step. Defaults to EmbeddingType.UNSUPERVISED.
- **embedding_dim** (*int*) – Dimension of the produced embedding. Lower dimension allows faster search, while at the cost of lossy reconstruction. An appropriate emebedding_dim selection depends of the data complexity and available data. Defaults to 8.
- **category_enc_dim** (*int*) – Autoencoder handles categorical variable as embedding/one hot encoding. This parameter defines the internal dimension to be used by the auto-encoder to derive the categorical embedding. Defaults to 3.
- **category_encoding** (*str*) – Strategy specification for categorical variable handling. Supported values are 'ohe' (One hot encoding) and 'label' (Uses embedding). Defaults to "ohe".
- **numeric_scaling** (*str*) – Data scaling to be used on numeric columns for computational stability. This uses global scaling, i.e., applied uniformly over the entire training batch for all numeric columns. Supported values are minmax, standard, quantile. Defaults to None.
- **layers_config** (*List[int]*) – This is auto-encoder internals specification. Autoencoder uses MLP layers to derive the embedding, this parameter specifies number of hidden layers in the embedding, and their respective dimensions. Defaults to [16, 16].
- **encoder_activation** (*str*) – Activation function used by the auto-encoder encoding layers. Supports all activation function as enabled by tensorflow framework. Defaults to

"relu".

- **decoder_activation** (*str*) – Activation function used by the auto-encoder decoding layers. Support all activation functions as supported by the tensorflow framework. Defaults to "relu".
- **embedding_activation** (*str*) – This is embedding layer activation, this can be separately specified than hidden layer activation. Support all tensorflow activation function. Defaults to "tanh".
- **encoder_kernel_regularizer** (*str*) – Regularization for the encoder MLP kernel. Regularization results in stable prediction model. Defaults to "l1".
- **encoder_kernel_initializer** (*str*) – Initialization algorithm for the MLP kernel. Defaults to "glorot_uniform".
- **encoder_bias_initializer** (*str*) – Initialization algorithm for the MLP bias. Defaults to "zeros".
- **encoder_activity_regularizer** (*str*) – Encoder activity regularizer for MLP layers. Defaults to None.
- **decoder_kernel_regularizer** (*str*) – Kernel regularizer for the decoder MLP layer. All tensorflow regularizer algorithm are supported. Defaults to "l1".
- **decoder_kernel_initializer** (*str*) – Decoder MLP kernel weight initializer algorithm. All tensorflow initializer algorithms are supported. Defaults to "glorot_uniform".
- **decoder_bias_initializer** (*str*) – Decoder MLP bias initializer algorithm. All tensorflow initializer algorithms are supported. Defaults to "zeros".
- **decoder_activity_regularizer** (*str*) – Decoder activity regularization algorithm. All tensorflow regularizer algorithms are supported. Defaults to None.
- **decoder_last_layer_activation** (*str*) – Decoder last layer activation. This layer produces the input reconstruction. Supports all tensorflow supported activation function. Defaults to "linear".
- **embedding_activity_regularizer** (*str*) – Embedding activity regularization method. Uses default tensorflow framework, support all activity regularizer algorithm. Defaults to None.
- **classifier_layers** (*List[int]*) – Supervised auto-encoder uses classification layer for the structural constraint on the embedding. MLP layer for this classification task. This specifies the dimension of the MLP layer for this classification task. Defaults to [16].
- **classifier_activation** (*str*) – Supervised auto-encoder uses classification layer for the structural constraint on the embedding. For supervised auto-encoder activation of MLP layer classification layer. Defaults to "relu".
- **classifier_kernel_regularizer** (*str*) – Supervised auto-encoder uses classification layer for the structural constraint on the embedding. This parameter describes the kernel regularization for supervised auto-encoder MLP layer for classification. Defaults to None.
- **classifier_kernel_initilizer** (*str*) – Supervised auto-encoder uses classification layer for the structural constraint on the embedding. This parameter describes the kernel initialization algorithm for supervised auto-encoder MLP layer for the classification. Defaults to "glorot_uniform".
- **classifier_bias_initializer** (*str*) – Describes the MLP bias initialization algorithm, for the classification layer of supervised auto-encoder. Defaults to "zeros".

explain_instance(*x*, ***kwargs*)
Explain (local explanation) the model prediction for provided instance(s).

Parameters **x** (*Union[pd.DataFrame, np.ndarray]*) – input instance to be explained.

Additional Parameters:

neighbors (int): Number of neighbors Overrides neighbors parameter provided in the initializer.

Returns

explanation object Dictionary or list of dictionaries with keys: features, categorical_features, query, neighbors, distances.

Return type Union(List[dict], dict)

fit(*x: pandas.core.frame.DataFrame*, *y: numpy.ndarray = None*, *features: List[str] = None*, *categorical_features: List[str] = []*, *categorical_values: dict = {}*, *epochs: int = 5*, *batch_size: int = 128*, *verbose: int = 0*, *shuffle: bool = True*, *validation_fraction: float = 0*, *max_training_records: int = 10000*, *exemplars: pandas.core.frame.DataFrame = None*, *random_seed: int = None*, ***kwargs*)
Fit the explainer.

Parameters

- **x** (*pd.DataFrame*) – Training data.
- **y** (*np.ndarray*) – If provided these data labels is used for training supervised auto-encoder. This labels need not be same as data target classes from target class label. Defaults to None.
- **features** (*List[str]*) – Names of the features to be used. If not specified all the columns in the training data will be used as features. Defaults to None.
- **categorical_features** (*List[str]*) – Names of the categorical features in the data. This must match the column names. Defaults to [].
- **categorical_values** (*dict*) – Lookup dictionary for all categorical variables, list of possible categorical values. Defaults to {}.
- **epochs** (*int*) – Number of epochs to be used for auto-encoder training. Defaults to 5.
- **batch_size** (*int*) – Batch-size for the auto-encoder training. Should be smaller than the available data in training data. Defaults to 128.
- **verbose** (*int*) – Log verbosity. Defaults to 0.
- **shuffle** (*bool*) – Shuffle batch per epoch. Defaults to True.
- **validation_fraction** (*float*) – Fraction specifying the validation split during encoder training. Defaults to 0.
- **max_training_records** (*int*) – Maximum number of records to be used for the auto-encoder training. Enables explainer performance optimization. Defaults to 10000.
- **exemplars** (*pd.DataFrame*) – Exemplar neighbors to be used to compute explanations. If None, training dataset will be used as exemplars. Defaults to None.
- **random_seed** (*int*) – Random seed to fit auto encoder. Defaults to None

get_params(**argv*, ***kwargs*) → dict
Get parameters for the explainer.

set_exemplars(*x: Union[pandas.core.frame.DataFrame, numpy.ndarray]*)
Set user provided exemplars to guide contrastive exploration.

