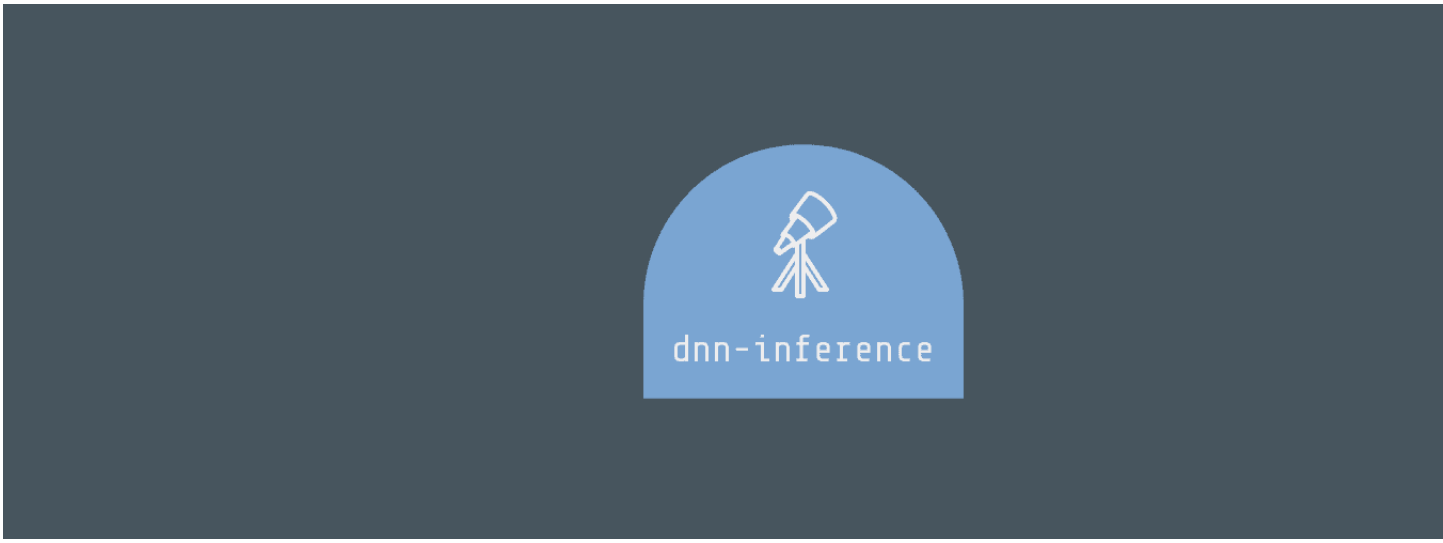
dnn-inference

Mar 19, 2021

Contents

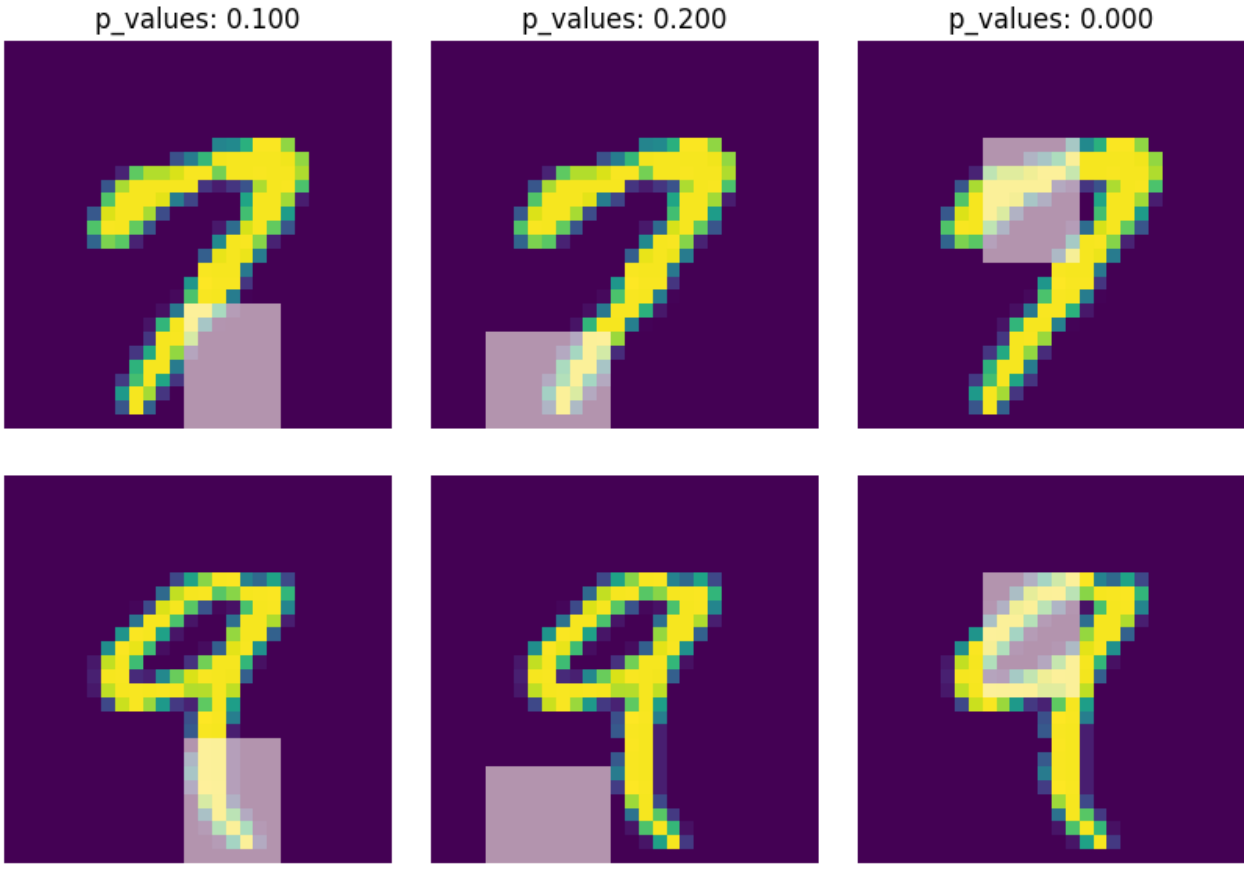
CHAPTER 1

Dnn-Inference

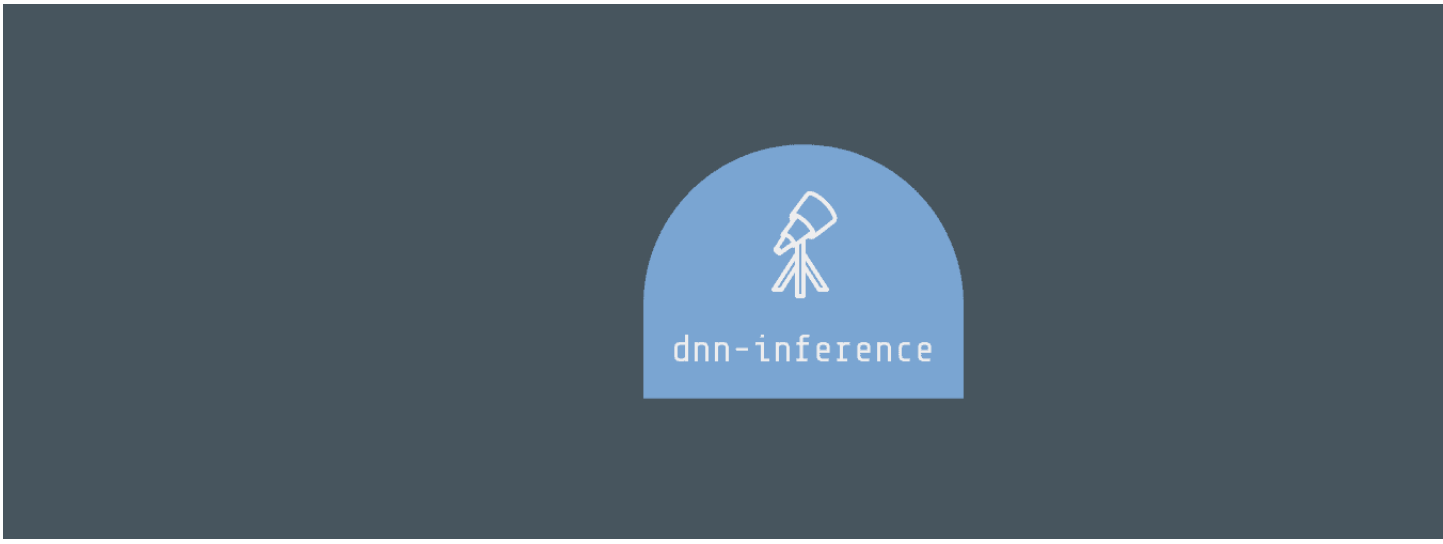


Dnn-Inference is a Python module for hypothesis testing based on deep neural networks.

This project was created by Ben Dai. If there is any question and suggestion please contact me via <bdai@umn.edu>.


