uq360

Release 0.1

IBM Research

Jul 19, 2023

PACKAGE REFERENCE:

The Uncertainty Quantification 360 (UQ360) toolkit is an open-source Python package that provides a diverse set of algorithms to quantify uncertainty, as well as capabilities to measure and improve UQ to streamline the development process. We provide a taxonomy and guidance for choosing these capabilities based on the user's needs. Further, UQ360 makes the communication method of UQ an integral part of development choices in an AI lifecycle. Developers can make a user-centered choice by following the psychology-based guidance on communicating UQ estimates, from concise descriptions to detailed visualizations.

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

CHAPTER

ONE

ALGORITHMS

1.1 Intrinsic UQ Algorithms

1.1.1 Homoscedastic Gaussian Process Regression

class uq360.algorithms.homoscedastic_gaussian_process_regression.**HomoscedasticGPRegression**(*kernel=ScaleKe (base_kernel): RBFK- er- nel((raw_lengthscal Pos- i- tive()) (raw_outputscal Pos- i- tive()), like- li- hood=None, con- fig=None*)

A wrapper around Botorch SingleTask Gaussian Process Regression[1] with homoscedastic noise.

References

Parameters

- **kernel** – gpytorch kernel function with default set to *RBFKernel* with output scale.
- **likelihood** – gpytorch likelihood function with default set to *GaussianLikelihood*.
- **config** – dictionary containing the config parameters for the model.

[1] https://botorch.org/api/models.html#singletaskgp

`fit`(*X*, *y*, ***kwargs*)

Fit the GP Regression model.

Additional arguments relevant for SingleTaskGP fitting can be passed to this function.

Parameters

- **`X`** – array-like of shape (n_samples, n_features). Features vectors of the training data.
- **`y`** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values
- **`**kwargs`** – Additional arguments relevant for SingleTaskGP fitting.

Returns
self

`predict`(*X*, *return_dists=False*, *return_epistemic=False*, *return_epistemic_dists=False*)

Obtain predictions for the test points.

In addition to the mean and lower/upper bounds, also returns epistemic uncertainty (return_epistemic=True) and full predictive distribution (return_dists=True).

Parameters

- **`X`** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **`return_dists`** – If True, the predictive distribution for each instance using scipy distributions is returned.
- **`return_epistemic`** – if True, the epistemic upper and lower bounds are returned.
- **`return_epistemic_dists`** – If True, the epistemic distribution for each instance using scipy distributions is returned.

Returns

A namedtuple that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

y_lower_epistemic: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of epistemic component of the predictive distribution of the test points. Only returned when *return_epistemic* is True.

y_upper_epistemic: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of epistemic component of the predictive distribution of the test points. Only returned when *return_epistemic* is True.

dists: list of predictive distribution as *scipy.stats* objects with length n_samples.
Only returned when *return_dists* is True.

Return type
namedtuple

1.1.2 Heteroscedastic Regression

class uq360.algorithms.heteroscedastic_regression.**HeteroscedasticRegression**(*model_type=None*, *model=None*, *config=None*, *device=None*, *verbose=True*)

Wrapper for heteroscedastic regression. We learn to predict targets given features, assuming that the targets are noisy and that the amount of noise varies between data points. https://en.wikipedia.org/wiki/Heteroscedasticity

Parameters

- **model_type** – The base model architecture. Currently supported values are [mlp]. mlp modeltype learns a multi-layer perceptron with a heteroscedastic Gaussian likelihood. Both the mean and variance of the Gaussian are functions of the data point ->git N(y_n | mlp_mu(x_n), mlp_var(x_n))
- **model** – (optional) The prediction model. Currently support pytorch models that returns mean and log variance.
- **config** – dictionary containing the config parameters for the model.
- **device** – device used for pytorch models ignored otherwise.
- **verbose** – if True, print statements with the progress are enabled.

fit(*X*, *y*)

Fit the Heteroscedastic Regression model.

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the training data.
- **y** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values

Returns
self

predict(*X*, *return_dists=False*)

Obtain predictions for the test points.

In addition to the mean and lower/upper bounds, also returns epistemic uncertainty (return_epistemic=True) and full predictive distribution (return_dists=True).

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **return_dists** – If True, the predictive distribution for each instance using scipy distributions is returned.

Returns

A namedtupe that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

dists: list of predictive distribution as *scipy.stats* objects with length n_samples.
Only returned when *return_dists* is True.

Return type
namedtuple

1.1.3 Ensemble Heteroscedastic Regression

class uq360.algorithms.ensemble_heteroscedastic_regression.**EnsembleHeteroscedasticRegression**(*model_type=*, *config=None*, *device=None*, *verbose=True*)

Ensemble Regression assumes an ensemble of models of Gaussian form for the predictive distribution and returns the mean and log variance of the ensemble of Gaussians.

Initializer for Ensemble of heteroscedastic regression. :param model_type: The base model used for predicting a quantile. Currently supported values are [heteroscedasticregression]. :param config: dictionary containing the config parameters for the model. :param device: device used for pytorch models ignored otherwise.

fit(*X*, *y*)

Fit the Ensemble of Heteroscedastic Regression models. :param X: array-like of shape (n_samples, n_features).

Features vectors of the training data.

Parameters
y – array-like of shape (n_samples,) or (n_samples, n_targets) Target values

Returns
self

predict(*X*, *return_dists=False*)

Obtain predictions for the test points. In addition to the mean and lower/upper bounds, also returns epistemic uncertainty (return_epistemic=True) and full predictive distribution (return_dists=True). :param X: array-like of shape (n_samples, n_features).

Features vectors of the test points.

Parameters
return_dists – If True, the predictive distribution for each instance using scipy distributions is returned.

