Permetrics

Release 2.0.0

Thieu

Mar 09, 2024

QUICK START

PerMetrics is a python library for performance metrics of machine learning models. We aim to implement all performance metrics for problems such as regression, classification, clustering, ... problems. Helping users in all field access metrics as fast as possible

- **Free software:** GNU General Public License (GPL) V3 license
- **Total metrics**: 111 (47 regression metrics, 20 classification metrics, 44 clustering metrics)
- **Documentation:** https://permetrics.readthedocs.io/en/latest/
- **Python versions:** >= 3.7.x
- **Dependencies:** numpy, scipy

CHAPTER

ONE

INSTALLATION

- Install the current PyPI release:

```
$ pip install permetrics==2.0.0
```

- Install directly from source code:

```
$ git clone https://github.com/thieu1995/permetrics.git
$ cd permetrics
$ python setup.py install
```

- In case, you want to install the development version from Github:

```
$ pip install git+https://github.com/thieu1995/permetrics
```

After installation, you can import Permetrics as any other Python module:

```
$ python
>>> import permetrics
>>> permetrics.__version__
```

Let's go through some examples.

CHAPTER

TWO

EXAMPLES

There are several ways you can use a performance metrics in this library. However, the most used are these two ways: functional-based and object-oriented based programming. We will go through detail of how to use 3 main type of metrics (regression, classification, and clustering) with these two methods.

2.1 Regression Metrics

2.1.1 Functional Style

- This is a traditional way to call a specific metric you want to use. Everytime you want to use a metric, you need to pass y_true and y_pred

```
## 1. Import packages, classes
## 2. Create object
## 3. From object call function and use

import numpy as np
from permetrics import RegressionMetric

y_true = np.array([3, -0.5, 2, 7, 5, 6])
y_pred = np.array([2.5, 0.0, 2, 8, 5, 6])

evaluator = RegressionMetric()

## 3.1 Call specific function inside object, each function has 2 names like below
rmse_1 = evaluator.RMSE(y_true, y_pred)
rmse_2 = evaluator.root_mean_squared_error(y_true, y_pred)
print(f"RMSE: {rmse_1}, {rmse_2}")

mse = evaluator.MSE(y_true, y_pred)
mae = evaluator.MAE(y_true, y_pred)
print(f"MSE: {mse}, MAE: {mae}")
```

2.1.2 Object-Oriented Style

- This is modern and better way to use metrics. You only need to pass y_true, y_pred one time when creating metric object.
- After that, you can get the value of any metrics without passing y_true, y_pred

```
## 1. Import packages, classes
## 2. Create object
## 3. From object call function and use

import numpy as np
from permetrics import RegressionMetric

y_true = np.array([3, -0.5, 2, 7, 5, 6])
y_pred = np.array([2.5, 0.0, 2, 8, 5, 6])

evaluator = RegressionMetric(y_true, y_pred)

## Get the result of any function you want to
rmse = evaluator.RMSE()
mse = evaluator.MSE()
mae = evaluator.MAE()

print(f"RMSE: {rmse}, MSE: {mse}, MAE: {mae}")
```

2.1.3 Multiple Metrics Style

- To reduce coding time when using multiple metrics. There are few ways to do it with Permetrics by using *OOP style*

```
import numpy as np
from permetrics import RegressionMetric

y_true = np.array([3, -0.5, 2, 7, 5, 6])
y_pred = np.array([2.5, 0.0, 2, 8, 5, 6])

evaluator = RegressionMetric(y_true, y_pred)

## Define list of metrics you want to use
list_metrics = ["RMSE", "MAE", "MAPE", "NSE"]

## 1. Get list metrics by using loop
list_results = []
for metric in list_metrics:
        list_results.append( evaluator.get_metric_by_name(metric) )
print(list_results)


## 2. Get list metrics by using function
dict_result_2 = evaluator.get_metrics_by_list_names(list_metrics)
print(dict_result_2)
```

(continues on next page)

(continued from previous page)

```
## 3. Get list metrics by using function and parameters
dict_metrics = {
    "RMSE": None,
    "MAE": None,
    "MAPE": None,
    "NSE": None,
}
dict_result_3 = evaluator.get_metrics_by_dict(dict_metrics)
print(dict_result_3)
```

2.1.4 Multiple Outputs for Multiple Metrics

- The Scikit-learn library is limited with multi-output metrics, but Permetrics can produce multi-output for all of metrics

```
import numpy as np
from permetrics import RegressionMetric

## This y_true and y_pred have 4 columns, 4 outputs
y_true = np.array([ [3, -0.5, 2, 7],
                    [5, 6, -0.3, 9],
                    [-11, 23, 8, 3.9] ])

y_pred = np.array([ [2.5, 0.0, 2, 8],
                    [5.2, 5.4, 0, 9.1],
                    [-10, 23, 8.2, 4] ])

evaluator = RegressionMetric(y_true, y_pred)

## 1. By default, all metrics can automatically return the multi-output results
# rmse = evaluator.RMSE()
# print(rmse)

## 2. If you want to take mean of all outputs, can set the parameter: multi-output =
↪"mean"
# rmse_2 = evaluator.RMSE(multi_output="mean")
# print(rmse_2)

## 3. If you want a specific metric has more important than other, you can set weight␣
↪for each output.
# rmse_3 = evaluator.RMSE(multi_output=[0.5, 0.05, 0.1, 0.35])
# print(rmse_3)


## Get multiple metrics with multi-output or single-output by parameters


## 1. Get list metrics by using list_names
list_metrics = ["RMSE", "MAE", "MSE"]
list_paras = [
```

(continues on next page)

(continued from previous page)

```
    {"multi_output": "mean"},
    {"multi_output": [0.5, 0.2, 0.1, 0.2]},
    {"multi_output": "raw_values"}
]
dict_result_1 = evaluator.get_metrics_by_list_names(list_metrics, list_paras)
print(dict_result_1)


## 2. Get list metrics by using dict_metrics
dict_metrics = {
    "RMSE": {"multi_output": "mean"},
    "MAE": {"multi_output": "raw_values"},
    "MSE": {"multi_output": [0.5, 0.2, 0.1, 0.2]},
}
dict_result_2 = evaluator.get_metrics_by_dict(dict_metrics)
print(dict_result_2)
```

2.2 Classification Metrics

2.2.1 Functional Style

- This is a traditional way to call a specific metric you want to use. Everytime you want to use a metric, you need to pass y_true and y_pred

```
## 1. Import packages, classes
## 2. Create object
## 3. From object call function and use

import numpy as np
from permetrics import ClassificationMetric

y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

evaluator = ClassificationMetric()

## 3.1 Call specific function inside object, each function has 2 names like below
ps1 = evaluator.precision_score(y_true, y_pred)
ps2 = evaluator.PS(y_true, y_pred)
ps3 = evaluator.PS(y_true, y_pred)
print(f"Precision: {ps1}, {ps2}, {ps3}")

recall = evaluator.recall_score(y_true, y_pred)
accuracy = evaluator.accuracy_score(y_true, y_pred)
print(f"recall: {recall}, accuracy: {accuracy}")
```

2.2.2 Object-Oriented Style

- This is modern and better way to use metrics. You only need to pass y_true, y_pred one time when creating metric object.
- After that, you can get the value of any metrics without passing y_true, y_pred

```
import numpy as np
from permetrics import ClassificationMetric

y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

evaluator = ClassificationMetric(y_true, y_pred)

## Get the result of any function you want to

hamming_score = evaluator.hamming_score()
mcc = evaluator.matthews_correlation_coefficient()
specificity = evaluator.specificity_score()
print(f"HL: {hamming_score}, MCC: {mcc}, specificity: {specificity}")
```

2.3 Clustering Metrics

Note that, this type of metrics is kinda differ from regression and classification. There are two type of clustering metrics include internal and external metrics, each serving a different purpose:

1. Internal Metrics:
   - Objective: Internal metrics evaluate the quality of clusters based on the data itself without relying on external information or ground truth labels.
   - Example Metrics:
     - Silhouette Score: Measures how well-separated clusters are.
     - Davies-Bouldin Score: Computes the compactness and separation of clusters.
     - Inertia (within-cluster sum of squares): Measures how far points within a cluster are from the cluster's centroid.
2. External Metrics:
   - Objective: External metrics assess the quality of clusters by comparing them to some external criterion, often ground truth labels or known groupings.
   - Example Metrics:
     - Adjusted Rand Index (ARI): Measures the similarity between true and predicted clusters, adjusted for chance.
     - Normalized Mutual Information Index (NMII): Measures the mutual information between true and predicted clusters, normalized.
     - Fowlkes-Mallows Index: Computes the geometric mean of precision and recall between true and predicted clusters.

While internal metrics provide insights into the structure of the data within the clusters, external metrics help evaluate clustering performance against a known or expected structure, such as labeled data. The choice between internal and external metrics depends on the availability of ground truth information and the specific goals of the clustering analysis.

To clearly distinguish between internal and external clustering metrics, we use specific suffixes in their function names. Using the suffix *index* can indicate internal clustering metrics, while using the suffix *score* can indicate external clustering metrics. This naming convention makes it easier for users to differentiate between the two types of metrics and facilitates their usage.

By following this convention, users can easily identify whether a metric is designed for evaluating the quality of clusters within a dataset (internal) or for comparing clusters to external reference labels or ground truth (external). This distinction is important because internal and external metrics serve different purposes and have different interpretations.

2.3.1 Functional Style

- External clustering metrics

```
import numpy as np
from permetrics import ClusteringMetric

y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

evaluator = ClusteringMetric()

ps1 = evaluator.mutual_info_score(y_true, y_pred)
ps2 = evaluator.MIS(y_true, y_pred)
print(f"Mutual Information score: {ps1}, {ps2}")

homogeneity = evaluator.homogeneity_score(y_true, y_pred)
completeness  = evaluator.CS(y_true, y_pred)
print(f"Homogeneity: {homogeneity}, Completeness : {completeness}")
```

- Internal clustering metrics

```
import numpy as np
from permetrics import ClusteringMetric
from sklearn.datasets import make_blobs

# generate sample data
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
y_pred = np.random.randint(0, 4, size=300)

evaluator = ClusteringMetric()

evaluator.BHI(X=X, y_pred=y_pred)
evaluator.BRI(X=X, y_pred=y_pred)
```

2.3.2 Object-Oriented Style

- External clustering metrics

```
import numpy as np
from permetrics import ClusteringMetric

y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

evaluator = ClusteringMetric(y_true, y_pred)

## Get the result of any function you want to
x1 = evaluator.kulczynski_score()
x2 = evaluator.mc_nemar_score()
x3 = evaluator.rogers_tanimoto_score()
print(f"Kulczynski: {x1}, Mc Nemar: {x2}, Rogers Tanimoto: {x3}")
```

- Internal clustering metrics

```
import numpy as np
from permetrics import ClusteringMetric
from sklearn.datasets import make_blobs

# generate sample data
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
y_pred = np.random.randint(0, 4, size=300)

evaluator = ClusteringMetric(X=X, y_pred=y_pred)

evaluator.BHI()
evaluator.CHI()
evaluator.DBI()

## Or
print(evaluator.get_metrics_by_list_names(["BHI", "CHI", "XBI", "BRI", "DBI", "DRI", "DI
↪", "KDI", "LDRI", "LSRI", "SI"]))
```