p_values: 0.100
p_values: 0.200
p_values: 0.000

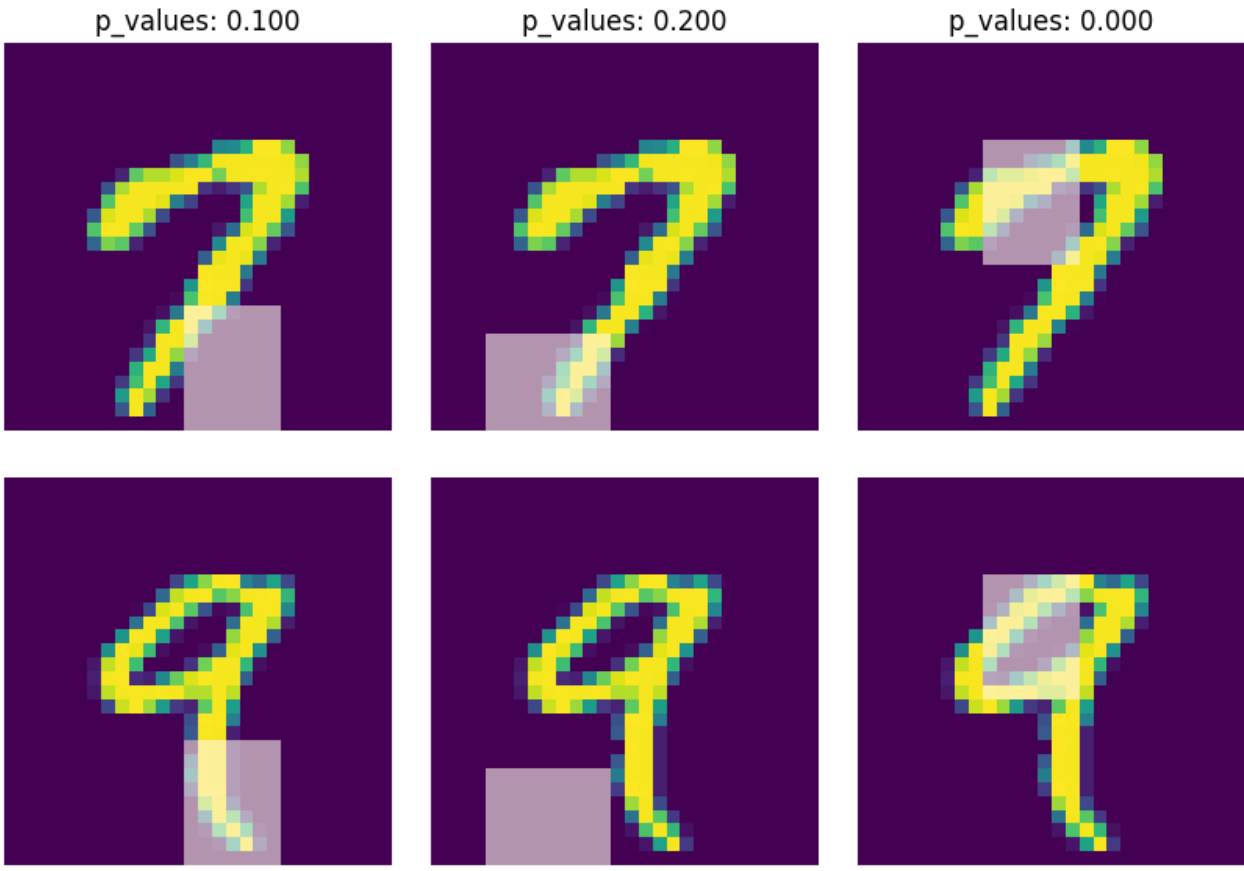
1.1 Dnn-Inference



Dnn-Inference is a Python module for hypothesis testing based on deep neural networks.

Website: https://dnn-inference.readthedocs.io

This project was created by Ben Dai. If there is any problem and suggestion please contact me via <bdai@umn.edu>.



1.1.1 Reference

If you use this code please star the repository and cite the following paper:

```
@misc{dai2021significance,
      title={Significance tests of feature relevance for a blackbox learner},
      author={Ben Dai and Xiaotong Shen and Wei Pan},
      year={2021},
      eprint={2103.04985},
      archivePrefix={arXiv},
      primaryClass={stat.ML}
}
```

1.1.2 Installation

Dependencies

Deep-Inference requires:

- Python
- Numpy
- Keras
- Tensorflow>=1.15
- sklearn
- SciPy

User installation

Install Deep-Inference using `pip`

```
pip install dnn-inference
```

or

```
pip install git+https://github.com/statmlben/dnn-inference.git
```

Source code

You can check the latest sources with the command:

```
git clone https://github.com/statmlben/dnn-inference.git
```

1.1.3 Documentation

DnnT

Class for one-split/two-split test based on deep neural networks.

```
class dnn_inference.DnnT(inf_cov, model, model_mask, change='mask', alpha=.05,␣
↪verbose=0, eva_metric='mse')