Returns

A namedtupe that holds y_mean: ndarray of shape (n_samples, [n_output_dims])

Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

dists: list of predictive distribution as *scipy.stats* objects with length n_samples.
Only returned when *return_dists* is True.

Return type
namedtuple

1.1.4 Actively Learned Model

class uq360.algorithms.actively_learned_model.**ActivelyLearnedModel**(*config=None*, *device=None*, *verbose=True*, *online=True*)

ActivelyLearnedModel assumes an existing BuiltinUQ model, and implements an active learning training of this model. This code is supporting Pestourie et al. "Active learning of deep surrogates for PDEs: application to metasurface design." npj Computational Materials 6.1 (2020): 1-7.

Initializer for Actively learned model. :param config: dictionary containing the config parameters for the model. For active learning: num_init, T, K, M, sampling_function, querry_function, for the used model:

{"model_function": BuilInUQ model to actively learn, "model_args": same arguments as the BuilInUQ model used, "model_kwargs": same keyword arguments as the BuilInUQ model used}

Parameters
device – device used for pytorch models ignored otherwise.

fit()

Fit the actively learned model, by increasing the dataset efficiently. NB: it does not take a dataset as argument, because it is building one during training. :returns: self

predict(*X*)

Obtain predictions for the test points. In addition to the mean and lower/upper bounds, also returns epistemic uncertainty (return_epistemic=True) and full predictive distribution (return_dists=True). :param X: array-like of shape (n_samples, n_features).

Features vectors of the test points.

Returns

A namedtupe that holds y_mean: ndarray of shape (n_samples, [n_output_dims])

Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

Return type
namedtuple

1.1.5 Quantile Regression

class uq360.algorithms.quantile_regression.**QuantileRegression**(*model_type='gbr'*, *config=None*)

Quantile Regression uses quantile loss and learns two separate models for the upper and lower quantile to obtain the prediction intervals.

Parameters

- **model_type** – The base model used for predicting a quantile. Currently supported values are [gbr]. gbr is sklearn GradientBoostingRegressor.
- **config** – dictionary containing the config parameters for the model.

fit(*X*, *y*)

Fit the Quantile Regression model.

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the training data.
- **y** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values

Returns
self

predict(*X*)

Obtain predictions for the test points.

In addition to the mean and lower/upper bounds, also returns epistemic uncertainty (return_epistemic=True) and full predictive distribution (return_dists=True).

Parameters
X – array-like of shape (n_samples, n_features). Features vectors of the test points.

Returns

A namedtupe that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

Return type
namedtuple

1.1.6 Bayesian Neural Network Regression

class uq360.algorithms.variational_bayesian_neural_networks.bnn.**BnnRegression**(*config*, *prior='Gaussian'*)

Variationally trained BNNs with Gaussian and Horseshoe[6] priors for regression.

[6] Ghosh, Soumya, Jiayu Yao, and Finale Doshi-Velez. "Structured variational learning of Bayesian neural networks with horseshoe priors." International Conference on Machine Learning. PMLR, 2018.

References

Parameters

- **config** – a dictionary specifying network and learning hyperparameters.
- **prior** – BNN priors specified as a string. Supported priors are Gaussian, Hshoe, RegHshoe

fit(*X*, *y*)

Fit the BNN regression model.

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the training data.
- **y** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values

Returns

self

predict(*X*, *mc_samples=100*, *return_dists=False*, *return_epistemic=True*, *return_epistemic_dists=False*)

Obtain predictions for the test points.

In addition to the mean and lower/upper bounds, also returns epistemic uncertainty (return_epistemic=True) and full predictive distribution (return_dists=True).

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **mc_samples** – Number of Monte-Carlo samples.
- **return_dists** – If True, the predictive distribution for each instance using scipy distributions is returned.
- **return_epistemic** – if True, the epistemic upper and lower bounds are returned.
- **return_epistemic_dists** – If True, the epistemic distribution for each instance using scipy distributions is returned.

Returns

A namedtupe that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

y_lower_epistemic: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of epistemic component of the predictive distribution of the test points. Only returned when *return_epistemic* is True.

y_upper_epistemic: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of epistemic component of the predictive distribution of the test points. Only returned when *return_epistemic* is True.

dists: list of predictive distribution as *scipy.stats* objects with length n_samples.
Only returned when *return_dists* is True.

Return type
namedtuple

1.1.7 Bayesian Neural Network Classification

class uq360.algorithms.variational_bayesian_neural_networks.bnn.**BnnClassification**(*config*, *prior='Gaussian'*, *device=None*)

Variationally trained BNNs with Gaussian and Horseshoe[Page 8, 6] priors for classification.

Parameters

- **config** – a dictionary specifying network and learning hyperparameters.
- **prior** – BNN priors specified as a string. Supported priors are Gaussian, Hshoe, RegHshoe

fit(*X=None*, *y=None*, *train_loader=None*)

Fits BNN regression model.

Parameters

- **X** – array-like of shape (n_samples, n_features) or (n_samples, n_classes). Features vectors of the training data or the probability scores from the base model. Ignored if train_loader is not None.
- **y** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values Ignored if train_loader is not None.
- **train_loader** – pytorch train_loader object.

Returns
self

predict(*X*, *mc_samples=100*)

Obtain calibrated predictions for the test points.

Parameters

- **X** – array-like of shape (n_samples, n_features) or (n_samples, n_classes). Features vectors of the training data or the probability scores from the base model.
- **mc_samples** – Number of Monte-Carlo samples.

