My Project Documentation

Release 0.0.1

Your Name

Apr 13, 2025

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

API REFERENCE

This section manually lists the modules within the counterfactual_explainers package.

1.1 Top-Level Modules

A module for generating dataset statistics and preprocessing pipeline metrics.

This module provides functionality to analyze datasets and their preprocessing pipelines, calculating key metrics about feature types, encoding results, and label distributions. Results are exported to CSV for easy reporting.

Key functions:
get_pipeline_stats: Calculates preprocessing pipeline metrics for a dataset main: Main execution flow that processes all configured datasets

Typical usage: TODO

counterfactual_explainers.dataset_stats.**get_pipeline_stats**(*data: DatasetDict*) → dict[str, int | None]

Analyzes a dataset and its preprocessing pipeline to calculate key metrics.

Parameters
data – Dataset dictionary containing features, target, and metadata from read_dataset. Must include: - continuous_features: List of continuous feature names - categorical_features: List of categorical feature names - features: Full feature DataFrame - target: Target variable Series - non_act_features: List of non-actionable feature names - encode: Encoding strategy used for categorical features - scaler: Scaling strategy used for continuous features

Returns

- 'Number of Records': Total number of instances in the dataset
- 'Number of Features': Original number of features before encoding
- 'Number of Continuous Features': Count of numerical features
- 'Number of Categorical Features': Count of categorical features
- 'Number of Actionable Features': Features available for modification
- 'Number of Encoded Features': Resulting features after encoding
- 'Number of Labels': Distinct classes in the target variable

Return type
Dictionary containing calculated metrics with keys

counterfactual_explainers.dataset_stats.**main**() → None

Main execution function that processes all datasets in configuration.

Reads and cleans configuration, processes each dataset listed in the configuration file, calculates pipeline statistics, and exports results to a CSV file in the results directory.

A module for training and evaluating machine learning models with explainer integration.

This module provides end-to-end functionality for: - Configuring and training Random Forest and DNN models - Hyperparameter tuning using RandomizedSearchCV - Model evaluation with multiple metrics - Integration with different explainer frameworks (AIDE/DICE) - Saving trained models and results

Key components: - build_dnn: Constructs customizable neural network architectures - train_model: Handles model training with hyperparameter tuning - evaluate_model: Calculates performance metrics - train_and_evaluate_for_dataset: Manages dataset-specific training workflows - main: Coordinates end-to-end training process

Typical usage: TODO

counterfactual_explainers.train_models.**build_dnn**(*dim_0*, *dim_out*, *dim_1=128*, *dim_2=64*, *activation_0='relu'*, *activation_1='relu'*, *activation_2='relu'*, *dropout_0=0.3*, *dropout_1=0.1*, *dropout_2=0.01*)

Construct a deep neural network architecture with configurable layers.

Creates a sequential neural network with dense layers and optional dropout. The final layer uses sigmoid activation for binary classification or softmax for multi-class (automatically determined during training).

Parameters

- **dim_0** – Input dimension size (must match preprocessed feature dimension)
- **dim_out** – Output dimension size (number of classes)
- **dim_1** – First hidden layer size, default 128
- **dim_2** – Second hidden layer size, default 64
- **activation_0** – Activation for input layer, default 'relu'
- **activation_1** – Activation for first hidden layer, default 'relu'
- **activation_2** – Activation for second hidden layer, default 'relu'
- **dropout_0** – Dropout rate after input layer, default 0.3
- **dropout_1** – Dropout rate after first hidden layer, default 0.1
- **dropout_2** – Dropout rate after second hidden layer, default 0.01

Returns
Uncompiled Keras Sequential model

Return type
Sequential

counterfactual_explainers.train_models.**evaluate_model**(*y_true: ndarray*, *y_pred: ndarray*) → dict[str, float]

Calculate evaluation metrics for model performance.

Parameters

- **y_true** – Ground truth labels
- **y_pred** – Model predictions

Returns

- accuracy: Overall accuracy
- f1_macro: Macro-averaged F1 score
- f1_micro: Micro-averaged F1 score

Return type
Dictionary containing

counterfactual_explainers.train_models.**parse_arguments**()

Parse command line arguments for explainer framework selection.