Parameters **x** (*Union[pd.DataFrame, np.ndarray]*) – Exemplar neighbors to be used to compute explanations.

set_params(**argv*, ***kwargs*)
Set parameters for the explainer.

1.2.4 Grouped Conditional Expectation (GroupedCE) Explainer

class aix360.algorithms.gce.gce.**GroupedCEExplainer**(*model: Callable, data: Union[int, List[List[object]]], feature_names: List[str] = None, n_samples: int = 25, features_selected: list = None, top_k_features: int = -1, feature_importance_method: str = 'SHAP', max_dataset_size: int = 10, random_seed: int = None, **kwargs*)

Grouped Conditional Expectation plots are generated for a given instance and set of features. They show how the model prediction is affected when a pair of features of a given instance are perturbed. The perturbed features can be either a provided subset of the input covariates, or the top K features based on the importance rank obtained by a global explainer (e.g., SHAP). If the user provides a single feature then the algorithm produces a standard ICE plot where the selected feature varies according to a linespace grid. If the user chooses more than one feature, then the explainer produces 3D ICE plots. Here, two features vary simultaneously according to a meshgrid, the output of the model is stored for each pair of values.

GroupedCEExplainer initialization.

Parameters

- **model** (*Callable*) – model prediction (predict/predict_proba) function that results a real value like probability or regressed value.
- **data** (*List[object]*) – Input dataset used for model training. Feature range is computed from this input dataset. The dataset is used in selected feature importance methods such as SHAP to determine top K features for group explanation.
- **feature_names** (*List[str]*) – List of valid numerical feature names in the input dataset. Defaults to None.
- **n_samples** (*int, optional*) – Number of discrete points sampled per feature. Defaults to 25.
- **features_selected** (*List[str], optional*) – List of features that will be considered in the explanation. If list contains single feature, GroupedCEExplainer return standard ICE explanation. Otherwise, returns grouped explanation for 3D ICE plots.
- **top_k_features** (*int, optional*) – Top K importance features to consider if features_selected is an empty list. If top_k_features <= 0, all the features are selected for explanation. Defaults to -1.
- **feature_importance_method** (*str,optional*) – Importance feature method to be used if top_k_features is > 0 and features_selected is empty. Defaults to 'SHAP'.
- **max_dataset_size** (*int*) – maximum dataset size used during selected feature importance method (feature_importance_method). Defaults to 10.

explain_instance(*instance: Union[pandas.core.frame.DataFrame, numpy.ndarray], **kwargs*)
Produces local explanation of the target model for selected feature(s).

Parameters **instance** (*Union[pd.DataFrame, np.ndarray]*) – input instance to be explained.

Returns

explanation object Dictionary with feature_name, feature_value, ice_value, current_value for ICE explanation. Dictionary with gce_values, x_grid, y_pred, current_values for GCE explanation.

Return type dict

get_params (**argv*, ***kwargs*) → dict
Get parameters for the explainer.

set_params (**argv*, ***kwargs*)
Set parameters for the explainer.

1.3 Global White Box Explainers

1.3.1 ProfWeight Explainer

1.4 Directly Interpretable Unsupervised Explainers

1.4.1 Disentangled Inferred Prior Variational Autoencoder (DIPVAE) Explainer

class aix360.algorithms.dipvae.dipvae.**DIPVAEExplainer**(*model_args*, *dataset=None*, *net=None*, *cuda_available=None*)

DIPVAEExplainer can be used to visualize the changes in the latent space of Disentangled Inferred Prior-VAE or DIPVAE[3]. This model is a Variational Autoencoder[4] variant that leads to a disentangled latent space. This is achieved by matching the covariance of the prior distributions with the inferred prior.

References

Initialize DIPVAEExplainer explainer.

Parameters

- **model_args** – This should contain all the parameter required for the generative model training and inference. This includes model type (vae, dipvae-i, dipvae-ii, user-defined). The user-defined model can be passed to the parameter net of the fit() function. Each of the model should have encoder and decode function defined. See the notebook example for other model specific parameters.
- **dataset** – The dataset object.
- **net** – If not None this is the user specified generative model.
- **cuda_available** – If True use GPU.

explain (*input_images*, *edit_dim_id*, *edit_dim_value*, *edit_z_sample=False*)
Edits the images in the latent space and returns the generated images.

Parameters

[3] Variational Inference of Disentangled Latent Concepts from Unlabeled Observations (DIP-VAE), ICLR 2018. Kumar, Sattigeri, Balakrishnan.
[4] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. ICLR, 2014.

- **input_images** – The input images.
- **edit_dim_id** – The latent dimension id that need to be edited.
- **edit_dim_value** – The value that is assigned to the latent dimension with id edit_dim_id.
- **edit_z_sample** – If True will use the sample from encoder instead of the mean.

Returns Edited images.

fit (*visualize=False*, *save_dir='results'*)
Train the underlying generative model.

Parameters

- **visualize** – Plot reconstructions during fit.
- **save_dir** – directory where plots and model will be saved.

Returns elbo

set_params (**argv*, ***kwargs*)
Set parameters for the explainer.

1.4.2 Protodash Explainer

class aix360.algorithms.protodash.PDASH.**ProtodashExplainer**
ProtodashExplainer provides exemplar-based explanations for summarizing datasets as well as explaining predictions made by an AI model. It employs a fast gradient based algorithm to find prototypes along with their (non-negative) importance weights. The algorithm minimizes the maximum mean discrepancy metric and has constant factor approximation guarantees for this weakly submodular function.[5].

References

Constructor method, initializes the explainer

explain (*X*, *Y*, *m*, *kernelType='other'*, *sigma=2*, *optimizer='cvxpy'*)
Return prototypes for data X, Y.

Parameters

- **X** (*double 2d array*) – Dataset you want to explain.
- **Y** (*double 2d array*) – Dataset to select prototypical explanations from.
- **m** (*int*) – Number of prototypes
- **kernelType** (*str*) – Type of kernel (viz. 'Gaussian', / 'other')
- **sigma** (*double*) – width of kernel
- **optimizer** (*string*) – qpsolver ('cvxpy' or 'osqp')

Returns m selected prototypes from X and their (unnormalized) importance weights

set_params (**argv*, ***kwargs*)
Set parameters for the explainer.