Returns

A namedtupe that holds

y_pred: ndarray of shape (n_samples,)
Predicted labels of the test points.

y_prob: ndarray of shape (n_samples, n_classes)
Predicted probability scores of the classes.

y_prob_var: ndarray of shape (n_samples,)
Variance of the prediction on the test points.

y_prob_samples: ndarray of shape (mc_samples, n_samples, n_classes)
Samples from the predictive distribution.

Return type
namedtuple

1.2 Extrinsic UQ Algorithms

1.2.1 Auxiliary Interval Predictor

class uq360.algorithms.auxiliary_interval_predictor.**AuxiliaryIntervalPredictor**(*model_type=None*, *main_model=None*, *aux_model=None*, *config=None*, *device=None*, *verbose=True*)

Auxiliary Interval Predictor[1] uses an auxiliary model to encourage calibration of the main model.

References

Parameters

- **model_type** – The model type used to build the main model and the auxiliary model. Currently supported values are [mlp, custom]. *mlp* modeltype learns a mlp neural network using pytorch framework. For *custom* the user provide *main_model* and *aux_model*.
- **main_model** – (optional) The main prediction model. Currently support pytorch models that return mean and log variance.
- **aux_model** – (optional) The auxiliary prediction model. Currently support pytorch models that return calibrated log variance.
- **config** – dictionary containing the config parameters for the model.
- **device** – device used for pytorch models ignored otherwise.
- **verbose** – if True, print statements with the progress are enabled.

fit(*X*, *y*)

Fit the Auxiliary Interval Predictor model.

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the training data.
- **y** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values

Returns
self

predict(*X*, *return_dists=False*)

Obtain predictions for the test points.

In addition to the mean and lower/upper bounds, also returns full predictive distribution (return_dists=True).

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the test points.

[1] Thiagarajan, J. J., Venkatesh, B., Sattigeri, P., & Bremer, P. T. (2020, April). Building calibrated deep models via uncertainty matching with auxiliary interval predictors. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 34, No. 04, pp. 6005-6012). https://arxiv.org/abs/1909.04079

- **return_dists** – If True, the predictive distribution for each instance using scipy distributions is returned.

Returns

A namedtupe that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

dists: list of predictive distribution as *scipy.stats* objects with length n_samples.
Only returned when *return_dists* is True.

Return type
namedtuple

1.2.2 Blackbox Metamodel Classification

1.2.3 Blackbox Metamodel Regression

1.2.4 Infinitesimal Jackknife

class uq360.algorithms.infinitesimal_jackknife.**InfinitesimalJackknife**(*params*, *gradients*, *hessian*, *config*)

Performs a first order Taylor series expansion around MLE / MAP fit. Requires the model being probed to be twice differentiable.

Initialize IJ. :param params: MLE / MAP fit around which uncertainty is sought. d*1 :param gradients: Per data point gradients, estimated at the MLE / MAP fit. d*n :param hessian: Hessian evaluated at the MLE / MAP fit. d*d

approx_ij(*w_query*)

Parameters
w_query – A n*1 vector to query parameters at.

Returns
new parameters at w_query

get_params(*deep=True*)

This method should not take any arguments and returns a dict of the __init__ parameters.

ij(*w_query*)

Parameters
w_query – A n*1 vector to query parameters at.

Returns
new parameters at w_query

predict(*X*, *model*)

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **model** – model object, must implement a set_parameters function

Returns

A namedtupe that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

Return type
namedtuple

1.2.5 Classification Calibration

class uq360.algorithms.classification_calibration.**ClassificationCalibration**(*num_classes*, *fit_mode='features'*, *method='isotonic'*, *base_model_prediction_func=None*

Post hoc calibration of classification models. Currently wraps *CalibratedClassifierCV* from sklearn and allows non-sklearn models to be calibrated.

Parameters

- **num_classes** – number of classes.
- **fit_mode** – features or probs. If probs the *fit* and *predict* operate on the base models probability scores, useful when these are precomputed.
- **method** – isotonic or sigmoid.
- **base_model_prediction_func** – the function that takes in the input features and produces base model's probability scores. This is ignored when operating in *probs* mode.

fit(*X*, *y*)

Fits calibration model using the provided calibration set.

Parameters

- **X** – array-like of shape (n_samples, n_features) or (n_samples, n_classes). Features vectors of the training data or the probability scores from the base model.
- **y** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values

Returns
self

get_params(*deep=True*)

This method should not take any arguments and returns a dict of the __init__ parameters.

predict(*X*)

Obtain calibrated predictions for the test points.

Parameters
X – array-like of shape (n_samples, n_features) or (n_samples, n_classes). Features vectors of the training data or the probability scores from the base model.

Returns

A namedtupe that holds

y_pred: ndarray of shape (n_samples,)
Predicted labels of the test points.

y_prob: ndarray of shape (n_samples, n_classes)
Predicted probability scores of the classes.

Return type
namedtuple

1.2.6 UCC Recalibration

class uq360.algorithms.ucc_recalibration.**UCCRecalibration**(*base_model*)

Recalibration a regression model to specified operating point using Uncertainty Characteristics Curve.

Parameters
base_model – pretrained model to be recalibrated.

fit(*X*, *y*)

Fit the Uncertainty Characteristics Curve.

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **y** – array-like of shape (n_samples,) or (n_samples, n_targets) Target values

Returns
self

get_params(*deep=True*)

This method should not take any arguments and returns a dict of the __init__ parameters.

predict(*X*, *missrate=0.05*)

Generate prediction and uncertainty bounds for data X.

Parameters

- **X** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **missrate** – desired missrate of the new operating point, set to 0.05 by default.

Returns

A namedtupe that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_lower: ndarray of shape (n_samples, [n_output_dims])
Lower quantile of predictive distribution of the test points.

y_upper: ndarray of shape (n_samples, [n_output_dims])
Upper quantile of predictive distribution of the test points.