CHAPTER

THREE

REGRESSION METRICS

STT	Metric	Metric Fullname	Characteristics
1	EVS	Explained Variance Score	Bigger is better (Best = 1), Range=(-inf, 1.0]
2	ME	Max Error	Smaller is better (Best = 0), Range=[0, +inf)
3	MBE	Mean Bias Error	Best = 0, Range=(-inf, +inf)
4	MAE	Mean Absolute Error	Smaller is better (Best = 0), Range=[0, +inf)
5	MSE	Mean Squared Error	Smaller is better (Best = 0), Range=[0, +inf)
6	RMSE	Root Mean Squared Error	Smaller is better (Best = 0), Range=[0, +inf)
7	MSLE	Mean Squared Log Error	Smaller is better (Best = 0), Range=[0, +inf)
8	MedAE	Median Absolute Error	Smaller is better (Best = 0), Range=[0, +inf)
9	MRE / MRB	Mean Relative Error / Mean Relative Bias	Smaller is better (Best = 0), Range=[0, +inf)
10	MPE	Mean Percentage Error	Best = 0, Range=(-inf, +inf)
11	MAPE	Mean Absolute Percentage Error	Smaller is better (Best = 0), Range=[0, +inf)
12	SMAPE	Symmetric Mean Absolute Percentage Error	Smaller is better (Best = 0), Range=[0, 1]
13	MAAPE	Mean Arctangent Absolute Percentage Error	Smaller is better (Best = 0), Range=[0, +inf)
14	MASE	Mean Absolute Scaled Error	Smaller is better (Best = 0), Range=[0, +inf)
15	NSE	Nash-Sutcliffe Efficiency Coefficient	Bigger is better (Best = 1), Range=(-inf, 1]
16	NNSE	Normalized Nash-Sutcliffe Efficiency Coefficient	Bigger is better (Best = 1), Range=[0, 1]
17	WI	Willmott Index	Bigger is better (Best = 1), Range=[0, 1]
18	R / PCC	Pearson's Correlation Coefficient	Bigger is better (Best = 1), Range=[-1, 1]
19	AR / APCC	Absolute Pearson's Correlation Coefficient	Bigger is better (Best = 1), Range=[-1, 1]
20	RSQ/R2S	(Pearson's Correlation Index) ^ 2	Bigger is better (Best = 1), Range=[0, 1]
21	R2 / COD	Coefficient of Determination	Bigger is better (Best = 1), Range=(-inf, 1]
22	AR2 / ACOD	Adjusted Coefficient of Determination	Bigger is better (Best = 1), Range=(-inf, 1]
23	CI	Confidence Index	Bigger is better (Best = 1), Range=(-inf, 1]
24	DRV	Deviation of Runoff Volume	Smaller is better (Best = 1.0), Range=[1, +inf)
25	KGE	Kling-Gupta Efficiency	Bigger is better (Best = 1), Range=(-inf, 1]
26	GINI	Gini Coefficient	Smaller is better (Best = 0), Range=[0, +inf)
27	GINI_WIKI	Gini Coefficient on Wikipage	Smaller is better (Best = 0), Range=[0, +inf)
28	PCD	Prediction of Change in Direction	Bigger is better (Best = 1.0), Range=[0, 1]
29	CE	Cross Entropy	Range(-inf, 0], Can't give comment about this
30	KLD	Kullback Leibler Divergence	Best = 0, Range=(-inf, +inf)
31	JSD	Jensen Shannon Divergence	Smaller is better (Best = 0), Range=[0, +inf)
32	VAF	Variance Accounted For	Bigger is better (Best = 100%), Range=(-inf, 100%]
33	RAE	Relative Absolute Error	Smaller is better (Best = 0), Range=[0, +inf)
34	A10	A10 Index	Bigger is better (Best = 1), Range=[0, 1]
35	A20	A20 Index	Bigger is better (Best = 1), Range=[0, 1]
36	A30	A30 Index	Bigger is better (Best = 1), Range=[0, 1]
37	NRMSE	Normalized Root Mean Square Error	Smaller is better (Best = 0), Range=[0, +inf)

continues on next page

Table 1 – continued from previous page

STT	Metric	Metric Fullname	Characteristics
38	RSE	Residual Standard Error	Smaller is better (Best = 0), Range=[0, +inf)
39	RE / RB	Relative Error / Relative Bias	Best = 0, Range=(-inf, +inf)
40	AE	Absolute Error	Best = 0, Range=(-inf, +inf)
41	SE	Squared Error	Smaller is better (Best = 0), Range=[0, +inf)
42	SLE	Squared Log Error	Smaller is better (Best = 0), Range=[0, +inf)
43	COV	Covariance	Bigger is better (No best value), Range=(-inf, +inf)
44	COR	Correlation	Bigger is better (Best = 1), Range=[-1, +1]
45	EC	Efficiency Coefficient	Bigger is better (Best = 1), Range=(-inf, +1]
46	OI	Overall Index	Bigger is better (Best = 1), Range=(-inf, +1]
47	CRM	Coefficient of Residual Mass	Smaller is better (Best = 0), Range=(-inf, +inf)

From now on:

- $\hat{y}$ is the estimated target output,
- y is the corresponding (correct) target output.
- $\hat{Y}$ is the whole estimated target output ,
- Y is the corresponding (correct) target output.
- $mean(\hat{Y})$ is the mean of whole estimated target output ,
- $mean(Y)$ is the mean of whole (correct) target output.

3.1 EVS - Explained Variance Score

$$EVS = 1 - \frac{Var\{y_{true} - y_{pred}\}}{Var\{y_{true}\}}$$

- Link to equation

The given math formula defines the explained variance score (EVS) [1], which is a metric used in regression analysis to evaluate the performance of a model. The formula computes the ratio of the variance of the difference between the true values y_true and the predicted values y_pred to the variance of the true values y_true.

The resulting score ranges between -infinity and 1, with a score of 1 indicating a perfect match between the true and predicted values and a score of 0 indicating that the model does not perform better than predicting the mean of the true values.

A higher value of EVS indicates a better performance of the model. Best possible score is 1.0, greater values are better. Range = (-inf, 1.0].

Example to use EVS metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.explained_variance_score())
```

(continues on next page)

(continued from previous page)

```
## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.EVS(multi_output="raw_values"))
```

3.2 ME - Max Error

$$\text{ME}(y, \hat{y}) = max(|y_i - \hat{y}_i|)$$

Latex equation code:

```
\text{ME}(y, \hat{y}) = max(| y_i - \hat{y}_i |)
```

The *max_error* function computes the maximum residual error , a metric that captures the worst case error between the predicted value and the true value. In a perfectly fitted single output regression model, max_error would be 0 on the training set and though this would be highly unlikely in the real world, this metric shows the extent of error that the model had when it was fitted.

- Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Example to use ME metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.max_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.ME(multi_output="raw_values"))
```

3.3 MAE - Mean Absolute Error

$$\text{MAE}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1} |y_i - \hat{y}_i|}{N}$$

Mean Absolute Error (MAE) [2] is a statistical measure used to evaluate the accuracy of a forecasting model, such as a regression model or a time series model. It measures the average magnitude of the errors between the predicted values and the actual values in the units of the response variable. The MAE is calculated as the average of the absolute differences between the predicted values and the actual values. In other words, it is the mean of the absolute errors. Best possible score is 0.0, smaller value is better. Range = [0, +inf)

The MAE is a widely used measure of forecast accuracy because it is easy to understand and interpret. A lower MAE indicates better forecast accuracy. However, like the RMSE, the MAE is not normalized and is dependent on the scale of the response variable, making it difficult to compare the MAE values across different datasets with different scales.

Example to use MAE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_absolute_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MAE(multi_output="raw_values"))
```

3.4 MSE - Mean Squared Error

$$\text{MSE}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}$$

Latex equation code:

```
\text{MSE}(y, \hat{y}) = \frac{\sum_{i=0}^{N - 1} (y_i - \hat{y}_i)^2}{N}
```

- Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- MSE: a risk metric corresponding to the expected value of the squared (quadratic) error or loss.
- Link to equation

Example to use MSE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_squared_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])
```

(continues on next page)

(continued from previous page)

```
evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MSE(multi_output="raw_values"))
```

3.5 BE - Mean Bias Error

$$\text{MBE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(f_i - y_i)$$

Latex equation code:

```
\text{MBE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n}(f_i - y_i)
```

The Mean Bias Error (MBE) [3] is a statistical measure used to assess the bias of a forecasting model. The MBE measures the average difference between the forecasted and actual values, without considering their direction.

The MBE is expressed in the same units as the forecasted and actual values, and a best possible score of 0.0 indicates no bias in the forecasting model. The MBE has a range of (-infinity, +infinity), with a positive MBE indicating that the forecasted values are, on average, larger than the actual values, and a negative MBE indicating the opposite.

The MBE is a useful measure to evaluate the systematic errors of a forecasting model, such as overestimation or underestimation of the forecasted values. However, it does not provide information about the magnitude or direction of the individual errors, and it should be used in conjunction with other statistical measures, such as the Mean Absolute Error (MAE), to provide a more comprehensive evaluation of the forecasting model's accuracy.

It is important to note that the MBE is sensitive to outliers and may not be appropriate for data with non-normal distributions or extreme values. In such cases, other measures, such as the Median Bias Error (MBE), may be more appropriate.

Example to use MBE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_bias_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MBE(multi_output="raw_values"))
```

3.6 RMSE - Root Mean Square Error

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}}$$

Latex equation code:

```
\text{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=0}^{N - 1} (y_i - \hat{y}_i)^2}{N}}
```

Root Mean Square Error (RMSE) [4] is a statistical measure that is often used to evaluate the accuracy of a forecasting model, such as a regression model or a time series model. It measures the difference between the predicted values and the actual values in the units of the response variable.

The RMSE is calculated as the square root of the average of the squared differences between the predicted values and the actual values. In other words, it is the square root of the mean of the squared errors. Best possible score is 0.0, smaller value is better. Range = [0, +inf)

The RMSE is a widely used measure of forecast accuracy because it is sensitive to both the magnitude and direction of the errors. A lower RMSE indicates better forecast accuracy. However, it has a drawback that it is not normalized, meaning that it is dependent on the scale of the response variable. Therefore, it is difficult to compare the RMSE values across different datasets with different scales.

The RMSE is commonly used in various fields, including finance, economics, and engineering, to evaluate the performance of forecasting models. It is often used in conjunction with other measures, such as the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE), to provide a more comprehensive evaluation of the model's performance.

Example to use RMSE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.root_mean_squared_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.RMSE(multi_output="raw_values"))
```

3.7 MSLE - Mean Squared Logarithmic Error

$$\text{MSLE}(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N-1}(\log_e(1+y_i) - \log_e(1+\hat{y}_i))^2$$

Latex equation code:

```
\text{MSLE}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N - 1} (\log_e (1 + y_i) - \log_e (1 + \hat{y}_i) )^2
```

Where $\log_e(x)$ means the natural logarithm of x. This metric is best to use when targets having exponential growth, such as population counts, average sales of a commodity over a span of years etc. Note that this metric penalizes an under-predicted estimate greater than an over-predicted estimate.

The Mean Squared Logarithmic Error (MSLE) [5] is a statistical measure used to evaluate the accuracy of a forecasting model, particularly when the data has a wide range of values. It measures the average of the squared differences between the logarithms of the predicted and actual values.

The logarithmic transformation used in the MSLE reduces the impact of large differences between the actual and predicted values and provides a better measure of the relative errors between the two values. The MSLE is always a positive value, with a smaller MSLE indicating better forecast accuracy.