[5] Karthik S. Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, "ProtoDash: Fast Interpretable Prototype Selection"

1.4.3 CoFrNet Explainer

class aix360.algorithms.cofrnet.CoFrNet.**CoFrNet_Explainer**(*cofrnet_model*)

explain(*explain_mode*, *max_layer_num=10*, *var_num=6*)
Provides Explanations of CoFrNet Model

Args: explain_mode: either "importances" or "print_co_fr", will raise exception if not one of these two options max_layer_num: For "print_co_fr": Choose Depth of Ladder to Show, Default 10 var_num: For "print_co_fr": Variable (index of input feature) for Which to Display Ladder, Default 6

set_params(**argv*, ***kwargs*)
Set parameters for the explainer.

1.5 Directly Interpretable Supervised Explainers

1.5.1 Boolean Rules via Column Generation Explainer

class aix360.algorithms.rbm.BRCG.**BRCGExplainer**(*model*)
Boolean Rule Column Generation explainer. Provides access to *aix360.algorithms.rbm.boolean_rule_cg.BooleanRuleCG*, which implements a directly interpretable supervised learning method for binary classification that learns a Boolean rule in disjunctive normal form (DNF) or conjunctive normal form (CNF) using column generation (CG). AIX360 implements a heuristic beam search version of BRCG that is less computationally intensive than the published integer programming version[4].

References

Initialize a BRCGExplainer object.

Parameters **model** – model to operate on, instance of *aix360.algorithms.rbm.boolean_rule_cg.BooleanRuleCG*

explain(**argv*, ***kwargs*)
Return rules comprising the underlying model.

Parameters

- **maxConj** (*int, optional*) – Maximum number of conjunctions to show
- **prec** (*int, optional*) – Number of decimal places to show for floating-value thresholds

Returns

Dictionary containing

- isCNF (bool): flag signaling whether model is CNF or DNF
- rules (list): selected conjunctions formatted as strings

fit(*X_train*, *Y_train*, **argv*, ***kwargs*)
Fit model to training data.

Parameters

[4] Michael Hind, Dennis Wei, Murray Campbell, Noel C. F. Codella, Amit Dhurandhar, Aleksandra Mojsilovic, Karthikeyan Natesan Ramamurthy, Kush R. Varshney, "TED: Teaching AI to Explain its Decisions," AAAI /ACM Conference on Artificial Intelligence, Ethics, and Society (AIES-19), 2019.

- **X_train** (*DataFrame*) – Binarized features with MultiIndex column labels
- **Y_train** (*array*) – Binary-valued target variable

Returns Self

Return type *BRCGExplainer*

predict(*X*, **argv*, ***kwargs*)
Predict class labels.

Parameters **X** (*DataFrame*) – Binarized features with MultiIndex column labels

Returns y – Predicted labels

Return type array

set_params(**argv*, ***kwargs*)
Set parameters for the explainer.

class aix360.algorithms.rbm.boolean_rule_cg.**BooleanRuleCG**(*lambda0=0.001*, *lambda1=0.001*, *CNF=False*, *iterMax=100*, *timeMax=100*, *K=10*, *D=10*, *B=5*, *eps=1e-06*, *solver='ECOS'*, *verbose=False*, *silent=False*)
BooleanRuleCG is a directly interpretable supervised learning method for binary classification that learns a Boolean rule in disjunctive normal form (DNF) or conjunctive normal form (CNF) using column generation (CG). AIX360 implements a heuristic beam search version of BRCG that is less computationally intensive than the published integer programming version **[#NeurIPS2018]_**.

References

Parameters

- **lambda0** (*float, optional*) – Complexity - fixed cost of each clause
- **lambda1** (*float, optional*) – Complexity - additional cost for each literal
- **CNF** (*bool, optional*) – CNF instead of DNF
- **iterMax** (*int, optional*) – Column generation - maximum number of iterations
- **timeMax** (*int, optional*) – Column generation - maximum runtime in seconds
- **K** (*int, optional*) – Column generation - maximum number of columns generated per iteration
- **D** (*int, optional*) – Column generation - maximum degree
- **B** (*int, optional*) – Column generation - beam search width
- **eps** (*float, optional*) – Numerical tolerance on comparisons
- **solver** (*str, optional*) – Linear programming - solver
- **verbose** (*bool, optional*) – Linear programming - verboseness
- **silent** (*bool, optional*) – Silence overall algorithm messages

compute_conjunctions(*X*)
Compute conjunctions of features as specified in self.z.

Parameters **X** (*DataFrame*) – Binarized features with MultiIndex column labels

Returns A – Conjunction values

Return type array

explain(*maxConj=None*, *prec=2*)
Return rules comprising the model.

Parameters

- **maxConj** (*int, optional*) – Maximum number of conjunctions to show
- **prec** (*int, optional*) – Number of decimal places to show for floating-value thresholds

Returns

Dictionary containing

- isCNF (bool): flag signaling whether model is CNF or DNF
- rules (list): selected conjunctions formatted as strings

fit(*X*, *y*)
Fit model to training data.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **y** (*array*) – Binary-valued target variable

Returns Self

Return type *BooleanRuleCG*

predict(*X*)
Predict class labels.

Parameters **X** (*DataFrame*) – Binarized features with MultiIndex column labels

Returns y – Predicted labels

Return type array

1.5.2 Generalized Linear Rule Model Explainer

class aix360.algorithms.rbm.GLRM.**GLRMExplainer**(*model*)
Generalized Linear Rule Model explainer. Provides access to the following directly interpretable supervised learning methods:

- **Linear Rule Regression:** linear regression on rule-based features[3].
- **Logistic Rule Regression:** logistic regression on rule-based features[3].

[3] D. Wei, S. Dash, T. Gao, O. Günlük, "Generalized linear rule models." International Conference on Machine Learning (ICML), 2019.

References

Initialize a GLRMExplainer object.

Parameters **model** – model to operate on. Instance of either

- *aix360.algorithms.rbm.linear_regression.LinearRuleRegression* or
- *aix360.algorithms.rbm.logistic_regression.LogisticRuleRegression*

explain(*maxCoeffs=None*, *highDegOnly=False*, *prec=2*)
Return DataFrame holding model features and their coefficients.

Parameters

- **maxCoeffs** (*int, optional*) – Maximum number of rules/numerical features to show
- **highDegOnly** (*bool, optional*) – Only show higher-degree rules
- **prec** (*int, optional*) – Number of decimal places to show for floating-value thresholds

Returns dfExpl – Rules/numerical features and their coefficients

Return type DataFrame

fit(*X_train*, *Y_train*, *Xstd=None*)
Fit model to training data.