Return type
namedtuple

1.2.7 Structured Data Predictor

class uq360.algorithms.blackbox_metamodel.structured_data_classification.**StructuredDataClassificationWra**

This predictor allows flexible feature and calibrator configurations, and uses a meta-model which is an ensemble of a GBM and a Logistic Regression model. It returns no errorbars (constant zero errorbars) of its own. PostHocUQ model based on the "structured_data" performance predictor

(uq360.algorithms.blackbox_metamodel.predictors.core.structured_data.py).

Returns an instance of a structured data predictor

Parameters
base_model – scikit learn estimator instance which has the capability of returning confidence (predict_proba). base_model can also be None

Returns
predictor instance

fit(*x_train*, *y_train*, *x_test*, *y_test*, *test_predicted_probabilities=None*)

Fit base and meta models.

Parameters

- **x_train** – Features vectors of the training data.
- **y_train** – Labels of the training data
- **x_test** – Features vectors of the test data.
- **y_test** – Labels of the test data
- **test_predicted_probabilities** – predicted probabilities on test data should be passed if the predictor is not instantiated with a base model

Returns
self

predict(*x*, *return_predictions=True*, *predicted_probabilities=None*)

Generate a base prediction for incoming data x

Parameters

- **x** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **return_predictions** – data point wise prediction will be returned when this flag is True
- **predicted_probabilities** – when the predictor is instantiated without a base model, predicted_probabilities on x from the pre-trained model should be passed to predict

Returns
namedtuple: A namedtuple that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_pred: ndarray of shape (n_samples,) Predicted labels of the test points. y_score: ndarray of shape (n_samples,)

Confidence score the test points.

1.2.8 Short Text Predictor

class uq360.algorithms.blackbox_metamodel.short_text_classification.**ShortTextClassificationWrapper**(*base_encoder*

This is very similar to the structured data predictor but it is fine tuned to handle text data. The meta model used by the predictor is an ensemble of an SVM, GBM, and MLP. Feature vectors can be either raw text or pre-encoded vectors. If raw text is passed and no encoder is specified in the initialization, USE embeddings will be used by default. PostHocUQ model based on the "text_ensemble" performance predictor (uq360.algorithms.blackbox_metamodel.predictors.core.short_text.py).

Returns an instance of a short text predictor :param base_model: scikit learn estimator instance which has the capability of returning confidence (predict_proba). base_model can also be None :return: predictor instance

fit(*x_train*, *y_train*, *x_test*, *y_test*, *test_predicted_probabilities=None*)

Fit base and meta models.

Parameters

- **x_train** – Features vectors of the training data.
- **y_train** – Labels of the training data
- **x_test** – Features vectors of the test data.
- **y_test** – Labels of the test data
- **test_predicted_probabilities** – predicted probabilities on test data should be passed if the predictor is not instantiated with a base model

Returns
self

predict(*x*, *return_predictions=True*, *predicted_probabilities=None*)

Generate a base prediction for incoming data x

Parameters

- **x** – array-like of shape (n_samples, n_features). Features vectors of the test points.
- **return_predictions** – data point wise prediction will be returned when this flag is True
- **predicted_probabilities** – when the predictor is instantiated without a base model, predicted_probabilities on x from the pre-trained model should be passed to predict

Returns
namedtuple: A namedtuple that holds

y_mean: ndarray of shape (n_samples, [n_output_dims])
Mean of predictive distribution of the test points.

y_pred: ndarray of shape (n_samples,) Predicted labels of the test points. y_score: ndarray of shape (n_samples,)

Confidence score the test points.

1.2.9 Confidence Predictor

1.2.10 Latent Space Anomaly Detection Scores

class uq360.algorithms.layer_scoring.mahalanobis.**MahalanobisScorer**(*model=None*, *layer=None*)

Implementation of the Mahalanobis Adversarial/Out-of-distribution detector [1].

[1] "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks", K. Lee et al., NIPS 2018.

Parameters

- **model** – torch Module to analyze
- **layer** – layer (torch Module) inside the model whose output is to be analyzed

Notes

The model and layer arguments are optional. If no model or layer is provided, it is expected that the inputs are already latent vectors. If both a model and layers are provided, inputs are expected to be model inputs to be mapped to latent vectors.

fit(*X: ndarray*, *y: ndarray*)

Register data X and class labels y as in-distribution data

get_params()

This method is parameterless

predict(*X: ndarray*)

Compute the Mahalanobis distance between query data X and the in-distribution data classes

class uq360.algorithms.layer_scoring.knn.**KNNScorer**(*n_neighbors: int*, *method: str = 'knn'*, *nearest_neighbors: Optional[BaseNearestNeighbors] = None*, *nearest_neighbors_kwargs={}*, *model=None*, *layer=None*)

KNN-based latent space anomaly detector. Return some measure of distance to the training data.

Parameters

- **n_neighbors** – number of nearest neighbors to consider in in-distribution data
- **method** – one of ("knn", "avg", "lid"). These correspond respectively to the distance to the k-th neighbor, the mean of the kNN,
- **nearest_neighbors** – nearest neighbor algorithm, see uq360.utils.transformers.nearest_neighbors
- **nearest_neighbors_kwargs** – keyword arguments for the NN algorithm
- **model** – torch Module to analyze
- **layer** – layer (torch Module) inside the model whose output is to be analyzed

Notes

The model and layer arguments are optional. If no model or layer is provided, it is expected that the inputs are already latent vectors. If both a model and layers are provided, inputs are expected to be model inputs to be mapped to latent vectors.

fit(*X: ndarray*)

Register X as in-distribution data

get_params()

This method should not take any arguments and returns a dict of the __init__ parameters.

predict(*X: ndarray*, *n_neighbors=None*, *method: Optional[str] = None*)

Compute a KNN-distance-based anomaly score on query data X.

