aix360

Release 0.1

Oct 18, 2019

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

class aix360.algorithms.contrastive.CEM.**CEMExplainer**(*model*)

CEMExplainer can be used to compute contrastive explanations for image and tabular data. This is achieved by finding what is minimally sufficient (PP - Pertinent Positive) and what should be necessarily absent (PN - Pertinent Negative) to maintain the original classification. We use elastic norm regularization to ensure minimality for both parts of the explanation i.e. PPs and PNs. An autoencoder can optionally be used to make the explanations more realistic.[1]

References

Constructor method, initializes the explainer

Parameters **model** – KerasClassifier model whose predictions needs to be explained

explain_instance(*input_X*, *arg_mode*, *AE_model*, *arg_kappa*, *arg_b*, *arg_max_iter*, *arg_init_const*, *arg_beta*, *arg_gamma*)

Explains an input instance input_X and returns contrastive explanations. Note that this assumes that the classifier was trained with inputs normalized in [-0.5,0.5] range.

Parameters

- **input_X** (*numpy.ndarray*) – input instance to be explained
- **arg_mode** (*str*) – 'PP' or 'PN'
- **AE_model** – Auto-encoder model
- **arg_kappa** (*double*) – Confidence gap between desired class and other classes

[1] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, Payel Das, "Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives," Advances in Neural Information Processing Systems (NeurIPS), 2018.

- **arg_b** (*double*) – Number of different weightings of loss function to try
- **arg_max_iter** (*int*) – For each weighting of loss function number of iterations to search
- **arg_init_const** (*double*) – Initial weighting of loss function
- **arg_beta** (*double*) – Weighting of L1 loss
- **arg_gamma** (*double*) – Weighting of auto-encoder

Returns

- **adv_X** (*numpy ndarray*) – Perturbed input instance for PP/PN
- **delta_X** (*numpy ndarray*) – Difference between input and Perturbed instance
- **INFO** (*str*) – Other information about PP/PN

Return type tuple

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

class aix360.algorithms.contrastive.CEM_MAF.**CEM_MAFImageExplainer**(*model*, *attributes*, *aix360_path*)
CEM_MAFImageExplainer is a Contrastive Image explainer that leverages Monotonic Attribute Functions. The main idea here is to explain images using high level semantically meaningful attributes that may either be directly available or learned through supervised or unsupervised methods.[2]

References

Initialize image explainer.

Currently accepting model input which is an ImageClassifier.

check_attributes_celebA(*attributes*, *x*, *y*)
Load attribute classifiers and check which attributes in original image x are modified in adversarial image y

Parameters

- **attributes** (*str list*) – list of attributes to load attribute classifiers for
- **x** (*numpy.ndarray*) – original image
- **y** (*numpy.ndarray*) – adversarial image

Returns string detailing which attributes were added to (or removed from) x resulting in y

Return type str

explain_instance(*sess*, *input_img*, *input_latent*, *arg_mode*, *arg_kappa*, *arg_binary_search_steps*, *arg_max_iterations*, *arg_initial_const*, *arg_gamma*, *arg_beta*, *arg_attr_reg=1*, *arg_attr_penalty_reg=1*, *arg_latent_square_loss_reg=1*)
Explains an input instance input_image e.g. celebA is shape (1, 224, 224, 3)

Hard coded batch_size=1, assuming we provide explanation for 1 input_image at a time. Returns either pertinent positive or pertinent depending on parameter.

Parameters

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

- **sess** (*tensorflow.python.client.session.Session*) – Tensorflow session
- **input_img** (*numpy.ndarray*) – image to be explained, of shape (1, size, size, channels)
- **input_latent** (*numpy.ndarray*) – image to be explained, of shape (1, size, size, channels) in the latent space
- **arg_mode** (*str*) – "PN" for pertinent negative or "PP" for pertinent positive
- **arg_kappa** (*float*) – Confidence parameter that controls difference between prediction of PN (or PP) and original prediction
- **arg_binary_search_steps** (*int*) – Controls number of random restarts to find best PN or PP
- **arg_max_iterations** (*int*) – Max number iterations to run some version of gradient descent on PN or PP optimization problem from a single random initialization, i.e., total number of iterations wll be arg_binary_search_steps * arg_max_iterations
- **arg_initial_const** (*int*) – Constant used for upper/lower bounds in binary search
- **arg_gamma** (*float*) – Penalty parameter encouraging addition of attributes for PN or PP
- **arg_beta** (*float*) – Penalty parameter encourages minimal addition of attributes to PN or sparsity of the mask that generates the PP
- **arg_attr_reg** (*float*) – Penalty parameter on regularization of PN to be predicted different from original image
- **arg_attr_penalty_reg** (*float*) – Penalty regularizing PN from being too different from original image
- **arg_latent_square_loss_reg** (*float*) – Penalty regularizing PN from being too different from original image in the latent space

Returns

- **adv_img** (*numpy.ndarray*) – the pertinent positive or the pertinent negative image
- **attr_mod** (*str*) – only for PN; a string detailing which attributes were modified from the original image
- **INFO** (*str*) – only for PN; a string of information about original vs PN class and original vs PN prediction probability

Return type tuple

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

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.