Parameters

- **X_train** (*DataFrame*) – Binarized features with MultiIndex column labels
- **Y_train** (*array*) – Target variable
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns Self

Return type *GLRMExplainer*

predict(*X*, *Xstd=None*)
Predict responses.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns y – Predicted responses

Return type array

predict_proba(*X*, *Xstd=None*)
Predict probabilities of Y=1. Only available if underlying model implements *predict_proba* method.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns p – Predicted probabilities

Return type array

Raises `ValueError` – if model doesn't implement *predict_proba*

set_params(**argv*, ***kwargs*)
Set parameters for the explainer.

visualize(*Xorig*, *fb*, *features=None*)
Plot generalized additive model component, which includes first-degree rules and linear functions of unbinarized ordinal features but excludes higher-degree rules.

Parameters

- **Xorig** (*DataFrame*) – Original unbinarized features
- **fb** – FeatureBinarizer object used to binarize features
- **features** (*list, optional*) – Subset of features to be plotted

class aix360.algorithms.rbm.linear_regression.**LinearRuleRegression**(*lambda0=0.05*, *lambda1=0.01*, *useOrd=False*, *debias=True*, *K=1*, *iterMax=200*, *B=1*, *wLB=0.5*, *stopEarly=False*, *eps=1e-06*)
Linear Rule Regression is a directly interpretable supervised learning method that performs linear regression on rule-based features.

Parameters

- **lambda0** (*float, optional*) – Regularization - fixed cost of each rule
- **lambda1** (*float, optional*) – Regularization - additional cost of each literal in rule
- **useOrd** (*bool, optional*) – Also use standardized numerical features
- **debias** (*bool, optional*) – Re-fit final solution without regularization
- **K** (*int, optional*) – Column generation - maximum number of columns generated per iteration
- **iterMax** (*int, optional*) – Column generation - maximum number of iterations
- **B** (*int, optional*) – Column generation - beam search width
- **wLB** (*float, optional*) – Column generation - weight on lower bound in evaluating nodes
- **stopEarly** (*bool, optional*) – Column generation - stop after current degree once improving column found
- **eps** (*float, optional*) – Numerical tolerance on comparisons

compute_conjunctions(*X*)
Compute conjunctions of features as specified in self.z.

Parameters **X** (*DataFrame*) – Binarized features with MultiIndex column labels

Returns A – Feature conjunction values, shape (X.shape[0], self.z.shape[1])

Return type array

explain(*maxCoeffs=None*, *highDegOnly=False*, *prec=2*)
Return DataFrame holding model features and their coefficients.

Parameters

- **maxCoeffs** (*int, optional*) – Maximum number of rules/numerical features to show
- **highDegOnly** (*bool, optional*) – Only show higher-degree rules
- **prec** (*int, optional*) – Number of decimal places to show for floating-value thresholds

Returns dfExpl – Rules/numerical features and their coefficients

Return type DataFrame

fit(*X*, *y*, *Xstd=None*)
Fit model to training data.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **y** (*array*) – Target variable
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns Self

Return type *LinearRuleRegression*

predict(*X*, *Xstd=None*)
Predict responses.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns yhat – Predicted responses

Return type array

visualize(*Xorig*, *fb*, *features=None*)
Plot generalized additive model component, which includes first-degree rules and linear functions of unbinarized ordinal features but excludes higher-degree rules.

Parameters

- **Xorig** (*DataFrame*) – Original unbinarized features
- **fb** – FeatureBinarizer object used to binarize features
- **features** (*list, optional*) – Subset of features to be plotted

class aix360.algorithms.rbm.logistic_regression.**LogisticRuleRegression**(*lambda0=0.05*, *lambda1=0.01*, *useOrd=False*, *debias=True*, *init0=False*, *K=1*, *iterMax=200*, *B=1*, *wLB=0.5*, *stopEarly=False*, *eps=1e-06*, *maxSolverIter=100*)

Logistic Rule Regression is a directly interpretable supervised learning method that performs logistic regression on rule-based features.

Parameters

- **lambda0** (*float, optional*) – Regularization - fixed cost of each rule
- **lambda1** (*float, optional*) – Regularization - additional cost of each literal in rule
- **useOrd** (*bool, optional*) – Also use standardized numerical features
- **debias** (*bool, optional*) – Re-fit final solution without regularization
- **init0** (*bool, optional*) – Initialize with no features
- **K** (*int, optional*) – Column generation - maximum number of columns generated per iteration
- **iterMax** (*int, optional*) – Column generation - maximum number of iterations
- **B** (*int, optional*) – Column generation - beam search width
- **wLB** (*float, optional*) – Column generation - weight on lower bound in evaluating nodes
- **stopEarly** (*bool, optional*) – Column generation - stop after current degree once improving column found
- **eps** (*float, optional*) – Numerical tolerance on comparisons
- **maxSolverIter** – Maximum number of logistic regression solver iterations

compute_conjunctions(*X*)

Compute conjunctions of features as specified in self.z.

Parameters **X** (*DataFrame*) – Binarized features with MultiIndex column labels

Returns A – Feature conjunction values, shape (X.shape[0], self.z.shape[1])

Return type array

explain(*maxCoeffs=None*, *highDegOnly=False*, *prec=2*)

Return DataFrame holding model features and their coefficients.

Parameters

- **maxCoeffs** (*int, optional*) – Maximum number of rules/numerical features to show
- **highDegOnly** (*bool, optional*) – Only show higher-degree rules
- **prec** (*int, optional*) – Number of decimal places to show for floating-value thresholds

Returns dfExpl – Rules/numerical features and their coefficients

Return type DataFrame

fit(*X*, *y*, *Xstd=None*)
Fit model to training data.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **y** (*array*) – Target variable
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns Self

Return type *LogisticRuleRegression*

predict(*X*, *Xstd=None*)
Predict class labels.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns yhat – Predicted labels

Return type array

predict_proba(*X*, *Xstd=None*)
Predict probabilities of Y=1.

Parameters

- **X** (*DataFrame*) – Binarized features with MultiIndex column labels
- **Xstd** (*DataFrame, optional*) – Standardized numerical features

Returns p – Predicted probabilities

Return type array

visualize(*Xorig*, *fb*, *features=None*)
Plot generalized additive model component, which includes first-degree rules and linear functions of unbinarized ordinal features but excludes higher-degree rules.

Parameters

- **Xorig** (*DataFrame*) – Original unbinarized features
- **fb** – FeatureBinarizer object used to binarize features
- **features** (*list, optional*) – Subset of features to be plotted

1.5.3 Teaching Explanations for Decisions (TED) Cartesian Product Explainer

class aix360.algorithms.ted.TED_Cartesian.**TED_CartesianExplainer**(*model*)

TED is an explainability framework that leverages domain-relevant explanations in the training dataset to predict both labels and explanations for new instances [#]_. This is an implementation of the simplest instantiation of TED, called the Cartesian Product.