Parameters

- **X** – query data
- **n_neighbors** – number of nearest neighbors to consider in in-distribution data
- **method** – one of ("knn", "avg", "lid"). These correspond respectively to the distance to the k-th neighbor, the mean of the kNN,

Returns

anomaly scores

class uq360.algorithms.layer_scoring.aklpe.**AKLPEScorer**(*nearest_neighbors: Optional[BaseNearestNeighbors] = None*, *nearest_neighbors_kwargs={}*, *n_neighbors: int = 50*, *n_bootstraps: int = 10*, *batch_size: int = 1*, *random_state: int = 123*, *model=None*, *layer=None*)

Implementation of Averaged K nearest neighbors Localized P-value Estimation (aK_LPE) [1].

[1] J. Qian and V. Saligrama, "New statistic in P-value estimation for anomaly detection," 2012 IEEE Statistical Signal Processing Workshop (SSP)

Parameters

- **nearest_neighbors** – nearest neighbor algorithm, see uq360.utils.transformers.nearest_neighbors
- **nearest_neighbors_kwargs** – keyword arguments for the NN algorithm
- **n_neighbors** – number of NN to consider
- **n_bootstraps** – number of bootstraps to estimate the p-value
- **batch_size** – int
- **random_state** – seed for RNG
- **model** – torch Module to analyze
- **layer** – layer (torch Module) inside the model whose output is to be analyzed

Notes

The model and layer arguments are optional. If no model or layer is provided, it is expected that the inputs are already latent vectors. If both a model and layers are provided, inputs are expected to be model inputs to be mapped to latent vectors.

fit(*X: ndarray*)

Register X as in-distribution data

get_params()

This method should not take any arguments and returns a dict of the __init__ parameters.

predict(*X: ndarray*)

Compute the anomaly score based on the AKLPE G-statistics

Parameters
X – query vector

Returns
g_stats, p_value containing respectively the G-statistics and the corresponding AKLPE anomaly p-value.

Return type
pair of numpy arrays

Nearest Neighbors Algorithms for KNN-based anomaly detection

class uq360.utils.transformers.nearest_neighbors.exact.**ExactNearestNeighbors**

Exact nearest neighbor search using scikit-learn

CHAPTER

TWO

METRICS

2.1 Classification Metrics

uq360.metrics.classification_metrics.**area_under_risk_rejection_rate_curve**(*y_true*, *y_prob*, *y_pred=None*, *selection_scores=None*, *risk_func=<function accuracy_score>*, *attributes=None*, *num_bins=10*, *subgroup_ids=None*, *return_counts=False*)

Computes risk vs rejection rate curve and the area under this curve. Similar to risk-coverage curves[3] where coverage instead of rejection rate is used.

References

Parameters

- **y_true** – array-like of shape (n_samples,) ground truth labels.
- **y_prob** – array-like of shape (n_samples, n_classes). Probability scores from the base model.
- **y_pred** – array-like of shape (n_samples,) predicted labels.
- **selection_scores** – scores corresponding to certainty in the predicted labels.
- **risk_func** – risk function under consideration.
- **attributes** – (optional) if risk function is a fairness metric also pass the protected attribute name.
- **num_bins** – number of bins.
- **subgroup_ids** – (optional) selectively compute risk on a subgroup of the samples specified by subgroup_ids.
- **return_counts** – set to True to return counts also.

Returns

[3] Franc, Vojtech, and Daniel Prusa. "On discriminative learning of prediction uncertainty." In International Conference on Machine Learning, pp. 1963-1971. 2019.

- aurrrc (float): area under risk rejection rate curve.
- rejection_rates (list): rejection rates for each bin (returned only if return_counts is True).
- selection_thresholds (list): selection threshold for each bin (returned only if return_counts is True).
- risks (list): risk in each bin (returned only if return_counts is True).

Return type
float or tuple

uq360.metrics.classification_metrics.**compute_classification_metrics**(*y_true*, *y_prob*, *option='all'*)

Computes the metrics specified in the option which can be string or a list of strings. Default option *all* computes the [aurrrc, ece, auroc, nll, brier, accuracy] metrics.

Parameters

- **y_true** – array-like of shape (n_samples,) ground truth labels.
- **y_prob** – array-like of shape (n_samples, n_classes). Probability scores from the base model.
- **option** – string or list of string contained the name of the metrics to be computed.

Returns
a dictionary containing the computed metrics.

Return type
dict

uq360.metrics.classification_metrics.**entropy_based_uncertainty_decomposition**(*y_prob_samples*)

Entropy based decomposition[2] of predictive uncertainty into aleatoric and epistemic components.

References

Parameters
y_prob_samples – ndarray of shape (mc_samples, n_samples, n_classes) Samples from the predictive distribution. Here mc_samples stands for the number of Monte-Carlo samples, n_samples is the number of data points and n_classes is the number of classes.

Returns

- total_uncertainty: entropy of the predictive distribution.
- aleatoric_uncertainty: aleatoric component of the total_uncertainty.
- epistemic_uncertainty: epistemic component of the total_uncertainty.

Return type
tuple

uq360.metrics.classification_metrics.**expected_calibration_error**(*y_true*, *y_prob*, *y_pred=None*, *num_bins=10*, *return_counts=False*)

Computes the reliability curve and the expected calibration error[1] .

[2] Depeweg, S., Hernandez-Lobato, J. M., Doshi-Velez, F., & Udluft, S. (2018, July). Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In International Conference on Machine Learning (pp. 1184-1193). PMLR.