- The MSLE is commonly used in applications such as demand forecasting, stock price prediction, and sales forecasting, where the data has a wide range of

values and the relative errors are more important than the absolute errors. + It is important to note that the MSLE is not suitable for data with negative values or zero values, as the logarithm function is not defined for these values. + Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Example to use MSLE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_squared_log_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6], [1, 2]])
y_pred = array([[0, 2], [-1, 2], [8, -5], [1.1, 1.9]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MSLE(multi_output="raw_values"))
```

3.8 MedAE - Median Absolute Error

$$\text{MedAE}(y, \hat{y}) = \text{median}(\mid y_1 - \hat{y}_1 \mid, \ldots, \mid y_n - \hat{y}_n \mid)$$

Latex equation code:

```
\text{MedAE}(y, \hat{y}) = \text{median}(\mid y_1 - \hat{y}_1 \mid, \ldots, \mid y_n - \
↪hat{y}_n \mid)
```

The Median Absolute Error (MedAE) [6] is particularly interesting because it is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the prediction.

- Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Example to use MedAE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.max_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.ME(multi_output="raw_values"))
```

3.9 MRE - Mean Relative Error

$$\text{MRE}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{|y_i - \hat{y}_i|}{|y_i|}$$

Latex equation code:

```
\text{MRE}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N - 1} \frac{|y_i - \hat{y}_i|}{|y_i|}
```

- Mean Relative Error (MRE) or Mean Relative Bias (MRB)
- Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Example to use MRE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
```

(continues on next page)

(continued from previous page)

```
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_relative_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MRE(multi_output="raw_values"))
```

3.10 MPE - Mean Percentage Error

$$\text{MPE}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N-1} \frac{y_i - \hat{y}_i}{y_i}.$$

Latex equation code:

```
\text{MPE}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N - 1} \frac{y_i - \hat{y}_i}{y_i}.
```

- Mean Percentage Error (MPE): Best possible score is 0.0. Range = (-inf, +inf)
- Link to equation

Example to use MPE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_percentage_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MPE(multi_output="raw_values"))
```

3.11 MAPE - Mean Absolute Percentage Error

$$\text{MAPE}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N-1} \frac{|y_i - \hat{y}_i|}{|y_i|}$$

The Mean Absolute Percentage Error (MAPE) [7] is a statistical measure of the accuracy of a forecasting model, commonly used in business and economics. The MAPE measures the average percentage difference between the forecasted and actual values, with a lower MAPE indicating better forecast accuracy.

The MAPE is expressed as a percentage, and a commonly used benchmark for a good forecast model is a MAPE of less than 20%. However, the benchmark may vary depending on the specific application and industry. The MAPE has a range of [0, +infinity), with a best possible score of 0.0, indicating perfect forecast accuracy. A larger MAPE indicates a larger average percentage difference between the forecasted and actual values, with infinite MAPE indicating a complete failure of the forecasting model. + Link equation

Example to use MAPE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_absolute_percentage_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MAPE(multi_output="raw_values"))
```

3.12 SMAPE - Symmetric Mean Absolute Percentage Error

$$\text{SMAPE}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N-1} \frac{2 * |y_i - \hat{y}_i|}{|y| + |\hat{y}|}$$

Latex equation code:

```
\text{SMAPE}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N - 1} \frac{ 2*|y_i - \hat{y}_i|}
↪{|y| + |\hat{y}|}
```

Symmetric Mean Absolute Percentage Error (SMAPE) [8], which is an accuracy measure commonly used in forecasting and time series analysis.

Given the actual values y and the predicted values y_hat, the SMAPE is calculated as the average of the absolute percentage errors between the two, where each error is weighted by the sum of the absolute values of the actual and predicted values.

The resulting score ranges between 0 and 1, where a score of 0 indicates a perfect match between the actual and predicted values, and a score of 1 indicates no match at all. A smaller value of SMAPE is better, and it is often multiplied by 100% to obtain the percentage error. Best possible score is 0.0, smaller value is better. Range = [0, 1].

- Link to equation

Example to use SMAPE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.symmetric_mean_absolute_percentage_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.SMAPE(multi_output="raw_values"))
```

3.13 MAAPE - Mean Arctangent Absolute Percentage Error

$$MAAPE = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{A_i - F_i}{A_i} \right| \arctan\left(\frac{A_i - F_i}{A_i} \right)$$

where A_i is the i-th actual value, F_i is the i-th forecasted value, and n is the number of observations.

The Mean Arctangent Absolute Percentage Error (MAAPE) is a statistical measure used to evaluate the accuracy of a forecasting model. It was introduced by Armstrong in 1985 as an alternative to the Mean Absolute Percentage Error (MAPE) that avoids the issue of dividing by zero when the actual value is zero.

The MAAPE is calculated as the average of the arctangent of the absolute percentage errors between the forecasted and actual values. The arctangent function is used to transform the percentage errors into a bounded range of -pi/2 to pi/2, which is more suitable for averaging than the unbounded range of the percentage errors.

The MAAPE measures the average magnitude and direction of the errors between the forecasted and actual values, with values ranging from 0% to 100%. A lower MAAPE indicates better forecast accuracy. The MAAPE is commonly used in time series forecasting applications, such as sales forecasting, stock price prediction, and demand forecasting.

- Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- Link to equation

Example to use MAAPE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_arctangent_absolute_percentage_error())
```

(continues on next page)

(continued from previous page)

```
## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MAAPE(multi_output="raw_values"))
```

3.14 MASE - Mean Absolute Scaled Error

$$\text{MASE}(y, \hat{y}) = \frac{\frac{1}{N}\sum_{i=0} N-1|y_i - \hat{y}_i|}{\frac{1}{N-1}\sum_{i=1}^{N-1}|y_i - y_{i-1}|}$$

Latex equation code:

```
\text{MASE}(y, \hat{y}) = \frac{ \frac{1}{N} \sum_{i=0}{N-1} |y_i - \hat{y_i}| }{ \frac
↪{1}{N-1} \sum_{i=1}^{N-1} |y_i - y_{i-1}| }
```

- Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- m = 1 for non-seasonal data, m > 1 for seasonal data
- Link to equation

Example to use MASE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.mean_absolute_scaled_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.MASE(multi_output="raw_values"))
```

3.15 NSE - Nash-Sutcliffe Efficiency

$$\text{NSE}(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1}(y_i - mean(y))^2}$$

Latex equation code:

```
\text{NSE}(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{N - 1} (y_i - \hat{y_i})^2}{ \sum_{i=0}^
↪{N - 1} (y_i - mean(y))^2}
```

The NSE [9] is calculated as the ratio of the mean squared error between the observed and simulated streamflow to the variance of the observed streamflow. The NSE ranges between -inf and 1, with a value of 1 indicating perfect agreement between the observed and simulated streamflow. + Link to equation

Example to use NSE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.nash_sutcliffe_efficiency())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.NSE(multi_output="raw_values"))
```

3.16 NNSE - Normalized NSE

$$\text{NNSE}(y, \hat{y}) = \frac{1}{2 - NSE}$$

Latex equation code:

```
\text{E}(y, \hat{y}) = \frac{1}{2 - NSE}
```

The Normalized NSE (NNSE) [10] is a statistical measure used to evaluate the performance of hydrological models in simulating streamflow. It is a variant of the Nash-Sutcliffe Efficiency (NSE), which is a widely used measure of model performance in hydrology.

The NNSE accounts for the variability in the observed streamflow and provides a more objective measure of model performance than the NSE alone. The NNSE is commonly used in hydrology and water resources engineering to evaluate the performance of hydrological models in simulating streamflow and to compare the performance of different models.

- Normalize Nash-Sutcliffe Efficiency (NNSE): Best possible score is 1.0, greater value is better. Range = [0, 1]
- Link to equation

Example to use NNSE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.normalized_nash_sutcliffe_efficiency())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6], [1, 2], [2.1, 2.2], [3.4, 5.5]])
y_pred = array([[0, 2], [-1, 2], [8, -5], [1.1, 1.9], [2.0, 2.3], [3.0, 4.2]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.NNSE(multi_output="raw_values"))
```

3.17 WI - Willmott Index

$$\text{WI}(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{N-1}(\hat{y}_i - y_i)^2}{\sum_{i=0}^{N-1}(|\hat{y}_i - mean(y)| + |y_i - mean(y)|)^2}$$

Latex equation code:

```
\text{WI}(y, \hat{y}) = 1 - \frac{ \sum_{i=0}^{N - 1} (\hat{y_i} - y_i)^2 }{ \sum_{i=0}^
↪{N - 1} (|\hat{y_i} - mean(y)| + |y_i - mean(y)|)^2}
```

The Willmott Index (WI) [11] is a statistical measure used to evaluate the performance of a forecasting model, particularly in the context of hydrological or climate-related variables. The WI compares the accuracy of a model to the accuracy of a reference model that simply predicts the mean value of the observed variable. Best possible score is 1.0, bigger value is better. Range = [0, 1]

The WI ranges between 0 and 1, with a value of 1 indicating perfect agreement between the predicted and observed values. A value of 0 indicates that the predicted values are no better than predicting the mean of the observed values.

The WI is commonly used in hydrology and climate-related fields to evaluate the accuracy of models that predict variables such as precipitation, temperature, and evapotranspiration. It is a useful tool for comparing the performance of different models or different methods of estimating a variable.

- Link to equation

Example to use WI metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.willmott_index())
```

(continues on next page)

(continued from previous page)

```
## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.WI(multi_output="raw_values"))
```

3.18 R - Pearson's Correlation Index

$$\mathrm{R}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1}((y_i - mean(y)) * (\hat{y}_i - mean(\hat{y})))}{\sqrt{\sum_{i=0}^{N-1}(y_i - mean(y))^2} * \sqrt{\sum_{i=0}^{N-1}(\hat{y}_i - mean(\hat{y}))^2}}$$

Latex equation code:

```
\text{R}(y, \hat{y}) = \frac{ \sum_{i=0}^{N - 1} ((y_i - mean(y))*(\hat{y_i} - mean(\hat
↪{y}))) }{ \sqrt{ \sum_{i=0}^{N - 1} (y_i - mean(y))^2} *\sqrt{ \sum_{i=0}^{N - 1} (\hat
↪{y_i} - mean(\hat{y}))^2} }
```

Pearson's Correlation Index, also known as Pearson's correlation coefficient [12], is a statistical measure that quantifies the strength and direction of the linear relationship between two variables. It is denoted by the symbol "r", and ranges between -1 and +1.

A value of +1 indicates a perfect positive linear relationship between the two variables, while a value of -1 indicates a perfect negative linear relationship. A value of 0 indicates no linear relationship between the two variables. The Pearson correlation coefficient can be used to determine the strength and direction of the relationship between two variables. A value of r close to +1 indicates a strong positive correlation, while a value close to -1 indicates a strong negative correlation. A value of r close to 0 indicates no correlation.

- Pearson's Correlation Coefficient (PCC or R) : Best possible score is 1.0, bigger value is better. Range = [-1, 1]
- The Pearson correlation coefficient is commonly used in various fields, including social sciences, economics, and engineering, to study the relationship

between two variables. + It is important to note that the Pearson correlation coefficient only measures linear relationships between variables, and may not capture other types of relationships, such as nonlinear or non-monotonic relationships.

Example to use R metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.pearson_correlation_coefficient())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
```

(continues on next page)

(continued from previous page)

```
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.R(multi_output="raw_values"))
```

3.19 AR - Absolute Pearson's Correlation Index

$$\text{R}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1}(|y_i - mean(y)| * |\hat{y}_i - mean(\hat{y})|)}{\sqrt{\sum_{i=0}^{N-1}(y_i - mean(y))^2} * \sqrt{\sum_{i=0}^{N-1}(\hat{y}_i - mean(\hat{y}))^2}}$$

Latex equation code:

```
\text{AR}(y, \hat{y}) = \frac{ \sum_{i=0}^{N - 1} (|y_i - mean(y)|*|\hat{y_i} - mean(\hat
↪{y})|) }{ \sqrt{ \sum_{i=0}^{N - 1} (y_i - mean(y))^2} *\sqrt{\sum_{i=0}^{N - 1} (\hat
↪{y_i} - mean(\hat{y}))^2} }
```

- Absolute Pearson's Correlation Coefficient (APCC or AR): Best possible score is 1.0, bigger value is better. Range = [0, 1]
- I developed this method, do not have enough time to analysis this metric.