```

- Parameters:
  - **inf_cov: {list-like of shape (num of tests, dim of features)}**

    List of covariates/Features under hypothesis testings, one element corresponding to a hypothesis testing.
  - **model: {keras-defined neural network}**

    A neural network for original full dataset
  - **model_mask: {keras-defined neural network}**

    A neural network for masked dataset by masking/changing the features under hypothesis testing
  - **change: {'mask', 'perm'}, default='mask'**

    The way to change the testing features, `'mask'` replaces testing features as zeros, while `'perm'` permutes features via instances.
  - **alpha: float (0,1), default=0.05**

The nominal level of the hypothesis testing

– **verbose: {0, 1}, default=0**

If print the testing results, 1 indicates YES, 0 indicates NO.

– **eva_metric: {'mse', 'zero-one', 'cross-entropy', or custom metric function}**

The evaluation metric, `'mse'` is the l2-loss for regression, `'zero-one'` is the zero-one loss for classification, `'cross-entropy'` is log-loss for classification. It can also be custom metric function as `eva_metric(y_true, y_pred)`.

- **Method**:

```
def testing(self, X, y, cv_num=5, cp='hommel', fit_params, split_params, inf_
↪ratio=None)
```

Method under class `DnnT`, conduct the hypothesis testings according to the given data.