[1] Chuan Guo, Geoff Pleiss, Yu Sun, Kilian Q. Weinberger; Proceedings of the 34th International Conference on Machine Learning, PMLR 70:1321-1330, 2017.

References

Parameters

- **y_true** – array-like of shape (n_samples,) ground truth labels.
- **y_prob** – array-like of shape (n_samples, n_classes). Probability scores from the base model.
- **y_pred** – array-like of shape (n_samples,) predicted labels.
- **num_bins** – number of bins.
- **return_counts** – set to True to return counts also.

Returns

- ece (float): expected calibration error.
- confidences_in_bins: average confidence in each bin (returned only if return_counts is True).
- accuracies_in_bins: accuracy in each bin (returned only if return_counts is True).
- frac_samples_in_bins: fraction of samples in each bin (returned only if return_counts is True).

Return type
float or tuple

uq360.metrics.classification_metrics.**multiclass_brier_score**(*y_true*, *y_prob*)

Brier score for multi-class.

Parameters

- **y_true** – array-like of shape (n_samples,) ground truth labels.
- **y_prob** – array-like of shape (n_samples, n_classes). Probability scores from the base model.

Returns
Brier score.

Return type
float

uq360.metrics.classification_metrics.**plot_reliability_diagram**(*y_true*, *y_prob*, *y_pred*, *plot_label=['']*, *num_bins=10*)

Plots the reliability diagram showing the calibration error for different confidence scores. Multiple curves can be plot by passing data as lists.

Parameters

- **y_true** – array-like or or a list of array-like of shape (n_samples,) ground truth labels.
- **y_prob** – array-like or or a list of array-like of shape (n_samples, n_classes). Probability scores from the base model.
- **y_pred** – array-like or or a list of array-like of shape (n_samples,) predicted labels.
- **plot_label** – (optional) list of names identifying each curve.
- **num_bins** – number of bins.

Returns

- ece_list: ece: list containing expected calibration error for each curve.

- accuracies_in_bins_list: list containing binned average accuracies for each curve.
- frac_samples_in_bins_list: list containing binned sample frequencies for each curve.
- confidences_in_bins_list: list containing binned average confidence for each curve.

Return type
tuple

uq360.metrics.classification_metrics.**plot_risk_vs_rejection_rate**(*y_true*, *y_prob*, *y_pred*, *selection_scores=None*, *plot_label=['']*, *risk_func=None*, *attributes=None*, *num_bins=10*, *subgroup_ids=None*)

Plots the risk vs rejection rate curve showing the risk for different rejection rates. Multiple curves can be plot by passing data as lists.

Parameters

- **y_true** – array-like or or a list of array-like of shape (n_samples,) ground truth labels.
- **y_prob** – array-like or or a list of array-like of shape (n_samples, n_classes). Probability scores from the base model.
- **y_pred** – array-like or or a list of array-like of shape (n_samples,) predicted labels.
- **selection_scores** – ndarray or a list of ndarray containing scores corresponding to certainty in the predicted labels.
- **risk_func** – risk function under consideration.
- **attributes** – (optional) if risk function is a fairness metric also pass the protected attribute name.
- **num_bins** – number of bins.
- **subgroup_ids** – (optional) ndarray or a list of ndarray containing subgroup_ids to selectively compute risk on a subgroup of the samples specified by subgroup_ids.

Returns

- aurrrc_list: list containing the area under risk rejection rate curves.
- rejection_rate_list: list containing the binned rejection rates.
- selection_thresholds_list: list containing the binned selection thresholds.
- risk_list: list containing the binned risks.

Return type
tuple

2.2 Regression Metrics

uq360.metrics.regression_metrics.**auucc_gain**(*y_true*, *y_mean*, *y_lower*, *y_upper*)

Computes the Area Under the Uncertainty Characteristics Curve (AUUCC) gain wrt to a null reference with constant band.

Parameters

- **y_true** – Ground truth
- **y_mean** – predicted mean
- **y_lower** – predicted lower bound
- **y_upper** – predicted upper bound

Returns
AUUCC gain

Return type
float

uq360.metrics.regression_metrics.**compute_regression_metrics**(*y_true*, *y_mean*, *y_lower*, *y_upper*, *option='all'*, *nll_fn=None*)

Computes the metrics specified in the option which can be string or a list of strings. Default option *all* computes the ["rmse", "nll", "auucc_gain", "picp", "mpiw", "r2"] metrics.

Parameters

- **y_true** – Ground truth
- **y_mean** – predicted mean
- **y_lower** – predicted lower bound
- **y_upper** – predicted upper bound
- **option** – string or list of string contained the name of the metrics to be computed.
- **nll_fn** – function that evaluates NLL, if None, then computes Gaussian NLL using y_mean and y_lower.

Returns
dictionary containing the computed metrics.

Return type
dict

uq360.metrics.regression_metrics.**mpiw**(*y_lower*, *y_upper*)

Mean Prediction Interval Width (MPIW). Computes the average width of the the prediction intervals. Measures the sharpness of intervals.

Parameters

- **y_lower** – predicted lower bound
- **y_upper** – predicted upper bound

Returns
the average width the prediction interval across samples.

Return type
float

uq360.metrics.regression_metrics.**negative_log_likelihood_Gaussian**(*y_true*, *y_mean*, *y_lower*, *y_upper*)

Computes Gaussian negative_log_likelihood assuming symmetric band around the mean.

Parameters

- **y_true** – Ground truth
- **y_mean** – predicted mean
- **y_lower** – predicted lower bound
- **y_upper** – predicted upper bound

Returns
nll

Return type
float

uq360.metrics.regression_metrics.**picp**(*y_true*, *y_lower*, *y_upper*)

Prediction Interval Coverage Probability (PICP). Computes the fraction of samples for which the grounds truth lies within predicted interval. Measures the prediction interval calibration for regression.