Example to use AR metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.absolute_pearson_correlation_coefficient())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.AR(multi_output="raw_values"))
```

3.20 R2 - Coefficient of Determination

$$R2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ and $\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} \epsilon_i^2$

Latex equation code:

```
R2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar
↪{y})^2}
```

- Coefficient of Determination (COD/R2) [13]: Best possible score is 1.0, bigger value is better. Range = (-inf, 1]
- Link to equation
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is Pearson's Correlation Coefficient.
- We should denote it as COD or R2 only.
- It represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of

fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance. + As such variance is dataset dependent, R2 may not be meaningfully comparable across different datasets. + Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R2 score of 0.0.

Example to use R2 metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.coefficient_of_determination())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.R2(multi_output="raw_values"))
```

3.21 AR2 - Adjusted R2

$$AR2(y, \hat{y}) =$$

where $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ and $\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} \epsilon_i^2$

Latex equation code:

```
\text{AR2}(y, \hat{y}) = \textrm{Adjusted R}^2 = 1 - \frac{(1 - R^2)(n - 1)}{n - k - 1}
```

- Adjusted Coefficient of Determination (ACOD/AR2): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]
- Link to equation
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is PCC.

- We should denote it as COD or R2 only.

Here, n is the sample size, k is the number of predictors, R^2 is the coefficient of determination, and the Adjusted R2 is calculated as a modification of the R2 that takes into account the number of predictors in the model. The Adjusted R2 provides a more accurate measure of the goodness-of-fit of a model with multiple predictors.

Example to use AR2 metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.adjusted_coefficient_of_determination())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.AR2(multi_output="raw_values"))
```

3.22 CI - Confidence Index

$$\text{CI}(y, \hat{y}) = \text{R}(y, \hat{y}) * \text{WI}(y, \hat{y})$$

Latex equation code:

```
\text{CI}(y, \hat{y}) = \text{R}(y, \hat{y}) * \text{WI}(y, \hat{y})
```

Confidence Index [10] or Performance Index (CI/PI) is score that measures the performance of each estimation method, with a higher value indicating better performance. The range of the CI/PI is (-inf, 1], meaning it can take any value less than or equal to 1, but not including negative infinity.

- Best possible score is 1.0, bigger value is better. Range = (-inf, 1], meaning of values:

```
> 0.85          Excellent Model
0.76-0.85       Very good
0.66-0.75       Good
0.61-0.65       Satisfactory
0.51-0.60       Poor
0.41-0.50       Bad
< 0.40          Very bad
```

Example to use CI metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
```

(continues on next page)

(continued from previous page)

```
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.confidence_index())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.CI(multi_output="raw_values"))
```

3.23 R2s - (Pearson's Correlation Index)**2

$$\text{R2s}(y, \hat{y}) = \left[\frac{\sum_{i=0}^{N-1}((y_i - mean(y)) * (\hat{y}_i - mean(\hat{y})))}{\sqrt{\sum_{i=0}^{N-1}(y_i - mean(y))^2} * \sqrt{\sum_{i=0}^{N-1}(\hat{y}_i - mean(\hat{y}))^2}} \right]^2$$

Latex equation code:

```
\text{R2s}(y, \hat{y}) = \Bigg[ \frac{ \sum_{i=0}^{N - 1} ((y_i - mean(y))*(\hat{y}_i - mean(\hat{y}))) }{ \sqrt{ \sum_{i=0}^{N - 1} (y_i - mean(y))^2}*\sqrt{\sum_{i=0}^{N - 1} (\hat{y}_i - mean(\hat{y}))^2} } \Bigg]^2
```

- (Pearson's Correlation Index)^2 = R^2 = R2s (R square): Best possible score is 1.0, bigger value is better. Range = [0, 1]
- This actually a useless metric that I implemented here just to demonstrate the misunderstanding between R2s and R2 (Coefficient of Determination).
- Most of online tutorials (article, wikipedia,...) or even scikit-learn library are denoted the wrong R2s and R2.
- R^2 = R2s = R squared makes people think it as (Pearson's Correlation Index)^2
- However, R2 = Coefficient of Determination, link

Example to use R2s metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.pearson_correlation_coefficient_square())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.R2s(multi_output="raw_values"))
```

3.24 DRV - Deviation of Runoff Volume

$$\text{DRV}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1} y_i}{\sum_{i=0}^{N-1} \hat{y}_i}$$

Latex equation code:

```
\text{DRV}(y, \hat{y}) = \frac{ \sum_{i=0}^{N - 1} y_i }{ \sum_{i=0}^{N - 1} \hat{y_i} }
```

- Best possible score is 1.0, smaller value is better. Range = [1, +inf)
- Link to equation

Example to use DRV metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.deviation_of_runoff_volume())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.DRV(multi_output="raw_values"))
```

3.25 KGE - Kling-Gupta Efficiency

$$\text{KGE}(y, \hat{y}) = 1 - \sqrt{(r(y, \hat{y}) - 1)^2 + (\beta(y, \hat{y}) - 1)^2 + (\gamma(y, \hat{y}) - 1)^2}$$

where: r = correlation coefficient, CV = coefficient of variation, μ = mean, σ = standard deviation.

$$\beta = \text{bias ratio} = \frac{\mu_{\hat{y}}}{\mu_y}$$

$$\gamma = \text{variability ratio} = \frac{CV_{\hat{y}}}{CV_y} = \frac{\sigma_{\hat{y}}/\mu_{\hat{y}}}{\sigma_y/\mu_y}$$

The Kling-Gupta Efficiency (KGE) [12] is a statistical measure used to evaluate the performance of hydrological models. It was proposed to overcome the limitations of other measures such as Nash-Sutcliffe Efficiency and R-squared that focus only on reproducing the mean and variance of observed data.

The KGE combines three statistical metrics: correlation coefficient, variability ratio and bias ratio, into a single measure of model performance. The KGE ranges between -infinity and 1, where a value of 1 indicates perfect agreement between the model predictions and the observed data.

The KGE measures not only the accuracy of the model predictions but also its ability to reproduce the variability and timing of the observed data. It has been widely used in hydrology and water resources engineering to evaluate the performance of hydrological models in simulating streamflow, groundwater recharge, and water quality parameters.

- Best possible score is 1, bigger value is better. Range = (-inf, 1]
- Link to equation

Example to use KGE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.kling_gupta_efficiency())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.KGE(multi_output="raw_values"))
```

3.26 GINI - GINI Coefficient

The Gini coefficient [14] is a statistical measure used to measure income or wealth inequality within a population. It is named after the Italian statistician Corrado Gini who developed the concept in 1912. The Gini coefficient ranges from 0 to 1, where 0 represents perfect equality (every individual in the population has the same income or wealth) and 1 represents perfect inequality (one individual has all the income or wealth and everyone else has none).

$$G = \frac{A}{A + B}$$

where G is the Gini coefficient, A is the area between the Lorenz curve and the line of perfect equality, and B is the area under the line of perfect equality.

The Gini coefficient is calculated by plotting the cumulative share of income or wealth (on the x-axis) against the cumulative share of the population (on the y-axis) and measuring the area between this curve and the line of perfect equality (which is a straight diagonal line from the origin to the upper right corner of the plot).

The Gini coefficient is widely used in social sciences and economics to measure income or wealth inequality within and between countries. It is also used to analyze the distribution of other variables, such as educational attainment, health outcomes, and access to resources.

- Best possible score is 1, bigger value is better. Range = [0, 1]
- This version is based on: https://github.com/benhamner/Metrics/blob/master/MATLAB/metrics/gini.m

Example to use Gini metric, there are two GINI versions:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])
```

(continues on next page)

(continued from previous page)

```
evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.gini())
print(evaluator.gini_wiki())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.GINI(multi_output="raw_values"))
print(evaluator.GINI_WIKI(multi_output="raw_values"))
```

3.27 PCD - Prediction of Change in Direction

$$\text{PCD}(y, \hat{y}) = \frac{1}{n-1} \sum_{i=2}^{n} I\left((f_i - f_{i-1})(y_i - y_{i-1}) > 0\right)$$

Latex equation code:

```
\text{PCD}(y, \hat{y}) = \frac{1}{n-1} \sum_{i=2}^{n} I\left((f_{i}-f_{i-1}) (y_{i}-y_{i-1}) > 0\right)
```

- where f_i is the predicted value at time i, y_i is the actual value at time i, n is the total number of predictions, and $I(cdot)$ is the

indicator function which equals 1 if the argument is true and 0 otherwise. + Best possible score is 1.0, bigger value is better . Range = [0, 1] + The Prediction of Change in Direction (PCD) metric is used to evaluate the performance of regression models on detecting changes in the direction of a target variable.

Example to use PCD metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.prediction_of_change_in_direction())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.PCD(multi_output="raw_values"))
```

3.28 CE - Cross Entropy

$$\text{CE}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Latex equation code:

```
\text{CE}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n} \left[y_i\log(\hat{y}_i) + (1-y_i)\
↪log(1-\hat{y}_i)\right]
```

- Range = (-inf, 0]. Can't give comment about this one
- Greater value of Entropy, the greater the uncertainty for probability distribution and smaller the value the less the uncertainty
- Link to equation

Example to use CE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.cross_entropy())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.CE(multi_output="raw_values"))
```

3.29 KLD - Kullback-Leibler Divergence

The Kullback-Leibler Divergence (KLD), [15] also known as relative entropy, is a statistical measure of how different two probability distributions are from each other. It was introduced by Solomon Kullback and Richard Leibler in 1951. The KLD is calculated as the sum of the logarithmic differences between the probabilities of each possible outcome, weighted by the probability of the outcome in the reference distribution. The KLD is always non-negative, and it is equal to zero if and only if the two distributions are identical. The equation for KLD between two probability distributions P and Q is given by:

$$D_{KL}(P||Q) = \sum_{i} P(i) \log \frac{P(i)}{Q(i)}$$

where P(i) and Q(i) are the probabilities of the i-th possible outcome in the two distributions, respectively.

The KLD measures the information lost when approximating one probability distribution by another. It is widely used in information theory, machine learning, and data science applications, such as clustering, classification, and data compression. The KLD has also found applications in other fields, such as physics, economics, and biology, to measure the distance between two probability distributions.