References

Parameters **model** (*sklearn.base.BaseEstimator*) – a binary estimator for classification, i.e., it implements fit and predict.

explain(*X*)

Use TED-enhanced classifier to provide an explanation (E) for passed instance

Parameters **X** (*list of ints*) – features

Returns predicted explanation [0..MaxE]

Return type int

fit(*X*, *Y*, *E*)

Train a classifier based on features (X), labels (Y), and explanations (E)

Parameters

- **X** – list of features vectors
- **Y** – list of labels
- **E** – list of explanations

predict(*X*)

Use TED-enhanced classifier to provide an prediction (Y) for passed instance

Parameters **X** (*list of ints*) – features

Returns predicted label {0,1}

Return type int

predict_explain(*X*)

Use TED-enhanced classifier to predict label (Y) and explanation (E) for passed instance

Parameters **X** (*list of ints*) – features

Returns

- **Y** (*int*) – predicted label {0,1}
- **E** (*int*) – predicted explanation [0..MaxE]

Return type tuple

score(*X_test*, *Y_test*, *E_test*)

Evaluate the accuracy (Y and E) of the TED-enhanced classifier using a test dataset

Parameters

- **X_test** (*list of lists*) – list of feature vectors
- **Y_test** (*list of int*) – list of labels {0, 1}
- **E_test** (*list of ints*) – list of explanations {0, . . . , NumExplanations -1}

Returns

- **YE_accuracy** – the accuracy of predictions when the labels (Y) and explanations (E) are treated as a combined label
- **Y_accuracy** – the prediction accuracy for labels (Y)
- **E_accuracy** – the prediction accuracy of explanations (E)

Return type tuple

set_params(**argv*, ***kwargs*)
Set parameters for the explainer.

1.5.4 RIPPER Explainer

class aix360.algorithms.rule_induction.ripper.**RipperExplainer**(*d: int = 64*, *k: int = 2*, *pruning_threshold: int = 20*, *random_state: int = 0*)
RIPPER (Repeated Incremental Pruning to Produce Error Reduction) is a heuristic rule induction algorithm based on separate-and-conquer. The explainer outputs a rule set in Disjunctive Normal Form (DNF) for a single target concept.

References

Parameters

- **d** (*int*) – The number of bits that a new rule need to gain. Defaults to 64.
- **k** (*int*) – The number of iterations for the optimization loop. Defaults to 2.
- **pruning_threshold** (*int*) – The minimum number of instances for splitting. Defaults to 20.
- **random_state** (*int*) – The random seed for the splitting function. Defaults to 0.

explain()
Export rule set to technical interchange format trxf from internal representation for the positive value (i.e. label value) it has been fitted for.

When the internal rule set is empty an empty dnf rule set with the internal pos value is returned.

Returns trxf.DnfRuleSet

explain_multiclass()
Export rules to technical interchange format trxf from internal representation Returns a list of rule sets.

Returns – Ordered list of rulesets

Return type list(trxf.DnfRuleSet)

fit(*train: pandas.core.frame.DataFrame*, *y: pandas.core.series.Series*, *target_label=None*)
The fit function for RIPPER algorithm. Its implementation is limited to DataFrame and Series because the RIPPER algorithm needs the information of feature name and have to support nominal data type. Only *float* dtypes are considered numerical features. All others (including int) are treated as nominal.

If target_label is specified, binary classification is assumed and asserted, and training uses target_label as selection of positive examples.

The induction of rules is deterministic by default as all random choices are initialized with self.random_state, which is 0 by default.

Parameters

- **train** (*pd.DataFrame*) – The features of the training set
- **y** (*pd.Series*) – The labels of the training set
- **target_label** (*Any*) – The target label to learn for binary classification, among the unique values of y. If not provided, Ripper will induce a native ordered ruleset with multiple labels/conclusions.

Returns self

predict (*X: pandas.core.frame.DataFrame*) → numpy.ndarray
The predict function for RIPPER algorithm. Its implementation is limited to DataFrame and Series because the RIPPER algorithm needs the information of feature name and have to support nominal data type

Parameters **X** (*pd.DataFrame*) – DataFrame of features

Returns predicted labels

Return type np.array

set_params (***kwargs*)
Set parameters for the explainer.

target_label
The latest positive value RIPPER has been fitted for.

1.5.5 IMD Explainer

class aix360.algorithms.imd.imd.**IMDExplainer**
Interpretable Model Differencing to explain the similarities and differences between two classifiers. Provides access to `aix360.algorithms.imd.jst.JointSurrogateTree`, a novel data structure to compactly represent the differences between the models in terms of rules, and also provides a way to visualize the joint surrogate tree structure.

References

Initialize an IMDExplainer object.

explain (**argv*, ***kwargs*)
Return diff-rules.

fit (*X_train: pandas.core.frame.DataFrame*, *Y1*, *Y2*, *max_depth*, *split_criterion=1*, *alpha=0.0*, *verbose=True*, ***kwargs*)
Fit joint surrogate tree to input data, and outputs from two models. :param X_train: input dataframe :param Y1: model1 outputs :param Y2: model2 outputs :param max_depth: maximum depth of the joint surrogate tree to be built :param feature_names: list of input feature names :param alpha: parameter to control degree of favouring common nodes vs. separate nodes :param split_criterion: which divergence criterion to use? (see paper for more details) :param verbose: :param **kwargs:

Returns self

metrics (*x_test: pandas.core.frame.DataFrame*, *y_test1*, *y_test2*, *name='test'*)
take x_test and check the precision and recall precision =

number of actual diff samples inside the diffregion / number of test samples inside the diffregion

recall = diff samples inside the region / total number of diff samples

Parameters

- **x_test** – test data (only x) to compute diff-based metrics
- **name** – string (*train* or *test*)

Returns a dictionary having *precision*, *recall*, *num-rules*, and *num-unique-preds* values as obtained from the diff-rules extracted from the jst.

predict (*X*, **argv*, ***kwargs*)
Predict diff-labels.

set_params (**argv*, ***kwargs*)
Set parameters for the explainer.

1.6 Time Series based Local Black Box Explainers

1.6.1 Time Series Individual Conditional Expectation (TSICE) Explainer

class aix360.algorithms.tsice.tsice.**TSICEExplainer**(*forecaster: Callable*, *input_length: int*, *forecast_lookahead: int*, *n_variables: int = 1*, *n_exogs: int = 0*, *n_perturbations: int = 25*, *features_to_analyze: List[str] = None*, *perturbers: List[Union[aix360.algorithms.tsutils.tsperturbers.tsperturb dict]] = None*, *explanation_window_start: int = None*, *explanation_window_length: int = 10*)

TSICEExplainer extends the Individual Conditional Expectation for correlated timeseries data (higher dimensions). It uses TSFeatures to derive time series structural features, and uses data perturber(TSPerturber) for generating simulated data. TSICEExplainer explains the trend in the model forecast change with time series derived features.