Parameters

- **y_true** – Ground truth
- **y_lower** – predicted lower bound
- **y_upper** – predicted upper bound

Returns
the fraction of samples for which the grounds truth lies within predicted interval.

Return type
float

uq360.metrics.regression_metrics.**plot_picp_by_feature**(*x_test*, *y_test*, *y_test_pred_lower_total*, *y_test_pred_upper_total*, *num_bins=10*, *ax=None*, *figsize=None*, *dpi=None*, *xlims=None*, *ylims=None*, *xscale='linear'*, *title=None*, *xlabel=None*, *ylabel=None*)

Plot how prediction uncertainty varies across the entire range of a feature.

Parameters

- **x_test** – One dimensional ndarray. Feature column of the test dataset.
- **y_test** – One dimensional ndarray. Ground truth label of the test dataset.
- **y_test_pred_lower_total** – One dimensional ndarray. Lower bound of the total uncertainty range.
- **y_test_pred_upper_total** – One dimensional ndarray. Upper bound of the total uncertainty range.
- **num_bins** – int. Number of bins used to discritize x_test into equal-sample-sized bins.
- **ax** – matplotlib.axes.Axes or None, optional (default=None). Target axes instance. If None, new figure and axes will be created.
- **figsize** – tuple of 2 elements or None, optional (default=None). Figure size.
- **dpi** – int or None, optional (default=None). Resolution of the figure.

- **xlims** – tuple of 2 elements or None, optional (default=None). Tuple passed to `ax.xlim()`.
- **ylims** – tuple of 2 elements or None, optional (default=None). Tuple passed to `ax.ylim()`.
- **xscale** – Passed to `ax.set_xscale()`.
- **title** – string or None, optional Axes title. If None, title is disabled.
- **xlabel** – string or None, optional X-axis title label. If None, title is disabled.
- **ylabel** – string or None, optional Y-axis title label. If None, title is disabled.

Returns
ax : The plot with PICP scores binned by a feature.

Return type
matplotlib.axes.Axes

uq360.metrics.regression_metrics.**plot_uncertainty_by_feature**(*x_test*, *y_test_pred_mean*, *y_test_pred_lower_total*, *y_test_pred_upper_total*, *y_test_pred_lower_epistemic=None*, *y_test_pred_upper_epistemic=None*, *ax=None*, *figsize=None*, *dpi=None*, *xlims=None*, *xscale='linear'*, *title=None*, *xlabel=None*, *ylabel=None*)

Plot how prediction uncertainty varies across the entire range of a feature.

Parameters

- **x_test** – one dimensional ndarray. Feature column of the test dataset.
- **y_test_pred_mean** – One dimensional ndarray. Model prediction for the test dataset.
- **y_test_pred_lower_total** – One dimensional ndarray. Lower bound of the total uncertainty range.
- **y_test_pred_upper_total** – One dimensional ndarray. Upper bound of the total uncertainty range.
- **y_test_pred_lower_epistemic** – One dimensional ndarray. Lower bound of the epistemic uncertainty range.
- **y_test_pred_upper_epistemic** – One dimensional ndarray. Upper bound of the epistemic uncertainty range.
- **ax** – matplotlib.axes.Axes or None, optional (default=None). Target axes instance. If None, new figure and axes will be created.
- **figsize** – tuple of 2 elements or None, optional (default=None). Figure size.
- **dpi** – int or None, optional (default=None). Resolution of the figure.
- **xlims** – tuple of 2 elements or None, optional (default=None). Tuple passed to `ax.xlim()`.
- **xscale** – Passed to `ax.set_xscale()`.
- **title** – string or None, optional Axes title. If None, title is disabled.
- **xlabel** – string or None, optional X-axis title label. If None, title is disabled.
- **ylabel** – string or None, optional Y-axis title label. If None, title is disabled.

Returns
ax : The plot with model's uncertainty binned by a feature.

Return type
matplotlib.axes.Axes

uq360.metrics.regression_metrics.**plot_uncertainty_distribution**(*dist*, *show_quantile_dots=False*, *qd_sample=20*, *qd_bins=7*, *ax=None*, *figsize=None*, *dpi=None*, *title='Predicted Distribution'*, *xlims=None*, *xlabel='Prediction'*, *ylabel='Density'*, ***kwargs*)

Plot the uncertainty distribution for a single distribution.

Parameters

- **dist** – scipy.stats._continuous_distns. A scipy distribution object.
- **show_quantile_dots** – boolean. Whether to show quantil dots on top of the density plot.
- **qd_sample** – int. Number of dots for the quantile dot plot.
- **qd_bins** – int. Number of bins for the quantile dot plot.
- **ax** – matplotlib.axes.Axes or None, optional (default=None). Target axes instance. If None, new figure and axes will be created.
- **figsize** – tuple of 2 elements or None, optional (default=None). Figure size.
- **dpi** – int or None, optional (default=None). Resolution of the figure.
- **title** – string or None, optional (default=Prediction Distribution) Axes title. If None, title is disabled.
- **xlims** – tuple of 2 elements or None, optional (default=None). Tuple passed to `ax.xlim()`.
- **xlabel** – string or None, optional (default=Prediction) X-axis title label. If None, title is disabled.
- **ylabel** – string or None, optional (default=Density) Y-axis title label. If None, title is disabled.

Returns
ax : The plot with prediction distribution.

Return type
matplotlib.axes.Axes

2.3 Uncertainty Characteristics Curve

class uq360.metrics.uncertainty_characteristics_curve.**UncertaintyCharacteristicsCurve**(*normalize=True*, *precompute_bias_data=True*)

Class with main functions of the Uncertainty Characteristics Curve (UCC).