- Parameters:

– **X: {array-like} of shape (n_samples, dim_features)**

Instances matrix/tensor, where n_samples in the number of samples and dim_features is the dimension of the features. If X is vectorized feature, `shape` should be `(#Samples, dim of feaures)` If X is image/matrix data, `shape` should be `(#samples, img_rows, img_cols, channel)`, that is, **X must channel_last image data**. - **y: {array-like} of shape (n_samples,)** Output vector/matrix relative to X.

– **fit_params: {dict of fitting parameters}**

See keras `fit`: (https://keras.rstudio.com/reference/fit.html), including `batch_size`, `epoch`, `callbacks`, `validation_split`, `validation_data`, and so on.

– **split_params: {dict of splitting parameters}**

* **split: {'one-split', 'two-split'}, default='one-split'**

one-split or two-split test statistic.

* **perturb: float, default=None**

Perturb level for the one-split test, if `perturb = None`, then the perturb level is determined by adaptive tunning.

* **num_perm: int, default=100**

Number of permutation for determine the splitting ratio.

* **ratio_grid: list of float (0,1), default=[.2, .4, .6, .8]**

A list of estimation/inference ratios under searching.

* **if_reverse: {0,1}, default=0**

`if_reverse = 0` indicates the loop of `ratio_grid` starts from smallest one to largest one; `if_reverse = 1` indicates the loop of `ratio_grid` starts from largest one to smallest one.

* **perturb_grid: list of float, default=[.01, .05, .1, .5, 1.]**

A list of perturb levels under searching.

* **min_inf: int, default=0**

The minimal size for inference sample.

        * **min_est: int, default=0**

          The minimal size for estimation sample.

        * **ratio_method: {'fuse', 'close'}, default='fuse'**

          The adaptive splitting method to determine the optimal estimation/inference ratios.

        * **cv_num: int, default=1**

          The number of cross-validation to shuffle the estimation/inference samples in adaptive ratio splitting.

        * **cp: {'gmean', 'min', 'hmean', 'Q1', 'hommel', 'cauchy'}, default ='hommel'**

          A method to combine p-values obtained from cross-validation. see (https://arxiv.org/pdf/1212.4966.pdf) for more detail.

        * **verbose: {0,1}, default=1**

    – **cv_num: int, default=1**

      The number of cross-validation to shuffle the estimation/inference samples in testing.

    – **cp: {'gmean', 'min', 'hmean', 'Q1', 'hommel', 'cauchy'}, default ='hommel'**

      A method to combine p-values obtained from cross-validation.

    – **inf_ratio: float, default=None**

      A pre-specific inference sample ratio, if `est_size=None`, then it is determined by adaptive splitting method `metric`.

- Return:

    – **P_value: array of float [0, 1]**

      The p_values for target hypothesis testings.

PermT

Class for permutation testing based on deep neural networks.

Remark: *permutation testing break the dependence of the features, which may lead to incorrect p-values*.

```
class dnn_inference.PermT(inf_cov, model, model_mask, alpha=.05, num_folds=5, num_
↪perm=100, eva_metric='mse', verbose=0)
```

- Parameters:

    – **inf_cov: {list-like of shape (num of tests, dim of features)}**

      List of covariates/Features under hypothesis testings, one element corresponding to a hypothesis testing.

    – **model: {keras-defined neural network}**

      A neural network for original full dataset

    – **model_mask: {keras-defined neural network}**

      A neural network for masked dataset by masking/changing the features under hypothesis testing

    – **alpha: float (0,1), default=0.05**

      The nominal level of the hypothesis testing

  - **num_folds: int, default=5**

    Number of CV-folds to compute the score.

  - **verbose: {0, 1}, default=0**

    If print the testing results, 1 indicates YES, 0 indicates NO.

  - **eva_metric: {'mse', 'zero-one', 'cross-entropy', or custom metric function}**

    The evaluation metric, `'mse'` is the l2-loss for regression, `'zero-one'` is the zero-one loss for classification, `'cross-entropy'` is log-loss for classification. It can also be custom metric function as `eva_metric(y_true, y_pred)`.