References

Initializer for TSICEExplainer

Parameters

- **forecaster** (*Callable*) – Callable object produces a forecast as numpy array for a given input as numpy array.
- **input_length** (*int*) – Input length for the forecaster.
- **forecast_lookahead** (*int*) – Lookahead length of the forecaster prediction.
- **n_variables** (*int*) – Number of variables in the forecaster input. Defaults to 1.
- **n_exogs** (*int*) – Number of exogenous variable required for the forecaster. Defaults to 0.
- **n_perturbations** (*int*) – Number of perturbed instance for TSExplanation. Defaults to 25.

- **features_to_analyze** (*List[str]*) – List of features used to analyze the perturbed timeseries during TSICE explanation. As the observing timeseries is complicated, these set of features can be used to observe the perturbations closer. Allowed values are "median", "mean", "min", "max", "std", "range", "intercept", "trend", "rsquared", "max_variation". If None, "mean" is used by default. Defaults to None.
- **perturbers** (*List[TSPerturber, dict]*) – data perturbation algorithm specification by TSPerturber instance or dict. Allowed values for "type" key in dictionary are block-bootstrap, frequency, moving-average, shift. Block-bootstrap split the time series into contiguous chunks called blocks, for each block noise is estimated and noise is exchanged and added to the signal between randomly selected blocks. Moving-average perturbation maintains the moving mean of the time series data with the specified window length, but add perturbed noise with similar distribution as the data. Frequency perturber performs FFT on the noise, and removes random high frequency components from the noise estimates. Number of frequencies to be removed is specified by the truncate_frequencies argument. Shift perturber adds random upward or downward shift in the data value over time continuous blocks. If not provided default perturber is combination of block-bootstrap, moving-average, and frequency. Defaults to None.
- **explanation_window_start** (*int*) – Explanation window is selected from the input timeseries starting explanation_window_start. This window is used to select the part of the timeseries for TSICE analysis. Perturbations are computed over this explanation window. If explanation_window_start is None, explanation_window is selected from recent upto explanation_window_length. Defaults to None.
- **explanation_window_length** (*int*) – Explanation window of length: explanation_window_length is selected from the input timeseries. This window is used to select the part of the timeseries for TSICE analysis. Perturbations are computed over this explanation window. Defaults to 10.

explain_instance(*ts: aix360.algorithms.tsutils.tsframe.tsFrame*, *ts_related: aix360.algorithms.tsutils.tsframe.tsFrame = None*, ***explain_params*)

Explain the forecast made by the forecaster at a certain point in time (**local explanation**).

Parameters

- **ts** (*tsFrame*) – The future univariate time series `tsFrame` to use for forecasting that extends forward from the end of training `tsFrame` (`ts_train`) or `timestamp_start` for the requested number of periods. This can be generated using `aix360.algorithms.tsframe.tsFrame`. A `tsFrame` is a pandas `DataFrame` indexed by `Timestamp` objects (that is `DatetimeIndex`). Each column corresponds to a target to forecast.
- **ts_related** (*tsFrame, optional*) – The related time series `tsFrame` containing the external regressors. A `tsFrame` is a pandas `DataFrame` indexed by `Timestamp` objects (that is `DatetimeIndex`). Each column corresponds to a related external regressor. Defaults to None.
- **explain_params** – Arbitrary explainer parameters.

Returns

explanation object Dictionary with data_x, feature_names, feature_values, signed_impact, total_impact, current_forecast, current_feature_values, perturbations and forecasts_on_perturbations.

Return type dict

set_params(**argv*, ***kwargs*)

Set parameters for the explainer.

1.6.2 Time Series Saliency (TSSaliency) Explainer

class aix360.algorithms.tssaliency.tssaliency.**TSSaliencyExplainer**(*model: Callable, input_length: int, feature_names: List[str], base_value: List[float] = None, n_samples: int = 50, gradient_samples: int = 25, gradient_function: Callable = None, random_seed: int = 22*)

Time Series Saliency (TSSaliency) Explainer is a model agnostic saliency explainer for time series associate tasks. The TSSaliency supports univariate and multivariate use cases. It explains temporal importance of different variates on the model prediction. TSSaliency incorporates an integrated gradient method for saliency estimation. The saliency measure involves the notion of a base value. For example, the base value can be the constant signal with average value. The saliency measure is computed by integrating the model sensitivity over a trajectory from the base value to the time series signal. The TSSaliency explainer provides variate wise contributions to model prediction at a temporal resolution.

References

Initializer for TSSaliencyExplainer

Parameters

- **model** (*Callable*) – Callable object produces a prediction as numpy array for a given input as numpy array. It can be a model prediction (predict/ predict_proba) function that results a real value like probability or regressed value. This function must accept numpy array of shape (input_length x len(feature_names)) as input and result in numpy array of shape (1, -1). Currently, TSSaliency supports sinlge output models only. For multi-output models, you can aggregate the output using a custom model_wrapper. Use model wrapper classes from aix360.algorithms.tsutils.model_wrappers.
- **input_length** (*int*) – length of history window used in model training.
- **feature_names** (*List[str]*) – list of feature names in the input data.
- **base_value** (*List[float]*) – base value to be used in saliency computation. The computed gradients are with respect to this base value. If None, mean value is used. Defaults to None.
- **n_samples** (*int*) – number of path samples to be created for each input instance while computing saliency metric. Defaults to 50.

- **gradient_samples** (*int*) – number of timeseries samples to be generated while computing integreated gradient on the input data. Defaults to 25.
- **gradient_function** (*Callable*) – gradient function to be used in saliency (integrated gradient) computation. If None, mc_gradient_compute is used. Defaults to None.
- **random_seed** (*int*) – random seed to get consistent results. Refer to numpy random state. Defaults to 22.

explain_instance(*ts: aix360.algorithms.tsutils.tsframe.tsFrame*, ***explain_params*)
Explain the prediction made by the time series model at a certain point in time (**local explanation**).

Parameters

- **ts** (*tsFrame*) – Input time series signal in `tsFrame` format. This can be generated using `aix360.algorithms.tsframe.tsFrame`. A `tsFrame` is a pandas `DataFrame` indexed by `Timestamp` objects (that is `DatetimeIndex`). Each column corresponds to an input feature.
- **explain_params** – Arbitrary explainer parameters.

Returns

explanation object Dictionary with input_data, saliency, feature_names, timestamps, base_value, instance_prediction, base_value_prediction.