Returns
Configured argument parser with explainer type

Return type
ArgumentParser

counterfactual_explainers.train_models.**save_model**(*best_pipeline: Pipeline*, *model_name: str*, *dataset_name: str*, *explainer_type: str*) → Path

Save trained model to disk in format appropriate for each model type.

Parameters

- **best_pipeline** – Trained scikit-learn pipeline
- **model_name** – Type of model ('RF' or 'DNN')
- **dataset_name** – Name of dataset used for training
- **explainer_type** – Type of explainer ('aide' or 'dice')

Returns
Location where model was saved

Return type
Path

counterfactual_explainers.train_models.**set_random_seeds**(*seed: int*) → None

Set random seeds for reproducibility across multiple libraries.

WARNING: May exhibit unexpected behavior with certain TensorFlow versions or when combined with GPU computations.

Parameters
seed – Integer value to seed all random number generators

counterfactual_explainers.train_models.**train_and_evaluate_for_dataset**(*dataset_name: str*, *config: dict[str, Any]*, *explainer_type: str*) → list[dict[str, Any]]

Orchestrate model training and evaluation for a single dataset.

Parameters

- **dataset_name** – Name of dataset to process
- **config** – Loaded configuration dictionary
- **explainer_type** – Type of explainer framework being used

Returns
List of dictionaries containing training results for each model

counterfactual_explainers.train_models.**train_model**(*X_train: DataFrame*, *y_train: Series*, *preprocessor: Any*, *model: Any*, *params_model: dict[str, Any]*, *seed: int*) → Pipeline

Train a machine learning model with hyperparameter tuning.

Constructs a scikit-learn pipeline with preprocessing and classifier, then performs randomized search for hyperparameter optimization.

Parameters

- **X_train** – Training features DataFrame
- **y_train** – Training target Series
- **preprocessor** – Configured data preprocessing transformer
- **model** – Uninitialized classifier model
- **params_model** – Hyperparameter search space for RandomizedSearchCV
- **seed** – Random seed for reproducibility

Returns
Best performing pipeline from hyperparameter search

Return type
Pipeline

A module for generating counterfactual explanations using the AIDE framework.

This module provides functionality for: - Loading pre-trained Keras models - Preparing dataset-specific configurations for AIDE - Generating counterfactual explanations using artificial immune networks - Handling dataset encoding/decoding for explanations - Saving counterfactual results and runtime metrics

Key components: - generate_and_save_counterfactuals: Core AIDE explanation generation workflow - generate_cfs_for_dataset: Dataset-specific explanation orchestration - decode_df: Helper for decoding encoded feature representations - DATASET_PARAMS: Pre-configured parameters for different datasets

Typical usage: TODO

counterfactual_explainers.gen_cfs_aide.**generate_and_save_counterfactuals**(*model: Model*, *X_test: DataFrame*, *index_in_arr: int*, *aide_data_object: dict[str, Any]*, *model_name: str*, *dataset_name: str*, *prob_dict: dict[str, float]*) → None

Generate and persist counterfactual explanations using AIDE.

Parameters

- **model** – Pretrained Keras model for prediction
- **X_test** – Test features DataFrame
- **index_in_arr** – Index of query instance in test set
- **aide_data_object** – Dataset-specific configuration dictionary
- **model_name** – Type of ML model being explained
- **dataset_name** – Name of dataset being used
- **prob_dict** – Probability dictionary for target classes

counterfactual_explainers.gen_cfs_aide.**generate_cfs_for_dataset**(*dataset_name: str*, *config: dict[str, Any]*) → None

Orchestrate counterfactual generation workflow for a dataset.

Parameters

- **dataset_name** – Name of dataset to process

- **config** – Configuration dictionary with parameters

counterfactual_explainers.gen_cfs_aide.**main**() → None

Main execution function for aide counterfactual generation.