- **Method**:

```
def testing(self, X, y, fit_params)
```

Method under class `DnnT`, conduct the hypothesis testings according to the given data.

- Parameters:

  - **X: {array-like}**

    Instances matrix/tensor, where n_samples in the number of samples and dim_features is the dimension of the features. If X is vectorized feature, `shape` should be `(#Samples, dim of feaures)` If X is image/matrix data, `shape` should be `(#samples, img_rows, img_cols, channel)`, that is, **X must channel_last image data**.

  - **y: {array-like} of shape (n_samples,)**

    Output vector/matrix relative to X.

  - **fit_params: {dict of fitting parameters}**

    See keras `fit`: (https://keras.rstudio.com/reference/fit.html), including `batch_size`, `epoch`, `callbacks`, `validation_split`, `validation_data`, and so on.

- Return:

  - **P_value: array of float [0, 1]**

    The p_values for target hypothesis testings.

Example

```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.python.keras import backend as K
import time
from sklearn.model_selection import train_test_split
from keras.optimizers import Adam, SGD
from dnn_inference import DnnT

num_classes = 2

# input image dimensions
img_rows, img_cols = 28, 28
```

(continues on next page)

(continued from previous page)

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
X = np.vstack((x_train, x_test))
y = np.hstack((y_train, y_test))
ind = (y == 9) + (y == 7)
X, y = X[ind], y[ind]
X = X.astype('float32')
X += .01*abs(np.random.randn(14251, 28, 28))
y[y==7], y[y==9] = 0, 1

if K.image_data_format() == 'channels_first':
        X = X.reshape(x.shape[0], 1, img_rows, img_cols)
        input_shape = (1, img_rows, img_cols)
else:
        X = X.reshape(X.shape[0], img_rows, img_cols, 1)
        input_shape = (img_rows, img_cols, 1)

X /= 255.

# convert class vectors to binary class matrices
y = keras.utils.to_categorical(y, num_classes)

K.clear_session()

def cnn():
        model = Sequential()
        model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_
↪shape))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(num_classes, activation='softmax'))
        model.compile(loss=keras.losses.binary_crossentropy, optimizer=keras.
↪optimizers.Adam(0.005), metrics=['accuracy'])
        return model

tic = time.perf_counter()
model, model_mask = cnn(), cnn()

from keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=10,␣
↪restore_best_weights=True)

fit_params = {'callbacks': [es],
                          'epochs': 5,
                          'batch_size': 32,
                          'validation_split': .2,
                          'verbose': 1}

inf_cov = [[np.arange(19,28), np.arange(13,20)], [np.arange(21,28), np.arange(4, 13)],
                   [np.arange(7,16), np.arange(9,16)]]

shiing = DnnT(inf_cov=inf_cov, model=model, model_mask=model_mask, change='mask', eva_
↪metric='zero-one')
```

(continues on next page)

(continued from previous page)

