GANBLR Documentation - AO

Overview

The GANBLR model combines a Bayesian network and a neural network, leveraging the k-Dependency Bayesian (kDB) algorithm for building a dependency graph and using various utility functions for data preparation and model constraints.

GANBLR Model Class

Defined in ganblr.py

Class Properties:

- _d: Placeholder for data.
- __gen_weights: Placeholder for generator weights.
- batch_size: Batch size for training.
- epochs: Number of epochs for training.
- k: Parameter for the model.
- constraints: Constraints for the model.
- _ordinal_encoder: Ordinal encoder for preprocessing data.
- _label_encoder: Label encoder for preprocessing data.

Class Methods:

__init__(): Initializes the model with default values.

fit(x, y, k=0, batch_size=32, epochs=10, warmup_epochs=1, verbose=0): Fits the model to the given data.
Parameters:
- x: Input data.
- y: Labels for the data.
- k: Parameter for the model (default: 0).
- batch_size: Size of batches for training (default: 32).
- epochs: Number of training epochs (default: 10).
- warmup_epochs: Number of warmup epochs (default: 1).
- verbose: Verbosity level (default: 0).

Returns the fitted model.

_augment_cpd(d, size=None, verbose=0): Augments the Conditional Probability Distribution (CPD).
Parameters:
- d: Data.
- size: Size of the sample (default: None).
- verbose: Verbosity level (default: 0).
Returns the augmented data.

_warmup_run(epochs, verbose=None): Runs a warmup phase.
Parameters:
- epochs: Number of epochs for warmup.
- verbose: Verbosity level (default: None).
Returns the history of the warmup run.

_run_generator(loss): Runs the generator model.
Parameters:
loss: Loss function.
Returns the history of the generator run.

_discrim(): Creates a discriminator model.
Returns the discriminator model.

_sample(size=None, verbose=0): Generate synthetic data in ordinal encoding format.
Parameters:
-size: Size of the sample (default: None).
-verbose: Verbosity level (default: 0).
Returns the sampled data.

kDB Algorithm

The kDB algorithm constructs a dependency graph for the Bayesian network. It is implemented in the kdb.py file.

Methods:

build_graph(X, y, k=2): Constructs a k-dependency Bayesian network graph.
Parameters:
- X: Input data (features).
- y: Labels.
- k: Number of parent nodes (default: 2).
Returns a list of graph edges.

get_cross_table(*cols, apply_wt=False): Generates a cross table from input columns.
Parameters:
- cols: Columns for cross table generation.
-apply_wt: Whether to apply weights (default: False).
Returns a tuple containing the cross table as a NumPy array, a list of unique values for all columns and a list of unique values for individual columns.

_get_dependencies_without_y(variables, y_name, kdb_edges): Finds the dependencies of each variable without considering y.
Parameters:
-variables: List of variable names.
-y_name: Class name.
-kdb_edges: List of tuples representing edges (source, target).
Returns a dictionary of dependencies.

_add_uniform(X, weight=1.0): A uniform distribution to the data.
Parameters:
-X: Input data, a numpy array or pandas DataFrame.
-weight: Weight for the uniform distribution (default: 1.0).
Returns the modified data with uniform distribution.

_normalize_by_column(array): Normalizes the array by columns.
Parameters:
-array: Input array to normalize.
Returns the normalized array.

_smoothing(cct, d): Probability smoothing for kDB.
Parameters:
-cct: Cross count table with shape (x0, *parents).
-d: Dimension of cct.
Returns a smoothed joint probability table.

get_high_order_feature(X, col, evidence_cols, feature_uniques): Encodes the high order feature of X[col] given evidences X[evidence_cols].
Parameters:
-X: Input data.
-col: Column to encode.
-evidence_cols: List of evidence columns.
-feature_uniques: Unique values for features.
Returns an encoded high order feature.

get_high_order_constraints(X, col, evidence_cols, feature_uniques): Gets high order constraints for the feature.
Parameters:
-X: Input data.
-col: Column to encode.
-evidence_cols: List of evidence columns.
-feature_uniques: Unique values for features.
Returns high order constraints.

Classes:

KdbHighOrderFeatureEncoder: Encodes high-order features for the kDB model.
Class Properties:
- feature_uniques_: Unique values for features.
- dependencies_: Dependencies for features.
- ohe_: OneHotEncoder instance.
Class Methods:
- fit(X, y, k=0): Fits the encoder to the data.
- transform(X, return_constraints=False, use_ohe=True): Transforms the input data.
- fit_transform(X, y, k=0, return_constraints=False): Fits the encoder and transforms the data.

Utility Functions

The utility functions are implemented in the utils.py file and provide various support functions for data preparation and model constraints.

softmax_weight Class: Constrains weight tensors to be under softmax.

DataUtils Class: Provides data utilities for preparation before training.

Function: elr_loss(KL_LOSS): Defines a custom loss function.

Function: KL_loss(prob_fake): Calculates the KL loss.

Function: get_lr(input_dim, output_dim, constraint=None, KL_LOSS=0): Creates a logistic regression model.

Function: sample(*arrays, n=None, frac=None, random_state=None): Generates random samples from the given arrays.

Function: get_demo_data(name='adult'): Downloads a demo dataset from the internet.