A module for calculating counterfactual explanation metrics.

This module provides functionality for: - Computing various counterfactual quality metrics (distance, diversity, actionability) - Encoding counterfactual data for metric calculation - Loading pre-generated counterfactual results - Generating comprehensive metric reports for different models and datasets

Key components: - calc_mad: Computes Median Absolute Deviation for continuous features - calc_distance: Calculates normalized distance between instances - calc_diversity: Measures diversity among counterfactual explanations - encode_cfs_to_dfs: Preprocesses data for metric calculation - calculate_metrics_for_dataset: Orchestrates metric calculation workflow

Typical usage: TODO

counterfactual_explainers.calculate_metrics.**calc_actionability**(*cf_row: Series*, *query_instance: DataFrame*, *non_act_features: list[str]*) → int

Check if counterfactual makes changes to non-actionable features.

Parameters

- **cf_row** – Counterfactual instance as pandas Series
- **query_instance** – Original query instance as DataFrame
- **non_act_features** – List of non-actionable feature names

Returns

1 if no changes to non-actionable features, 0 otherwise

counterfactual_explainers.calculate_metrics.**calc_changes**(*cf_row: Series*, *query_instance: Series*, *features: list[str]*) → int

Count number of feature changes between counterfactual and query instance.

Parameters

- **cf_row** – Counterfactual instance as pandas Series
- **query_instance** – Original query instance as pandas Series
- **features** – List of feature names to consider

Returns

Number of changed features as integer

counterfactual_explainers.calculate_metrics.**calc_distance**(*cf_row: Series*, *query_instance: Series*, *mad: Series*, *continuous_features: list[str]*, *categorical_features: list[str]*) → float

Calculate normalized distance between counterfactual and query instance.

Parameters

- **cf_row** – Counterfactual instance as pandas Series
- **query_instance** – Original query instance as pandas Series
- **mad** – MAD values for continuous features
- **continuous_features** – List of continuous feature names
- **categorical_features** – List of categorical feature names

Returns
Combined normalized distance (continuous + categorical) as float

counterfactual_explainers.calculate_metrics.**calc_diversity**(*cfs: DataFrame*, *continuous_features: list[str]*, *categorical_features: list[str]*, *feature_cols: list[str]*, *mad: Series*) → tuple[float, float]

Calculate diversity metrics for a set of counterfactuals.

Parameters

- **cfs** – DataFrame of counterfactual instances
- **continuous_features** – List of continuous feature names
- **categorical_features** – List of categorical feature names
- **feature_cols** – All feature names
- **mad** – MAD values for continuous features

Returns

- diversity_distance: Normalized pairwise distance metric
- diversity_count: Normalized feature change count metric

Return type
Tuple containing

counterfactual_explainers.calculate_metrics.**calc_mad**(*cf_row: Series*) → float

Calculate Median Absolute Deviation (MAD) for a given data row.

Parameters
cf_row – Pandas Series representing a single data row

Returns
MAD value as float. Returns 1.0 if MAD is zero to avoid division by zero.

counterfactual_explainers.calculate_metrics.**calc_range_alignment**(*cfs_df: DataFrame*, *target_class_instances: DataFrame*, *continuous_features: list[str]*, *categorical_features: list[str]*) → float

Calculates the alignment between the value ranges of counterfactuals and the value ranges of actual instances belonging to the target class.

For continuous features, it uses a discretization approach over the union range, then computes the Jaccard index using scipy. For categorical features, it creates binary indicator arrays for the sets of unique values and computes the Jaccard index using scipy.

The scores are printed directly for each feature.

Parameters

- **cfs_df** – DataFrame of counterfactual instances (original, unencoded values).
- **target_class_instances** – DataFrame containing only instances from the original dataset belonging to the target class (original, unencoded values).
- **continuous_features** – List of continuous feature names.
- **categorical_features** – List of categorical feature names.

counterfactual_explainers.calculate_metrics.**calc_size**(*num_required_cfs: int*, *cfs_df: DataFrame*) → float

Calculate size metric as ratio of generated CFs to required CFs.