Return type dict

get_params(**argv*, ***kwargs*) → dict
Get parameters for the explainer.

set_params(**argv*, ***kwargs*)
Set parameters for the explainer.

1.6.3 Time Series Local Interpretable Model-agnostic Explainer (TSLime)

class aix360.algorithms.tslime.tslime.**TSLimeExplainer**(*model: Callable*, *input_length: int*, *n_perturbations: int = 2000*, *relevant_history: int = None*, *perturbers: List[Union[aix360.algorithms.tsutils.tsperturbers.tspe dict]] = None*, *local_interpretable_model: aix360.algorithms.tslime.surrogate.LinearSurrogateM = None*, *random_seed: int = None*)
Time Series Local Interpretable Model-agnostic Explainer (TSLime) is a model-agnostic local time series explainer. LIME (Locally interpretable Model agnostic explainer) is a popular algorithm for local explanation. LIME explains the model behavior by approximating the model response with linear models. LIME algorithm specifically assumes tabular data format, where each row is a data point, and columns are features. A generalization of LIME algorithm for image data uses super pixel based perturbation. TSLIME generalizes LIME algorithm for time series context.

TSLIME uses time series perturbation methods to produce a local input perturbation, and linear model surrogate which best approximates the model response. TSLime produces an interpretable explanation. The explanation weights produced by the TSLime explanation indicates model local sensitivity.

References

Initializer for TSLimeExplainer

Parameters

- **model** (*Callable*) – Callable object produces a prediction as numpy array for a given input as numpy array. It can be a model prediction (predict/ predict_proba) function that results a real value like probability or regressed value. This function must accept numpy array of shape (input_length x len(feature_names)) as input and result in numpy array of shape (1, -1). Currently, TSLime supports sinlge output models only. For multi-output models, you can aggregate the output using a custom model_wrapper. Use model wrapper classes from aix360.algorithms.tsutils.model_wrappers.
- **input_length** (*int*) – Input (history) length used for input model.
- **n_perturbations** (*int*) – Number of perturbed instance for TSExplanation. Defaults to 25.
- **relevant_history** (*int*) – Interested window size for explanations. The explanation is computed for selected latest window of length *relevant_history*. If *input_length=20* and *relevant_history=10*, explanation is computed for last 10 time points. If None, relevant_history is set to input_length. Defaults to None.
- **perturbers** (*List[TSPerturber, dict]*) – data perturbation algorithm specification by TSPerturber instance or dict. Allowed values for "type" key in dictionary are block-bootstrap, frequency, moving-average, shift. Block-bootstrap split the time series into contiguous chunks called blocks, for each block noise is estimated and noise is exchanged and added to the signal between randomly selected blocks. Moving-average perturbation maintains the moving mean of the time series data with the specified window length, but add perturbed noise with similar distribution as the data. Frequency perturber performs FFT on the noise, and removes random high frequency components from the noise estimates. Number of frequencies to be removed is specified by the truncate_frequencies argument. Shift perturber adds random upward or downward shift in the data value over time continuous blocks. If not provided default perturber is block-bootstrap. Defaults to None.
- **local_interpretable_model** (*LinearSurrogateModel*) – Local interpretable model, a surrogate that is to be trained on the given input time series neighborhood. This model is used to provide local weights for each time point in the selected timeseries. If None, sklearn's Linear Regression model, aix360.algorithms.tslime.surrogate.LinearRegressionSurrogate is used. Defaults to None.
- **random_seed** (*int*) – random seed to get consistent results. Refer to numpy random state. Defaults to None.

explain_instance(*ts: aix360.algorithms.tsutils.tsframe.tsFrame*, ***explain_params*)

Explain the prediction made by the time series model at a certain point in time (**local explanation**).

Parameters

- **ts** (*tsFrame*) – Input time series signal in `tsFrame` format. This can be generated using `aix360.algorithms.tsframe.tsFrame`. A `tsFrame` is a pandas `DataFrame` indexed by `Timestamp` objects (that is `DatetimeIndex`). Each column corresponds to an input feature.
- **explain_params** – Arbitrary explainer parameters.

Returns

explanation object Dictionary with keys: input_data, history_weights, model_prediction, surrogate_prediction, x_perturbations, y_perturbations.

Return type dict

set_params(**argv*, ***kwargs*)
Set parameters for the explainer.

CHAPTER 2

Metrics

aix360.metrics.local_metrics.**faithfulness_metric**(*model*, *x*, *coefs*, *base*)

This metric evaluates the correlation between the importance assigned by the interpretability algorithm to attributes and the effect of each of the attributes on the performance of the predictive model. The higher the importance, the higher should be the effect, and vice versa, The metric evaluates this by incrementally removing each of the attributes deemed important by the interpretability metric, and evaluating the effect on the performance, and then calculating the correlation between the weights (importance) of the attributes and corresponding model performance.[1]

References

Parameters

- **model** – Trained classifier, such as a ScikitClassifier that implements a predict() and a predict_proba() methods.
- **x** (*numpy.ndarray*) – row of data.
- **coefs** (*numpy.ndarray*) – coefficients (weights) corresponding to attribute importance.
- **base** (*(numpy.ndarray*) – base (default) values of attributes

Returns correlation between attribute importance weights and corresponding effect on classifier.

Return type float

aix360.metrics.local_metrics.**monotonicity_metric**(*model*, *x*, *coefs*, *base*)

This metric measures the effect of individual features on model performance by evaluating the effect on model performance of incrementally adding each attribute in order of increasing importance. As each feature is added, the performance of the model should correspondingly increase, thereby resulting in monotonically increasing model performance.[2]

[1] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 7775-7784. 2018.

[2] Ronny Luss, Pin-Yu Chen, Amit Dhurandhar, Prasanna Sattigeri, Karthikeyan Shanmugam, and Chun-Chen Tu. Generating Contrastive Explanations with Monotonic Attribute Functions. CoRR abs/1905.13565. 2019.

References

Parameters

- **model** – Trained classifier, such as a ScikitClassifier that implements a predict() and a predict_proba() methods.
- **x** (*numpy.ndarray*) – row of data.
- **coefs** (*numpy.ndarray*) – coefficients (weights) corresponding to attribute importance.
- **base** (*(numpy.ndarray*) – base (default) values of attributes

Returns True if the relationship is monotonic.

Return type bool

CHAPTER 3

Datasets

3.1 CDC Dataset

class aix360.datasets.**CDCDataset**(*custom_preprocessing=<function default_preprocessing>*, *dirpath=None*)

The CDC (Center for Disease Control and Prevention) questionnaire datasets[5] are surveys conducted by the organization involving 1000s of civilians about various facets of daily life. There are 44 questionnaires that collect data about income, occupation, health, early childhood and many other behavioral and lifestyle aspects of people living in the US. These questionnaires are thus a rich source of information indicative of the quality of life of many civilians. More information about each questionaire and the type of answers are available in the following reference.