1.2 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.

1.3 Global White Box Explainers

1.3.1 ProfWeight Explainer

class aix360.algorithms.profwt.profwt.**ProfweightExplainer**
The main explainer class that implements functions for computation of sample-wise weights from probe confidences of different layers from the complex model and then retrains the simple model with those weights. Implements the technique in the following reference:[1].

References

Initialize ProfweightExplainer.

[1] Dhurandhar, Shanmugam, Luss, Olsen. Improving Simple Models with Confidence Profiles. NeurIPS 2018

explain(*x_train*, *y_train*, *x_test*, *y_test*, *simple_model*, *hps*, *list_probe_filenames*, *start_layer*, *end_layer*, *model_type='neural_keras'*)

Obtains (train) sample-wise weights from stored probe confidences for different layers of the complex moddel. Retrains a simple model using the corresponding weighted training set.

Parameters

- **x_train** (*numpy array*) – Dataset of features to retrain the simple model using weights derived from probe confidences. Dimensions (num of samples x feature dimensions)
- **y_train** (*numpy array*) – Labels for the dataset that is used to retrain simple model using weights. Dimensions (num of samples x num of classes)
- **x_test** (*numpy array*) – Test dataset to evaluate the simple model trained using weights. Dimensions (num of samples x feature dimensions)
- **y_test** (*numpy array*) – Test labels to evaulate the simple model trained using weights. Dimensions (num of samples x num of classes)
- **hps** (*namedtuple*) – Hyperparameters (usually a named tuple with entries) to train the simple model. Please see fit function to find out the required set of parameters expected by the fit function.
- **list_probe_filenames** (*list of strings*) – List of strings indicated path to files where different probe confidences are stored.
- **start_layer** (*int*) – Index corresponding to the starting layer whose probe confidences are going to be averaged to obtain weights. This is an index of list_probe_filenames.
- **end_layer** (*int*) – Index corresponding to the last layer whose probe confidences are going to be averaged to obtain weights. This is an index of list_probe_filenames.
- **model_type** (*string*) – This specifies the type of simple model to be trained. Default is 'neural_keras'. Only this option is implemented now.
- **simple_model** (*function object for a Keras model*) – This is a function object that would initialize a keras model to specify the architecture of the simple model.

Returns None

fit(*x_train*, *y_train*, *x_test*, *y_test*, *simple_model*, *hps*, *model_type='neural_keras'*, *sample_weight=None*)

Fits the training data by initializing a simple model with hyper parameters and returns the test accuracy on the test dataset. This can be trained with or without sample weights. The first 500 samples of the test dataset is used as validation data.

Parameters

- **x_train** (*numpy array*) – Training dataset features for training the simple model. Dimensions (num of samples x feature dimensions)
- **y_train** (*numpy array*) – Labels for the training dataset to train on. Dimensions (num of samples x num of classes)
- **x_test** (*numpy array*) – Test dataset features. Dimensions (num of samples x feature dimensions)
- **y_test** (*numpy array*) – Test dataset labels. Dimensions (num of samples x num of classes)
- **hps** (*namedtuple*) – A namedtuple that is expected to have the following named tuple elements:

- optimizer - specified the optimizer in keras.
- complexity_param - Used for Resenet based simple model to specify number of Resunits. Used by simple model function object to intialize a simple model of appropriate complexity.
- num_classes - scalar specifying number of classes used by the simple model function.
- checkpoint_path - specifies the path for saving a checkpoint of the trained model. This is expected.
- lr_scheduler - a function object that takes in a scalar (epochs) and specified a learning rate (scalar). This is a learning rate Scheduler. Expected.
- lr_reducer - a function object that specifies how learning rates must be reduced if validation accuracy does not improve - Optional.

- **simple_model** (*function object for a Keras model*) – A function object that constructs a keras model for the simple model and returns the model object. It is expected to take in input_shape, hps.complexity_param and num_classes. It is expected to implement a keras model fit function. It is also expected to implement a keras model evaulate function.

Returns

- **model_d** (*Keras model object*) – Returns the trained model that is initialized by simple_model functions.
- **scores[1]** (*float*) – Returns the test accuracy of the trained model on (x_test,y_test.)

Return type tuple

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

[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.

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

- **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*)
Return prototypes for data X, Y.

Parameters

- **X** (*double 2d array*) – Dataset to select prototypical explanations from.
- **Y** (*double 2d array*) – Dataset you want to explain.
- **m** (*int*) – Number of prototypes

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

- **kernelType** (*str*) – Type of kernel (viz. 'Gaussian', / 'other')
- **sigma** (*double*) – width of kernel

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

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

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

Returns Self

Return type *BRCGExplainer*

[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.

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*, *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
- **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].

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.

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

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.

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

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

T

V