Parameters

- **num_required_cfs** – Requested number of counterfactuals
- **cfs_df** – DataFrame containing generated counterfactuals

Returns
Size metric as float

counterfactual_explainers.calculate_metrics.**calculate_metrics_for_dataset**(*config: dict[str, Any]*, *dataset: str*) → None

Calculate metrics for a dataset across models and explainers.

Parameters

- **config** – Configuration dictionary
- **dataset** – Dataset name to process

counterfactual_explainers.calculate_metrics.**encode_cfs_to_dfs**(*query_instance: DataFrame*, *cfs: DataFrame*, *continuous_features: list[str]*, *categorical_features: list[str]*, *features: DataFrame*, *non_act_features: list[str]*) → tuple[DataFrame, DataFrame, DataFrame, list[str], list[str], list[str]]

Preprocess and encode data for metric calculations.

Parameters

- **query_instance** – Original query instance
- **cfs** – Counterfactual instances
- **continuous_features** – List of continuous feature names
- **categorical_features** – List of categorical feature names
- **features** – Original dataset features
- **non_act_features** – List of non-actionable feature names

Returns

- Encoded counterfactuals DataFrame
- Encoded query instance DataFrame
- Encoded features DataFrame
- Continuous feature columns
- Categorical feature columns
- Non-actionable feature columns

Return type
Tuple containing

counterfactual_explainers.calculate_metrics.**load_counterfactual_data**(*dataset: str*, *model_name: str*, *explainer: str*, *num_required_cfs: int*) → tuple[DataFrame, DataFrame, float]

Load pre-generated counterfactual data and runtime information.

Parameters

- **dataset** – Dataset name
- **model_name** – Model type name
- **explainer** – Explainer method name
- **num_required_cfs** – Number of requested counterfactuals

Returns

- Counterfactuals DataFrame
- Query instance DataFrame
- Runtime in seconds

Return type
Tuple containing

counterfactual_explainers.calculate_metrics.**main**() → None

Main execution function for metric calculation workflow.

A module for visualizing and statistically analyzing counterfactual explanation metrics.

This module provides functionality for: - Loading and aggregating counterfactual evaluation metrics from CSV files. - Performing statistical tests (ANOVA and Tukey HSD) on the metric scores. - Generating interactive line plots for each metric to compare different counterfactual generation methods. - Creating parallel coordinates plots for a multi-dimensional visualization of the dataset, query instances, and counterfactuals.

Key components: - plot_metrics: Reads metric CSV files, computes statistics, and generates metric comparison plots. - plot_parallel_coordinates: Preprocesses dataset features and renders a parallel coordinates plot. - main: Orchestrates the execution workflow for visualizing counterfactual explanation results.

Typical usage: TODO

counterfactual_explainers.plot_and_stats.**main**() → None

Main function for plotting and statistical analysis.

counterfactual_explainers.plot_and_stats.**plot_metrics**(*dataset: str*) → tuple[DataFrame, list[str]]

Plot counterfactual evaluation metrics for a given dataset.

This function reads CSV files for three methods (DICE gradient-based, DICE genetic, and AIDE) if available, extracts metric values, performs ANOVA and Tukey HSD statistical tests, and creates a subplot for each metric.

Parameters
dataset – Name of the dataset to process.

Returns

- A concatenated DataFrame of metric scores from available methods.
- A list of metric names plotted.

Return type
A tuple containing

counterfactual_explainers.plot_and_stats.**plot_parallel_coordinates**(*config: dict[str, Any]*, *dataset: str*) → None

Generate and display a parallel coordinates plot for a given dataset.

This function reads and preprocesses the dataset, transforms categorical features to numerical codes, and creates a parallel coordinates plot with Plotly. It distinguishes between the original dataset, the query instance, and counterfactuals generated using different methods.

Parameters

- **config** – Configuration dictionary containing dataset and model parameters.
- **dataset** – Name of the dataset to process.

Returns

None

PYTHON MODULE INDEX

C

INDEX