```
p_value_tmp = shiing.testing(X, y, fit_params=fit_params)
toc = time.perf_counter()
print('testing time: %.3f' %(toc-tic))
shiing.visual(X, y)
```

CHAPTER 2

Reference manual

2.1 Python-API

2.1.1 API reference for DnnT class

class dnn_inference.**DnnT**(*inf_feats*, *model*, *model_mask*, *change='mask'*, *alpha=0.05*, *verbose=0*, *eva_metric='mse'*, *cp_path='./dnnT_checkpoints'*)

Class for one-split/two-split test based on deep neural networks.

Parameters

- **inf_feats** (*list-like | shape = (num of tests, dim of features)*) – List of covariates/Features under hypothesis testings, one element corresponding to a hypothesis testing.
- **model** (*{keras-defined neural network}*) – A neural network for original full dataset
- **model_mask** (*{keras-defined neural network}*) – A neural network for masked dataset by masking/changing the features under hypothesis testing
- **change** (*{'mask', 'perm'}, default='mask'*) – The way to change the testing features, `'mask'` replaces testing features as zeros, while `'perm'` permutes features via instances.
- **alpha** (*float (0,1), default=0.05*) – The nominal level of the hypothesis testing
- **verbose** (*{0, 1}, default=0*) – If print the testing results, 1 indicates YES, 0 indicates NO.
- **eva_metric** (*{'mse', 'zero-one', 'cross-entropy', or custom metric function}*) – The evaluation metric, `'mse'` is the l2-loss for regression, `'zero-one'` is the zero-one loss for classification, `'cross-entropy'` is log-loss for classification. It can also be custom metric function as `eva_metric(y_true, y_pred)`.

- **cp_path** (*string, default='./dnnT_checkpoints'*) – The checkpoints path to save the models

adaRatio(*X, y, k=0, fit_params={}, perturb=None, split='one-split', perturb_grid=[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0], ratio_grid=[0.2, 0.4, 0.6, 0.8], if_reverse=0, min_inf=0, min_est=0, ratio_method='fuse', num_perm=100, cv_num=1, cp='hommel', verbose=1*)

Return a data-adaptive splitting ratio and perturbation level.

Parameters

- **X** (*array-like | shape=(n_samples, dim1, dim2, ..)*) – Features.
- **y** (*array-like | shape=(n_samples, dim)*) – Outcomes.
- **k** (*integer, default = 0*) – k-th hypothesized features in inf_feats
- **fit_params** (*dict | shape = dict of fitting parameters*) – See keras `fit`: (https://keras.rstudio.com/reference/fit.html), including `batch_size`, `epoch`, `callbacks`, `validation_split`, `validation_data`.
- **perturb** (*float | default=None*) – Perturb level for the one-split test, if `perturb = None`, then the perturb level is determined by adaptive tunning.
- **split** (*{'one-split', 'two-split'}*) – one-split or two-split test statistic.
- **perturb_grid** (*list of float | default=[01, 05, 1, 5, 1.]*) – A list of perturb levels under searching.
- **ratio_grid** (*list of float (0,1) | default=[2, 4, 6, 8]*) – A list of estimation/inference ratios under searching.
- **if_reverse** (*{0,1} | default = 0*) – `if_reverse = 0` indicates the loop of `ratio_grid` starts from smallest one to largest one; `if_reverse = 1` indicates the loop of `ratio_grid` starts from largest one to smallest one.
- **min_inf** (*integer | default = 0*) – The minimal size for inference sample.
- **min_est** (*integer | default = 0*) – The minimal size for estimation sample.
- **ratio_method** (*{'close', 'fuse'} | default = 'fuse'*) – The adaptive splitting method to determine the optimal estimation/inference ratios.
- **cv_num** (*int, default=1*) – The number of cross-validation to shuffle the estimation/inference samples in adaptive ratio splitting.
- **cp** (*{'gmean', 'min', 'hmean', 'Q1', 'hommel', 'cauchy'} | default = 'hommel'*) – A method to combine p-values obtained from cross-validation. see (https://arxiv.org/pdf/1212.4966.pdf) for more detail.
- **verbose** (*{0,1} | default=1*) – If print the adaptive splitting process.

Returns

- **n_opt** (*integer*) – A reasonable estimation sample size.
- **m_opt** (*integer*) – A reasonable inference sample size.
- **perturb_opt** (*float*) – A reasonable perturbation level.

mask_cov(*X, k=0*)