References

3.2 CelebA Dataset

class aix360.datasets.**CelebADataset**(*dirpath=None*)

Images are based on the CelebA Dataset[6][7]. Specifically, we use a GAN developed by Karras et. al[8] in order to generate new images similar to CelebA. We use these generated images in order to also store the latent variables used to generate them, which are required for generating pertinent negatives in CEM-MAF[9].

[5] NHANES 2013-2014 Questionnaire Data
[6] Liu, Luo, Wang, Tang. Large-scale CelebFaces Attributes (CelebA) Dataset.
[7] Liu, Luo, Wang, Tang. Deep Learning Face Attributes in the Wild. ICCV. 2015.
[8] Karras, Aila, Laine, Lehtinen Progressive Growing of GANs for Improved Quality, Stability, and Variation. ICLR. 2018.
[9] Luss, Chen, Dhurandhar, Sattigeri, Shanmugam, Tu. Generating Contrastive Explanations with Monotonic Attribute Functions. 2019.

References

3.3 CIFAR Dataset

class aix360.datasets.**CIFARDataset**(*dirpath=None*)

The CIFAR-10 dataset[10] consists of 60000 32x32 color images. Target variable is one amongst 10 classes. The dataset has 6000 images per class. There are 50000 training images and 10000 test images. The classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship ,truck. We further divide the training set into train1 (30000 samples) and train2 (20,000 samples). For ProfWt, the complex model is trained on train1 while the simple model is trained on train2.

References

3.4 Fashion MNIST Dataset

class aix360.datasets.**FMnistDataset**(*batch_size=256*, *subset_size=50000*, *test_batch_size=256*, *dirpath=None*)

Fashion-MNIST[11] is a large-scale image dataset of various fashion items (T-shirt/top, Trouser, Pullover, Dress. Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. The images are grayscale and 28x28 in size with each image belong to one the above mentioned 10 categories. The training set contains 60000 examples and the test set contains 10000 examples.

References

3.5 HELOC Dataset

class aix360.datasets.**HELOCDataset**(*custom_preprocessing=<function default_preprocessing>*, *dirpath=None*)

HELOC Dataset.

The FICO HELOC dataset[12] contains anonymized information about home equity line of credit (HELOC) applications made by real homeowners. A HELOC is a line of credit typically offered by a US bank as a percentage of home equity (the difference between the current market value of a home and the outstanding balance of all liens, e.g. mortgages). The customers in this dataset have requested a credit line in the range of USD 5,000 - 150,000.

The target variable in this dataset is a binary variable called RiskPerformance. The value "Bad" indicates that an applicant was 90 days past due or worse at least once over a period of 24 months from when the credit account was opened. The value "Good" indicates that they have made their payments without ever being more than 90 days overdue.

This dataset can be used to train a machine learning model to predict whether the homeowner qualifies for a line of credit or not. The HELOC dataset and more information about it, including instructions to download are available in the reference below.

[10] Krizhevsky, Hinton. Learning multiple layers of features from tiny images. Technical Report, University of Toronto 1 (4), 7. 2009
[11] Xiao, Han, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.
[12] Explainable Machine Learning Challenge - FICO Community.

References

3.6 MEPS Dataset

class aix360.datasets.**MEPSDataset**(*custom_preprocessing=<function default_preprocessing>*, *dirpath=None*)

The Medical Expenditure Panel Survey (MEPS)[13] data consists of large scale surveys of families and individuals, medical providers, and employers, and collects data on health services used, costs & frequency of services, demographics, health status and conditions, etc., of the respondents.

This specific dataset contains MEPS survey data for calendar year 2015 obtained in rounds 3, 4, and 5 of Panel 19, and rounds 1, 2, and 3 of Panel 20. See `aix360/data/meps_data/README.md` for more details on the dataset and instructions on downloading/processing the data.

References

3.7 TED Dataset

class aix360.datasets.**TEDDataset**(*dirpath=None*)

The goal of this synthetic dataset is to predict employee attrition from a fictious company. The dataset is generated by a python file called `GenerateData.py` in the `aix360/data/ted_data/` directory.

Like most datasets, each instance consists of a feature vector and a Y label, which represents whether the employee associated with the feature vector will leave the company. However, unlike most datasets, each instance will also have an Explanation (E). This is motivated by the TED framework, which requires explanations in its training data, but can be used by other explainability algorithms as a metric for explainability.

See also:

- AIES'19 paper by Hind et al.[14] for more information on the TED framework.
- The tutorial notebook `TED_Cartesian_test.ipynb` for information about how to use this dataset and the TED framework.
- `GenerateData.py` for more information on how the dataset is generated or to created a tailored version of the dataset.

References

load_file(*fileName='Retention.csv'*)

Open dataset file and populate X, Y, and E

Parameters **fileName** (*String*) – filename of dataset, a structured (CSV) dataset where

- The first N-2 columns are the features (X).
- The next to last column is the label (Y) {0, 1}
- The last column gives the explanations (E) {0, 1, . . . , MaxE}. We assume the explanation space is dense, i.e., if there are MaxE+1 unique explanations, they will be given IDs from 0 .. MaxE

[13] Medical Expenditure Panel Survey data

[14] Michael Hind, Dennis Wei, Murray Campbell, Noel C. F. Codella, Amit Dhurandhar, Aleksandra Mojsilovic, Karthikeyan Natesan Ramamurthy, Kush R. Varshney, "TED: Teaching AI to Explain its Decisions," AAAI /ACM Conference on Artificial Intelligence, Ethics, and Society (AIES-19), 2019.

- first row contains header information for each column and should be "Y" for labels and "E" for explanations
- each row is an instance

Returns

- **X** – list of features vectors
- **Y** – list of labels
- **E** – list of explanations

Return type tuple

3.8 eSNLI Dataset

class aix360.datasets.**eSNLIDataset**

The e-SNLI dataset[15] contains pairs of sentences each accompanied by human-rationale annotations as to which words are in each pairs are most important for matching.

The sentence pairs are from the Stanford Natural Language Inference dataset with labels that indicate if the sentence pair is a logical entailment, contradiction or neutral.

References

get_example(*example_id: str*) → Dict

Return an e-SNLI example.

The example_id indexes the "docs.jsonl" file of the downloaded dataset.

Parameters **example_id** (*str*) – the example index.

Returns e-SNLI example in dictionary form.

[15] Camburu, Oana-Maria, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom, "E-SNLI: Natural Language Inference with Natural Language Explanations.", 2018

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

P

R

S

T

V