- Best possible score is 0.0 . Range = (-inf, +inf)
- Link to equation

Example to use KLD metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.kullback_leibler_divergence())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6], [1, 2], [2.1, 2.2], [3.4, 5.5]])
y_pred = array([[0, 2], [-1, 2], [8, -5], [1.1, 1.9], [2.0, 2.3], [3.0, 4.2]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.KLD(multi_output="raw_values"))
```

3.30 JSD - Jensen-Shannon Divergence

The Jensen-Shannon Divergence (JSD) [16] is a statistical measure of the similarity between two probability distributions. It is named after Danish mathematician Johan Jensen and American mathematician Richard Shannon, who introduced the concept in 1991.

The JSD is a symmetric and smoothed version of the Kullback-Leibler Divergence (KLD), which is a measure of how much one probability distribution differs from another. Unlike the KLD, the JSD is always a finite value and satisfies the triangle inequality, making it a proper metric.

The JSD is calculated as follows:

- Compute the average probability distribution by taking the arithmetic mean of the two distributions:

$$M = 0.5 * (P + Q)$$

where P and Q are the two probability distributions being compared.

- Calculate the KLD between each distribution and the average distribution:

$$D(P||M) = \sum_i P(i) \log \frac{P(i)}{M(i)}$$

$$D(Q||M) = \sum_i Q(i) \log \frac{Q(i)}{M(i)}$$

- Compute the JSD as the arithmetic mean of the two KLD values:

$$JSD(P||Q) = \frac{1}{2} (D(P||M) + D(Q||M))$$

The JSD measures the distance between two probability distributions, with values ranging from 0 (when the distributions are identical) to 1 (when the distributions are completely different). It is commonly used in machine learning and information retrieval applications, such as text classification and clustering.

- Best possible score is 0.0 (identical), smaller value is better . Range = [0, +inf)
- Link to equation

Example to use JSD metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.jensen_shannon_divergence())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6], [1, 2], [2.1, 2.2], [3.4, 5.5]])
y_pred = array([[0, 2], [-1, 2], [8, -5], [1.1, 1.9], [2.0, 2.3], [3.0, 4.2]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.JSD(multi_output="raw_values"))
```

3.31 VAF - Variance Accounted For

$$\text{VAF}(y, f_i) = 100\% \times \frac{\sum_{i=1}^{n}(y_i - \bar{y})(f_i - \bar{f})}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Latex equation code:

```
\text{VAF}(y, f_i) = 100\% \times \frac{\sum_{i=1}^{n}(y_i - \bar{y})(f_i - \bar{f})}{\
↪sum_{i=1}^{n}(y_i - \bar{y})^2}
```

- Variance Accounted For (VAF) is a metric used to evaluate the performance of a regression model. It measures the proportion of the total variance in the

actual values that is accounted for by the variance in the predicted values. + Variance Accounted For between 2 signals (VAF): Best possible score is 100% (identical signal), bigger value is better. Range = (-inf, 100%] + Link to equation

Example to use VAF metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.variance_accounted_for())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])
```

(continues on next page)

(continued from previous page)

```
evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.VAF(multi_output="raw_values"))
```

3.32 RAE - Relative Absolute Error

$$\text{RAE}(y, \hat{y}) = \frac{\left[\sum_{i=1}^{n}(\hat{y}_i - y_i)^2\right]^{1/2}}{\left[\sum_{i=1}^{n}(y_i)^2\right]^{1/2}}$$

Latex equation code:

```
\text{RAE}(y, \hat{y}) = \frac{\Big[\sum_{i=1}^{n}(\hat{y}_i - y_i)^2\Big]^{1/2}}{\Big[\
↪sum_{i=1}^{n}(y_i)^2\Big]^{1/2}}
```

- Relative Absolute Error (RAE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- Link to equation
- Link to equation
- The Relative Absolute Error (RAE) is a metric used to evaluate the accuracy of a regression model by measuring the ratio of the mean absolute error to the

mean absolute deviation of the actual values.

Example to use RAE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.relative_absolute_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.RAE(multi_output="raw_values"))
```

3.33 A10 - A10 index

$$\text{A10}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \begin{cases} 1, & \text{if } \frac{|\hat{y}_i - y_i|}{y_i} \leq 0.1 \\ 0, & \text{otherwise} \end{cases}$$

Latex equation code:

```
\text{A10}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \left\{\begin{array}{ll} 1, & \textrm
↪{if } \frac{|\hat{y}_i - y_i|}{y_i} \leq 0.1\\ 0, & \textrm{otherwise} \end{array}\
↪right.
```

- Best possible score is 1.0, bigger value is better. Range = [0, 1]
- a10-index is engineering index for evaluating artificial intelligence models by showing the number of samples that fit the

prediction values with a deviation of ±10% compared to experimental values + Link to equation

In other words, the A10 metric measures the proportion of cases where the absolute difference between the predicted and actual values is less than or equal to 10% of the actual value. A higher A10 score indicates better predictive accuracy, as the model is able to make more accurate predictions that are closer to the actual values.

Example to use A10 metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.a10_index())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.A10(multi_output="raw_values"))
```

3.34 A20 - A20 index

$$\text{A20}(y, \hat{y}) = \text{A10}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \begin{cases} 1, & \text{if } \frac{|\hat{y}_i - y_i|}{y_i} \leq 0.2 \\ 0, & \text{otherwise} \end{cases}$$

Latex equation code:

```
\text{A20}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \left\{\begin{array}{ll} 1, & \textrm
↪{if } \frac{|\hat{y}_i - y_i|}{y_i} \leq 0.2\\ 0, & \textrm{otherwise} \end{array}\
↪right.
```

- a20-index (A20) [12] evaluated metric by showing the number of samples that fit the prediction values with a deviation of ±20%

compared to experimental values. + In other words, the A20 metric measures the proportion of cases where the absolute difference between the predicted and actual values is less than or equal to 20% of the actual value. A higher A20 score indicates better predictive accuracy, as the model is able to make more accurate predictions that are closer to the actual values. + Best possible score is 1.0, bigger value is better. Range = [0, 1]

Example to use A20 metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.a20_index())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.A20(multi_output="raw_values"))
```

3.35 A30 - A30 index

$$\text{A30}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \left\{\begin{array}{ll} 1, & \text{if } \frac{|\hat{y}_i - y_i|}{y_i} \leq 0.3 \\ 0, & \text{otherwise} \end{array}\right.$$

Latex equation code:

```
\text{A30}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \left\{\begin{array}{ll} 1, & \textrm
↪{if } \frac{|\hat{y}_i - y_i|}{y_i} \leq 0.3\\ 0, & \textrm{otherwise} \end{array}\
↪right.
```

- Best possible score is 1.0, bigger value is better. Range = [0, 1]
- a30-index (A30) [12] evaluated metric by showing the number of samples that fit the prediction values with a deviation of ±30%

compared to experimental values. + In other words, the A30 metric measures the proportion of cases where the absolute difference between the predicted and actual values is less than or equal to 30% of the actual value. A higher A30 score indicates better predictive accuracy, as the model is able to make more accurate predictions that are closer to the actual values.

Example to use A30 metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])
```

(continues on next page)

(continued from previous page)

```
evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.a30_index())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.A30(multi_output="raw_values"))
```

3.36 NRMSE - Normalized Root Mean Square Error

The NRMSE [17] is calculated as the RMSE divided by the range of the observed values, expressed as a percentage. The range of the observed values is the difference between the maximum and minimum values of the observed data.

- Normalized Root Mean Square Error (NRMSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- Link to equation
- Example to use NMRSE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.normalized_root_mean_square_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6], [1, 2], [2.1, 2.2], [3.4, 5.5]])
y_pred = array([[0, 2], [-1, 2], [8, -5], [1.1, 1.9], [2.0, 2.3], [3.0, 4.2]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.NRMSE(multi_output="raw_values"))
```

3.37 RSE - Residual Standard Error

$$\text{RSE}(y, f_i) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - f_i)^2}{n-p-1}}$$

Latex equation code:

```
\text{RSE}(y, f_i) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - f_i)^2}{n-p-1}}
```

- Residual Standard Error (RSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- Link to equation

- Link to equation
- The Residual Standard Error (RSE) is a metric used to evaluate the goodness of fit of a regression model. It measures the average distance between the

observed values and the predicted values.

Example to use RSE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.residual_standard_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.RSE(multi_output="raw_values"))
```

3.38 COV - Covariance

Covariance of population .. math:

```
\text{COV}(y, \hat{y}) = \frac{\sum_{i=1}^{N} (y_i - mean(Y)) (\hat{y}_i - mean(\hat{Y}
↪))}{N}
```

Covariance of sample .. math:

```
\text{COV}(y, \hat{y}) = \frac{\sum_{i=1}^{N} (y_i - mean(Y)) (\hat{y}_i - mean(\hat{Y}
↪))}{N - 1}
```

- There is no best value, bigger value is better. Range = [-inf, +inf)
- Positive covariance: Indicates that two variables tend to move in the same direction.
- Negative covariance: Reveals that two variables tend to move in inverse directions.
- COV is a measure of the relationship between two random variables evaluates how much – to what extent – the variables change together, does not assess the

dependency between variables. + Link to equation

Example to use COV metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
```

(continues on next page)

(continued from previous page)

```
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.covariance())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.COV(multi_output="raw_values"))
```

3.39 COR - Correlation

$$\text{COR}(y, \hat{y}) = \frac{COV(y, \hat{y})}{std(y) * std(\hat{y})}$$

Correlation [18] measures the strength of the relationship between the variables and is a scaled measure of covariance. The correlation coefficient ranges from -1 to +1, where a value of 1 indicates a perfect positive correlation, a value of -1 indicates a perfect negative correlation, and a value of 0 indicates no correlation.

To calculate the correlation coefficient, you divide the covariance of the variables by the product of their standard deviations. This normalization allows for comparison between different pairs of variables. The correlation coefficient is dimensionless and does not have any specific units of measurement.

- Best possible value = 1, bigger value is better. Range = [-1, +1)
- Measures the strength of the relationship between variables, is the scaled measure of covariance. It is dimensionless.
- the correlation coefficient is always a pure value and not measured in any units.
- Link to equation

Example to use COR metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.correlation())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.COR(multi_output="raw_values"))
```

3.40 EC - Efficiency Coefficient

$$\text{EC}(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - mean(Y))^2}$$

Latex equation code:

```
\text{EC}(y, \hat{y}) = 1 - \frac{ \sum_{i=1}^n (y_i - \hat{y_i})^2 }{ \sum_{i=1}^n (y_i - mean(Y))^2 }
```

Efficiency Coefficient (EC) [18] is a metric used to evaluate the accuracy of a regression model in predicting continuous values.

- Best possible value = 1, bigger value is better. Range = [-inf, +1]
- The EC ranges from negative infinity to 1, where a value of 1 indicates a perfect match between the model predictions and the observed data, and a value

of 0 indicates that the model predictions are no better than the benchmark prediction. + A negative value indicates that the model predictions are worse than the benchmark prediction. + Link to equation

Example to use EC metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.efficiency_coefficient())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.EC(multi_output="raw_values"))
```

3.41 OI - Overall Index

$$\text{OI}(y, \hat{y}) = \frac{1}{2}\left[1 - \frac{RMSE}{y_{max} - y_{min}} + EC\right]$$

Latex equation code:

```
\text{OI}(y, \hat{y}) = \frac{1}{2} \biggr[ 1 - \frac{RMSE}{y_{max} - y_{min}} + EC \
↪biggr]
```

The Overall Index (OI) [19] is a composite measure used to evaluate the accuracy of a forecasting model. It combines the Root Mean Squared Error (RMSE) with a measure of the relative error and a correction term. + Best possible value = 1, bigger value is better. Range = [-1, +1)

Example to use COR metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.overall_index())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.OI(multi_output="raw_values"))
```

3.42 CRM - Coefficient of Residual Mass

$$\text{CRM}(y, \hat{y}) = \frac{\sum \hat{Y} - \sum Y}{\sum Y}$$

The CRM [19] is a measure of the accuracy of the model in predicting the values of the dependent variable. A lower value of CRM indicates that the model is better at predicting the values of the dependent variable, while a higher value indicates poorer performance. The coefficient of residual mass is typically used in environmental engineering and hydrology to measure the accuracy of models used to predict water quality and quantity, sediment transport, and erosion. + Best possible value = 0, smaller value is better. Range = (-inf, +inf) + Link to equation

Example to use CRM metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.coefficient_of_residual_mass())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.CRM(multi_output="raw_values"))
```

3.43 RE - Relative Error

$$\text{RE}(y, \hat{y}) = \frac{|y_i - \hat{y}_i|}{|y_i|}$$

Latex equation code:

```
\text{RE}(y, \hat{y}) = \frac{|y_i - \hat{y}_i|}{|y_i|}
```

- Relative Error (RE): Best possible score is 0.0, smaller value is better. Range = (-inf, +inf)
- Note: Computes the relative error between two numbers, or for element between a pair of list, tuple or numpy arrays.
- The Relative Error (RE) is a metric used to evaluate the accuracy of a regression model by measuring the ratio of the absolute error to the actual value.