Return instances with masked k-th hypothesized features.

Parameters

- **X** (*array-like*) – Target instances.
- **k** (*integer, default = 0*) – k-th hypothesized features in inf_feats

perm_cov(*X*, *k=0*)
Return instances with permuted k-th hypothesized features.

Parameters

- **X** (*array-like*) – Target instances.
- **k** (*integer, default = 0*) – k-th hypothesized features in inf_feats

reset_model()
Reset the full and mask network models under class Dnn

save_init()
Save the initialization for full and mask network models under class Dnn

testing(*X*, *y*, *fit_params*, *split_params={}*, *cv_num=5*, *cp='hommel'*, *inf_ratio=None*)
Return p-values for hypothesis testing for inf_feats in class Dnn.

Parameters

- **X** (*{array-like} of shape (n_samples, dim_features)***) –

  Instances matrix/tensor, where n_samples in the number of samples and dim_features is the dimension of If X is vectorized feature, `shape` should be `(#Samples, dim of feaures)` If X is image/matrix data, `shape` should be `(#samples, img_rows, img_cols, channel)`, that is, **X must channel_last image data**. - **y: {array-like} of shape (n_samples,)** Output vector/matrix relative to X.

- **fit_params** (*{dict of fitting parameters}***) – See keras `fit`: (https://keras.rstudio.com/reference/fit.html), including `batch_size`, `epoch`, `callbacks`, `validation_split`, `validation_data`, and so on.

- **split_params** (*{dict of splitting parameters}*) –

  split: {'one-split', 'two-split'}, default='one-split' one-split or two-split test statistic.

  perturb: float, default=None Perturb level for the one-split test, if `perturb = None`, then the perturb level is determined by adaptive tunning.

  num_perm: int, default=100 Number of permutation for determine the splitting ratio.

  ratio_grid: list of float (0,1), default=[.2, .4, .6, .8]** A list of estimation/inference ratios under searching.

  if_reverse: {0,1}, default=0 `if_reverse = 0` indicates the loop of `ratio_grid` starts from smallest one to largest one; `if_reverse = 1` indicates the loop of `ratio_grid` starts from largest one to smallest one.

  perturb_grid: list of float, default=[.01, .05, .1, .5, 1.]** A list of perturb levels under searching.

  min_inf: int, default=0 The minimal size for inference sample.

  min_est: int, default=0 The minimal size for estimation sample.

  ratio_method: {'fuse', 'close'}, default='fuse' The adaptive splitting method to determine the optimal estimation/inference ratios.

  cv_num: int, default=1 The number of cross-validation to shuffle the estimation/inference samples in adaptive ratio splitting.

  cp: {'gmean', 'min', 'hmean', 'Q1', 'hommel', 'cauchy'}, default ='hommel'** A method to combine p-values obtained from cross-validation. see (https://arxiv.org/pdf/1212.4966.pdf) for more detail.

verbose: {0,1}, default=1**

- **cv_num** (*int, default=5*) – The number of cross-validation to shuffle the estimation/inference samples in testing.
- **cp** (*{'gmean', 'min', 'hmean', 'Q1', 'hommel', 'cauchy'}, default ='hommel'***) – A method to combine p-values obtained from cross-validation.
- **inf_ratio** (*float, default=None***) – A pre-specific inference sample ratio, if `est_size=None`, then it is determined by adaptive splitting method `metric`.

Returns **P_value** – The p_values for target hypothesis testings.

Return type array of float [0, 1]**

visual(*X*, *y*, *plt_params={'alpha': 0.6, 'cmap': 'RdBu'}*, *plt_mask_params={'alpha': 0.6, 'cmap': 'RdBu'}*)

Visualization for the inference results based on one illustrative example

Parameters

- **X** (*array-like*) – demo instances.
- **y** (*array-like*) – demo labels
- **plt_params** (*dict*) – dictory of parameters for the imshow for original image see: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.imshow.html
- **plt_mask_params** (*dict*) – dictory of parameters for the imshow for mask see: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.imshow.html

2.1.2 API reference for PermT class

class dnn_inference.**PermT**(*inf_feats*, *model*, *model_perm*, *alpha=0.05*, *num_folds=5*, *num_perm=100*, *verbose=0*, *eva_metric='mse'*)

Index