Parameters

- **normalize** – set initial axes normalization flag (can be changed via set_coordinates())
- **precompute_bias_data** – if True, fit() will compute statistics necessary to generate bias-based UCCs (in addition to the scale-based ones). Skipping this precomputation may speed up the fit() call if bias-based UCC is not needed.

fit(*X*, *gt*)

Calculates internal arrays necessary for other methods (plotting, auc, cost minimization). Re-entrant.

Parameters

- **X** – [numsamples, 3] numpy matrix, or list of numpy matrices. Col 1: predicted values Col 2: lower band (deviate) wrt predicted value (always positive) Col 3: upper band wrt predicted value (always positive) If list is provided, all methods will output corresponding metrics as lists as well!
- **gt** – Ground truth array (i.e.,the 'actual' values corresponding to predictions in X

Returns

self

get_AUUCC(*vary_bias=False*, *aucfct='trapz'*, *partial_x=None*, *partial_y=None*)

returns approximate area under the curve on current coordinates, for each component.

Parameters

- **vary_bias** – False == varies scale, True == varies bias
- **aucfct** – specifies AUC integrator (can be "trapz", "simps")
- **partial_x** – tuple (x_min, x_max) defining interval on x to calc a a partial AUC. The interval bounds refer to axes as visualized (ie. potentially normed)
- **partial_y** – tuple (y_min, y_max) defining interval on y to calc a a partial AUC. partial_x must be None.

Returns

list of floats with AUUCCs for each input component, or a single float, if there is only 1 component.

get_OP(*scale=1.0*, *bias=0.0*)

Returns all Operating Points for original input data, on coordinates currently set up, given a scale/bias.

Parameters

- **scale** –
- **bias** –

Returns

list of tuples (x point, y point, unit of x, unit of y) or a single tuple if there is only 1 component.

get_specific_operating_point(*req_x_axis_value=None*, *req_y_axis_value=None*, *req_critical_value=None*, *vary_bias=False*)

Finds corresponding operating point on the current UCC, given a point on either x or y axis. Returns a list of recipes how to achieve the point (x,y), for each component. If there is only one component, returns a single recipe dict.

Parameters

- **req_x_axis_value** – requested x value on UCC (normalization status is taken from current display)
- **req_y_axis_value** – requested y value on UCC (normalization status is taken from current display)
- **vary_bias** – set to True when referring to bias-induced UCC (scale UCC default)

Returns

list of dicts (recipes), or a single dict

minimize_cost(*x_axis_cost=0.5*, *y_axis_cost=0.5*, *augment_cost_by_normfactor=True*, *search=('scale', 'bias')*)

Find minima of a linear cost function for each component. Cost function C = x_axis_cost * x_axis_value + y_axis_cost * y_axis_value. A minimum can occur in the scale-based or bias-based UCC (this can be constrained by the 'search' arg). The function returns a 'recipe' how to achieve the corresponding minimum, for each component.

Parameters

- **x_axis_cost** – weight of one unit on x_axis
- **y_axis_cost** – weight of one unit on y_axis
- **augment_cost_by_normfactor** – when False, the cost multipliers will apply as is. If True, they will be pre-normed by the corresponding axis norm (where applicable), to account for range differences between axes.
- **search** – list of types over which minimization is to be performed, valid elements are 'scale' and 'bias'.

Returns

list of dicts - one per component, or a single dict, if there is only one component. Dict keys are - 'operation': can be 'bias' (additive) or 'scale' (multiplicative), 'modvalue': value to multiply by or to add to error bars to achieve the minimum, 'new_x'/'new_y': new coordinates (operating point) with that minimum, 'cost': new cost at minimum point, 'original_cost': original cost (original operating point).

plot_UCC(*titlestr=''*, *syslabel='model'*, *outfn=None*, *vary_bias=False*, *markers=None*, *xlim=None*, *ylim=None*, ***kwargs*)

Will plot/display the UCC based on current data and coordinates. Multiple curves will be shown if there are multiple data components (via fit())

Parameters

- **titlestr** – Plot title string
- **syslabel** – list is label strings to appear in the plot legend. Can be single, if one component.
- **outfn** – base name of an image file to be created (will append .png before creating)
- **vary_bias** – True will switch to varying additive bias (default is multiplicative scale)
- **markers** – None or a list of marker styles to be used for each curve. List must be same or longer than number of components. Markers can be one among these ['o', 's', 'v', 'D', '+'].
- **xlim** – tuples or lists of specifying the range for the x axis, or None (auto)
- **ylim** – tuples or lists of specifying the range for the y axis, or None (auto)
- ****kwargs** – Additional arguments passed to the main plot call.

Returns

list of areas under the curve (or single area, if one data component) list of operating points (or single op): format of an op is tuple (xaxis value, yaxis value, xunit, yunit)

set_coordinates(*x_axis_name=None*, *y_axis_name=None*, *normalize=None*)

Assigns user-specified type to the axes and normalization behavior (sticky).

Parameters

- **x_axis_name** – None-> unchanged, or name from self.axes_name2idx

- **y_axis_name** – ditto
- **normalize** – True/False will activate/deactivate norming for specified axes. Behavior for Axes_name that are None will not be changed. Value None will leave norm status unchanged. Note, axis=='missrate' will never get normalized, even with normalize == True

Returns
none

set_std_unit(*std_unit=None*)

Sets the UCC's unit to be used when displaying normalized axes.

Parameters
std_unit – if None, the unit will be calculated as stddev of the ground truth data (ValueError raised if data has not been set at this point) or set to the user-specified value.

Returns

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX

H

I

K

M

N

P

Q

S

U