Example to use RE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.single_relative_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.RE())
```

3.44 AE - Absolute Error

$$\text{AE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Latex equation code:

```
\text{AE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} | \hat{y}_i - y_i |
```

- Best possible score is 0.0, smaller value is better. Range = (-inf, +inf)
- Computes the absolute error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Example to use AE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric
```

(continues on next page)

(continued from previous page)

```
## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.single_absolute_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.AE())
```

3.45 SE - Squared Error

$$\text{SE}(y, f_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_i)^2$$

Latex equation code:

```
\text{SE}(y, f_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_i)^2
```

- Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- Note: Computes the squared error between two numbers, or for element between a pair of list, tuple or numpy arrays.
- The Squared Error (SE) is a metric used to evaluate the accuracy of a regression model by measuring the average of the squared differences between the

predicted and actual values.

Example to use SE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.single_squared_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.SE())
```

3.46 SLE - Squared Log Error

$$\text{SLE}(y, f_i) = \frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(f_i + 1))^2$$

Latex equation code:

```
\text{SLE}(y, f_i) = \frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(f_i + 1))^2
```

- Squared Log Error (SLE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)
- Note: Computes the squared log error between two numbers, or for element between a pair of list, tuple or numpy arrays.
- The Squared Log Error (SLE) is a metric used to evaluate the accuracy of regression models that predict logarithmic values. It measures the average of the

squared differences between the logarithm of the predicted values and the logarithm of the actual values.

Example to use SLE metric:

```
from numpy import array
from permetrics.regression import RegressionMetric

## For 1-D array
y_true = array([3, -0.5, 2, 7])
y_pred = array([2.5, 0.0, 2, 8])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.single_squared_log_error())

## For > 1-D array
y_true = array([[0.5, 1], [-1, 1], [7, -6]])
y_pred = array([[0, 2], [-1, 2], [8, -5]])

evaluator = RegressionMetric(y_true, y_pred)
print(evaluator.SLE())
```

CHAPTER

FOUR

CLASSIFICATION METRICS

STT	Metric	Metric Fullname	Characteristics
1	PS	Precision Score	Bigger is better (Best = 1), Range = [0, 1]
2	NPV	Negative Predictive Value	Bigger is better (Best = 1), Range = [0, 1]
3	RS	Recall Score	Bigger is better (Best = 1), Range = [0, 1]
4	AS	Accuracy Score	Bigger is better (Best = 1), Range = [0, 1]
5	F1S	F1 Score	Bigger is better (Best = 1), Range = [0, 1]
6	F2S	F2 Score	Bigger is better (Best = 1), Range = [0, 1]
7	FBS	F-Beta Score	Bigger is better (Best = 1), Range = [0, 1]
8	SS	Specificity Score	Bigger is better (Best = 1), Range = [0, 1]
9	MCC	Matthews Correlation Coefficient	Bigger is better (Best = 1), Range = [-1, +1]
10	HS	Hamming Score	Bigger is better (Best = 1), Range = [0, 1]
11	CKS	Cohen's kappa score	Bigger is better (Best = +1), Range = [-1, +1]
12	JSI	Jaccard Similarity Coefficient	Bigger is better (Best = +1), Range = [0, +1]
13	GMS	Geometric Mean Score	Bigger is better (Best = +1), Range = [0, +1]
14	ROC-AUC	ROC-AUC	Bigger is better (Best = +1), Range = [0, +1]
15	LS	Lift Score	Bigger is better (No best value), Range = [0, +inf)
16	GINI	GINI Index	Smaller is better (Best = 0), Range = [0, +1]
17	CEL	Cross Entropy Loss	Smaller is better (Best = 0), Range=[0, +inf)
18	HL	Hinge Loss	Smaller is better (Best = 0), Range=[0, +inf)
19	KLDL	Kullback Leibler Divergence Loss	Smaller is better (Best = 0), Range=[0, +inf)
20	BSL	Brier Score Loss	Smaller is better (Best = 0), Range=[0, +1]

In extending a binary metric to multiclass or multilabel problems, the data is treated as a collection of binary problems, one for each class. There are then a number of ways to average binary metric calculations across the set of classes, each of which may be useful in some scenario. Where available, you should select among these using the average parameter.

- "micro" gives each sample-class pair an equal contribution to the overall metric (except as a result of sample-weight). Rather than summing the metric per

class, this sums the dividends and divisors that make up the per-class metrics to calculate an overall quotient. Micro-averaging may be preferred in multilabel settings, including multiclass classification where a majority class is to be ignored. Calculate metrics globally by considering each element of the label indicator matrix as a label.

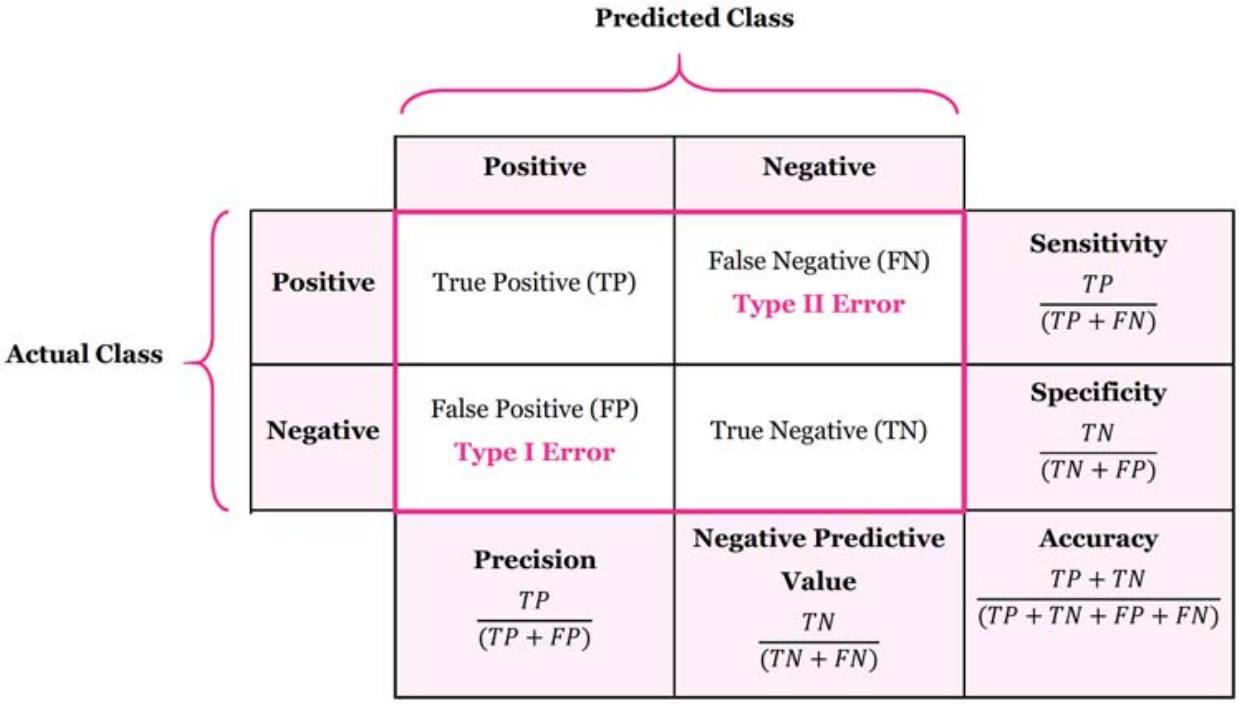
- "macro" simply calculates the mean of the binary metrics, giving equal weight to each class. In problems where infrequent classes are nonetheless important,

macro-averaging may be a means of highlighting their performance. On the other hand, the assumption that all classes are equally important is often untrue, such that macro-averaging will over-emphasize the typically low performance on an infrequent class.

- "weighted" accounts for class imbalance by computing the average of binary metrics in which each class's score is weighted by its presence in the true data sample.

- None: will return an array with the score for each class.

4.1 Accuracy Score (AS)



In the multi-class and multi-label case, the "average of the AS score" refers to the average of the Accuracy Score (AS) for each class. The weighting of the average depends on the average parameter, which determines the type of averaging to be performed. There are several options for the average parameter, including "macro", "micro", and "weighted". Here's a description of each averaging method: * Macro averaging: Calculates the AS for each class independently and then takes the average. Each class is given equal weight, regardless of its size or distribution. This averaging method treats all classes equally. * Micro averaging: Calculates the AS by considering the total number of true positives, false negatives, and false positives across all classes. This method gives more weight to classes with larger numbers of instances. * Weighted averaging: Similar to macro averaging, this method calculates the AS for each class independently and takes the average. However, each class is given a weight proportional to its number of instances. This means that classes with more instances contribute more to the overall average.

The choice of averaging method depends on the specific requirements and characteristics of the problem at hand. It's important to consider the class distribution, class imbalance, and the desired focus of evaluation when selecting the appropriate averaging method.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
- https://www.debadityachakravorty.com/ai-ml/cmatrix/
- https://neptune.ai/blog/evaluation-metrics-binary-classification

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric
```

(continues on next page)

(continued from previous page)

```
## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.accuracy_score(average=None))
print(cm.accuracy_score(average="micro"))
print(cm.AS(average="macro"))
print(cm.AS(average="weighted"))
```

4.2 Cohen Kappa Score (CKS)

The Cohen's Kappa score is a statistic that measures the level of agreement between two annotators on a categorical classification problem. It is a measure of inter-annotator reliability that is often used in medical diagnoses, quality control, and content analysis.

The Kappa score is calculated as the ratio of the observed agreement between two annotators to the agreement that would be expected by chance. The observed agreement is the number of instances that are classified the same way by both annotators, and the expected agreement is the number of instances that are classified the same way by chance, given the individual annotator's classifications.

The formula for the Cohen's Kappa score is as follows .. math:

```
k = (observed agreement - expected agreement) / (1 - expected agreement)
```

where observed agreement is the proportion of items that are classified the same way by both annotators, and expected

agreement is theproportion of items that are classified the same way by chance.

$\kappa = \frac{p_o - p_e}{1 - p_e}$

$where$

tp$represents the number of true positive annotations (agreements betwee$

tn$represents the number of true negative annotations (agreements betwe$

fp$represents the number of false positive annotations (disagreements be$

fn$represents the number of false negative annotations (disagreements be$

- Best possible score is 1.0, higher value is better. Range = [-1, 1]
- The value of k ranges from -1 to 1, with values closer to 1 indicating high levels of agreement, and values closer to -1 indicating low levels of agreement.

A value of 0 indicates that the agreement between the annotators is no better than chance. A value of 1 indicates perfect agreement.

The Cohen's Kappa score can be used for both binary and multi-class classification problems. For multi-class classification problems, the observed agreement and expected agreement are calculated based on a confusion matrix, which is a table that shows the number of instances that are classified into each possible pair of true and predicted classes. The confusion matrix is used to calculate the observed agreement and expected agreement between the annotators, and the resulting values are used in the formula for the Cohen's Kappa score.

It's important to note that the Cohen's Kappa score can be negative if the agreement between y_true and y_pred is lower than what would be expected by chance. A value of 1.0 indicates perfect agreement, and a value of 0.0 indicates no agreement beyond chance.

Also, this implementation of the Cohen's Kappa score is flexible and can handle binary as well as multi-class classification problems. The calculation of the confusion matrix and the subsequent calculation of the expected and observed agreements is based on the assumption that the ground truth labels and predicted labels are integer values that represent the different classes. If the labels are represented as strings or some other data type, additional pre-processing would be required to convert them to integer values that can be used in the confusion matrix.

Example:

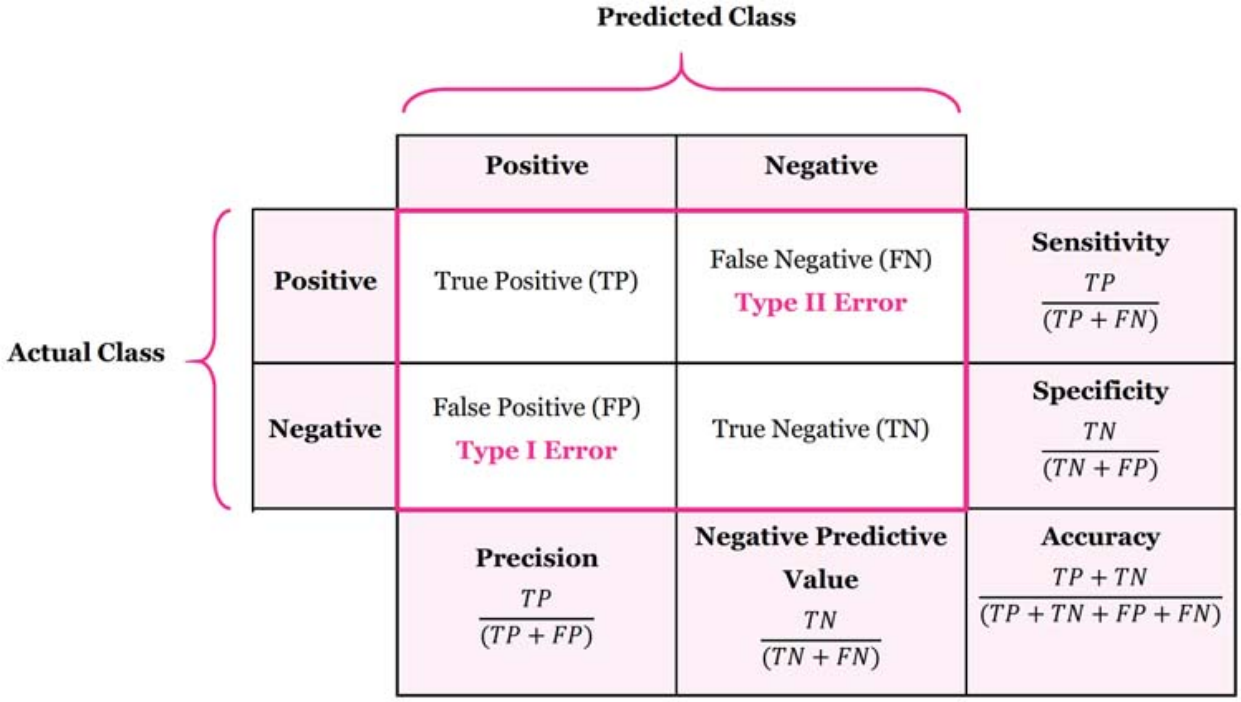
```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 2, 0, 2, 1, 1, 2, 2, 0]
y_pred = [0, 0, 2, 0, 2, 2, 1, 1, 2, 0]
```

(continues on next page)

(continued from previous page)

```
# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

cm = ClassificationMetric(y_true, y_pred)

print(cm.cohen_kappa_score(average=None))
print(cm.CKS(average="micro"))
print(cm.CKS(average="macro"))
print(cm.CKS(average="weighted"))
```

4.3 F1 Score (F1S)



Compute the F1 score, also known as balanced F-score or F-measure.

The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is

$$F1 = 2 * (precision * recall)/(precision + recall)$$

In the multi-class and multi-label case, this is the average of the F1 score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
- https://www.debadityachakravorty.com/ai-ml/cmatrix/

- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.f1_score(average=None))
print(cm.F1S(average="micro"))
print(cm.F1S(average="macro"))
print(cm.F1S(average="weighted"))
```

4.4 F2 Score (F2S)

It's a metric that combines precision and recall, putting 2x emphasis on recall

$$F2 = 5 * (precision * recall)/(4 * precision + recall)$$

In the multi-class and multi-label case, this is the average of the F2 score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
- https://www.debadityachakravorty.com/ai-ml/cmatrix/
- https://neptune.ai/blog/evaluation-metrics-binary-classification

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.f2_score(average=None))
```

(continues on next page)

(continued from previous page)

```
print(cm.F2S(average="micro"))
print(cm.F2S(average="macro"))
print(cm.F2S(average="weighted"))
```

4.5 F-Beta Score (FBS)

The F-beta score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0.

The beta parameter determines the weight of recall in the combined score. beta < 1 lends more weight to precision, while beta > 1 favors recall (beta -> 0 considers only precision, beta -> +inf only recall).

$$F - beta = (1 + beta * *2) * (precision * recall)/(beta * *2 * precision + recall)$$

In the multi-class and multi-label case, this is the average of the FBS score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html#sklearn.metrics.fbeta_score

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.fbeta_score(average=None))
print(cm.fbeta_score(average="micro"))
print(cm.FBS(average="macro"))
print(cm.FBS(average="weighted"))
```

4.6 GINI Index

The Gini index is a measure of impurity or inequality often used in decision tree algorithms for evaluating the quality of a split. It quantifies the extent to which a split divides the target variable (class labels) unevenly across the resulting branches.

The Gini index ranges from 0 to 1, where 0 indicates a perfect split, meaning all the samples in each branch belong to the same class, and 1 indicates an impure split, where the samples are evenly distributed across all classes. To calculate the Gini index, you can use the following formula:

Gini index = 1 - (sum of squared probabilities of each class)

For a binary classification problem, with two classes (0 and 1), the Gini index can be calculated as:

```
Gini index = 1 - (p0^2 + p1^2)
where p0 is the probability of class 0 and p1 is the probability of class 1 in the split.
```

For a multiclass classification problem, the Gini index is calculated as:

```
Gini index = 1 - (p0^2 + p1^2 + ... + pn^2)
where p0, p1, ..., pn are the probabilities of each class in the split.
```

The Gini index is used to evaluate the quality of a split and guide the decision tree algorithm to select the split that results in the lowest Gini index.

It's important to note that the Gini index is not typically used as an evaluation metric for the overall performance of a classification model. Instead, it is primarily used within the context of decision trees for determining the optimal splits during the tree-building process. The Gini index is also used as a metric to evaluate the performance of a binary classification model. It is a measure of how well the model separates the positive and negative classes.

- Smaller is better (Best = 0), Range = [0, +1]

Example:

```python
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)
print(cm.gini_index(average=None))
print(cm.GINI()
print(cm.GINI()
```

4.7 G-Mean Score (GMS)

G-mean is a performance metric in the field of machine learning and specifically in binary classification problems. It is a balanced version of the geometric mean, which is calculated as the square root of the product of true positive rate (TPR) and true negative rate (TNR) also known as sensitivity and specificity, respectively.

The G-mean is a commonly used metric to evaluate the performance of a classifier in imbalanced datasets where one class has a much higher number of samples than the other. It provides a balanced view of the model's performance as it penalizes low values of TPR and TNR in a single score. The G-mean score provides a balanced evaluation of a classifier's performance by considering both the positive and negative classes.

The formula for the G-mean score is given by

$$G - mean = sqrt(TPR * TNR)$$
$$Gmean = \sqrt{TPR * TNR}$$

where TPR (True Positive Rate) is defined as

$$\text{TPR} = \text{TP}_{\overline{TP+FN}}$$

and TNR (True Negative Rate) is defined as

$$\text{TNR} = \text{TN}_{\overline{TN+FP}}$$

with TP (True Positives) as the number of instances that are correctly classified as positive, TN (True Negatives) as the number of instances that are correctly classified as negative, FP (False Positives) as the number of instances that are wrongly classified as positive, and FN (False Negatives) as the number of instances that are wrongly classified as negative.

- Best possible score is 1.0, higher value is better. Range = [0, 1]

For a binary classification problem with two classes, the G-mean score provides a single value that represents the overall accuracy of the classifier. A G-mean score of 1.0 indicates perfect accuracy, while a score of less than 1.0 indicates that one of the classes is being misclassified more frequently than the other.

In a multi-class classification problem, the G-mean score can be calculated for each class and then averaged over all classes to provide a single value that represents the overall accuracy of the classifier. The average can be weighted or unweighted, depending on the desired interpretation of the results.

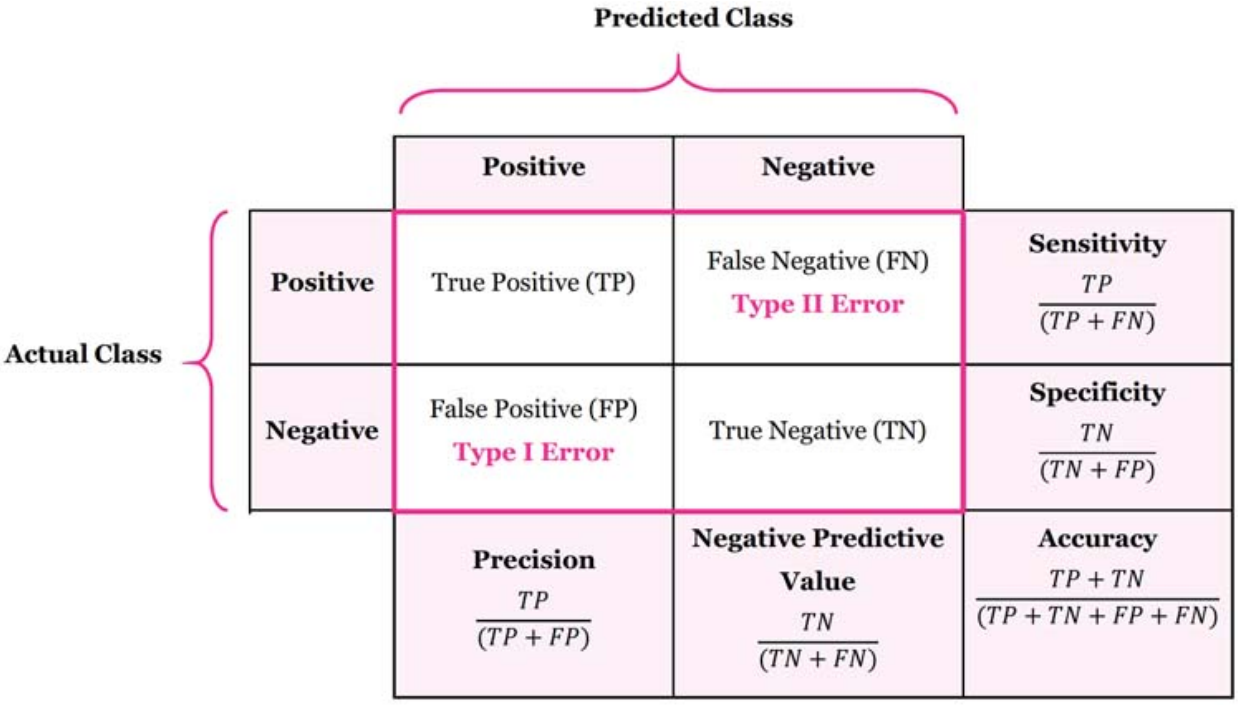
For example, consider a multi-class classification problem with three classes: class A, class B, and class C. The G-mean score for each class can be calculated using the formula above, and then averaged over all classes to provide an overall G-mean score for the classifier.

The G-mean score provides a way to balance the accuracy of a classifier between positive and negative classes, and is particularly useful in cases where the class distribution is imbalanced, or when one class is more important than the other.

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.g_mean_score(average=None))
print(cm.GMS(average="micro"))
print(cm.GMS(average="macro"))
print(cm.GMS(average="weighted"))
```

4.8 Precision Score (PS)



The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

In the multi-class and multi-label case, this is the average of the PS score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
- https://www.debadityachakravorty.com/ai-ml/cmatrix/
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score

Example:

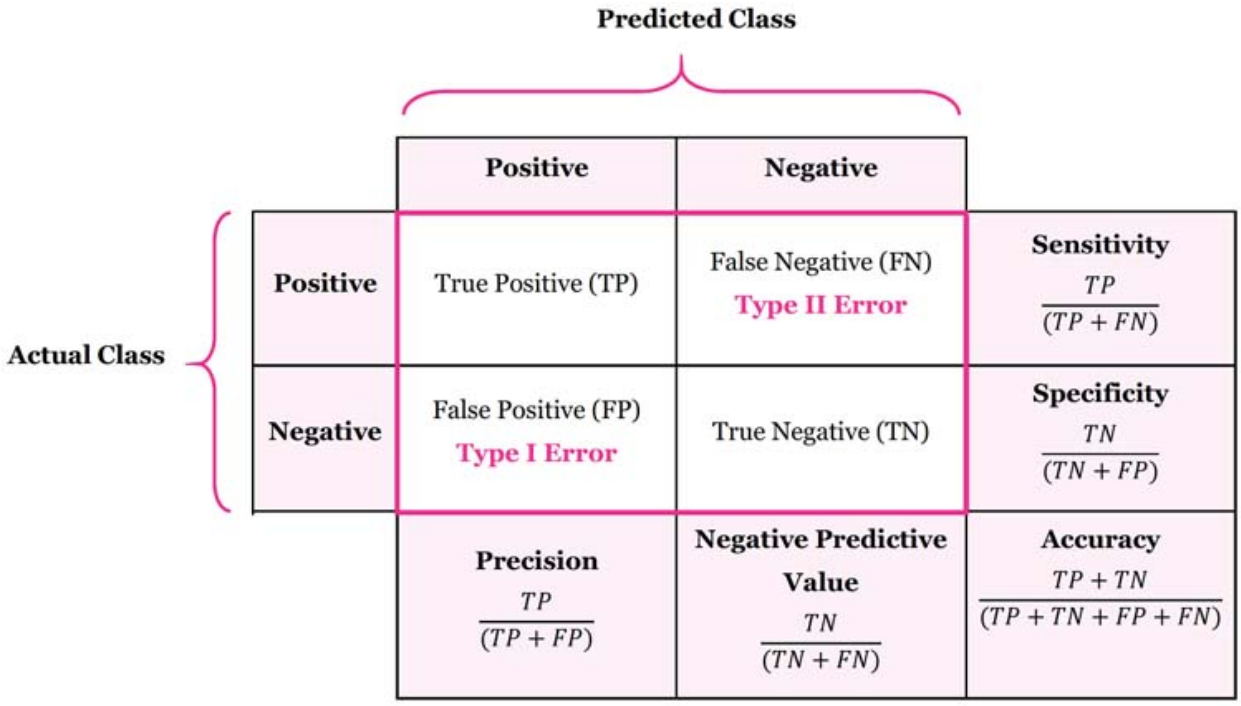
```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)
```

(continues on next page)

(continued from previous page)

```
print(cm.PS(average=None))
print(cm.PS(average="micro"))
print(cm.PS(average="macro"))
print(cm.PS(average="weighted"))
```

4.9 Negative Predictive Value (NPV)



The negative predictive value is defined as the number of true negatives (people who test negative who don't have a condition) divided by the total number of people who test negative.

The negative predictive value is the ratio tn / (tn + fn) where tn is the number of true negatives and fn the number of false negatives.

In the multi-class and multi-label case, this is the average of the NPV score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://www.debadityachakravorty.com/ai-ml/cmatrix/
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2

Example:

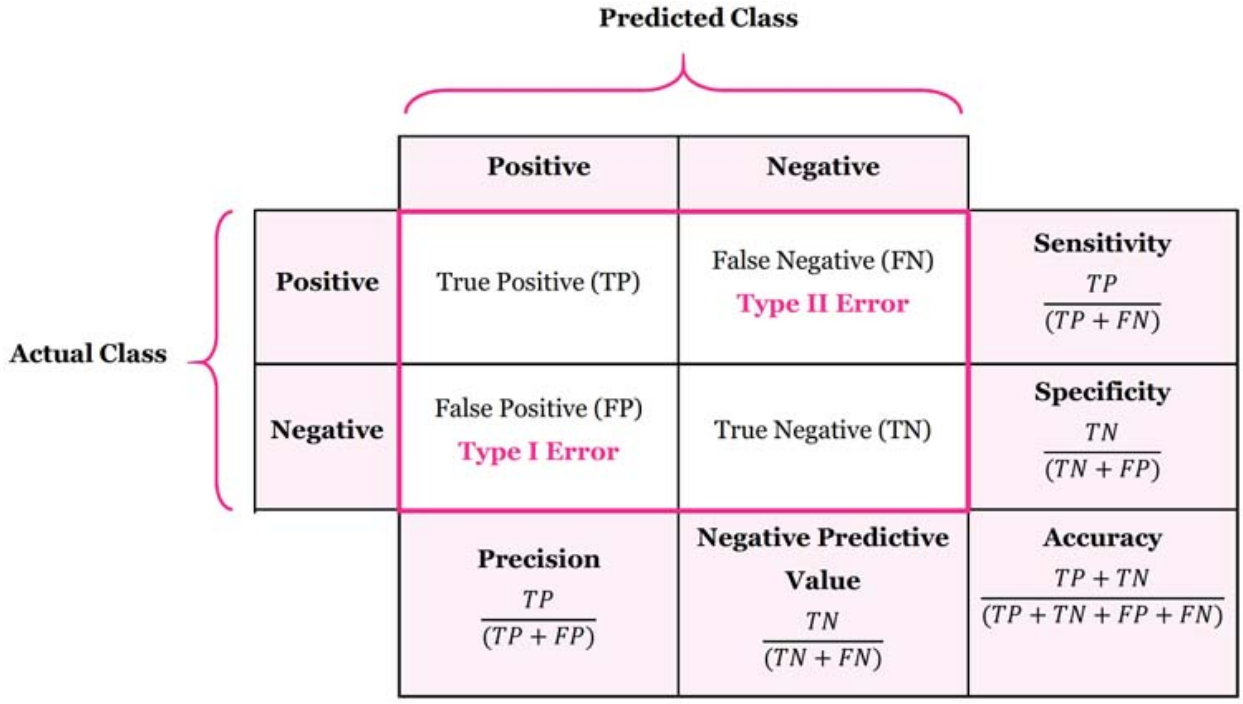
```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
```

(continues on next page)

(continued from previous page)

```
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.npv(average=None))
print(cm.NPV(average="micro"))
print(cm.NPV(average="macro"))
print(cm.NPV(average="weighted"))
```

4.10 Recall Score (RS)



The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

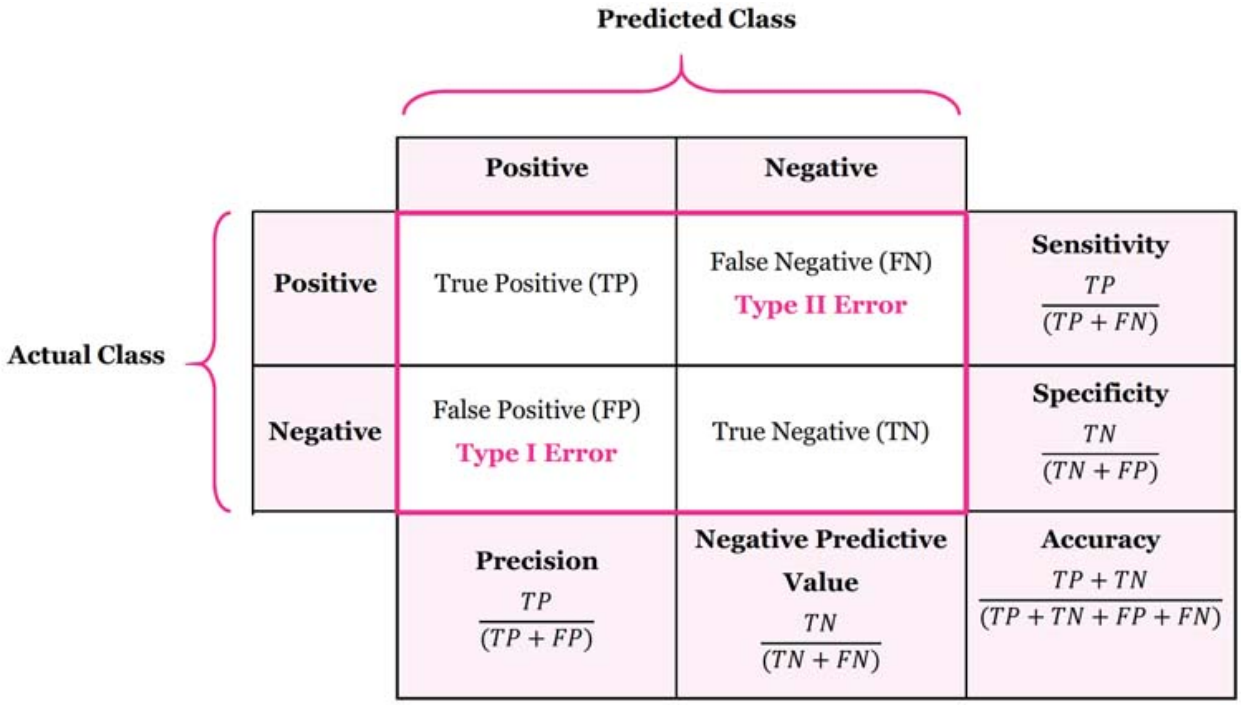
In the multi-class and multi-label case, this is the average of the RS score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
- https://www.debadityachakravorty.com/ai-ml/cmatrix/
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.recall_score(average=None))
print(cm.RS(average="micro"))
print(cm.RS(average="macro"))
print(cm.RS(average="weighted"))
```

4.11 Specificity Score (SS)



The specificity score is the ratio tn / (tn + fp) where tn is the number of false positives and fp the number of false positives. It measures how many observations out of all negative observations have we classified as negative. In fraud detection example, it tells us how many transactions, out of all non-fraudulent transactions, we marked as clean.

In the multi-class and multi-label case, this is the average of the SS score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://neptune.ai/blog/evaluation-metrics-binary-classification

- https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
- https://www.debadityachakravorty.com/ai-ml/cmatrix/

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.specificity_score(average=None))
print(cm.ss(average="micro"))
print(cm.SS(average="macro"))
print(cm.SS(average="weighted"))
```

4.12 Matthews Correlation Coefficient (MCC)

In the multi-class and multi-label case, this is the average of the MCC score of each class with weighting depending on the average parameter.

- Best possible score is 1.0, higher value is better. Range = [-1, +1]
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html#sklearn.metrics.matthews_corrcoef

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.mcc(average=None))
print(cm.MCC(average="micro"))
print(cm.MCC(average="macro"))
print(cm.MCC(average="weighted"))
```

4.13 Hamming Score (HS)

The Hamming score is 1 - the fraction of labels that are incorrectly predicted.

In the multi-class and multi-label case, this is the average of the HL score of each class with weighting depending on the average parameter.

- Higher is better (Best = 1), Range = [0, 1]
- A little bit difference than hamming_score in scikit-learn library.
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.hamming_score.html#sklearn.metrics.hamming_score

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]

cm = ClassificationMetric(y_true, y_pred)

print(cm.hamming_score(average=None))
print(cm.HS(average="micro"))
print(cm.HS(average="macro"))
print(cm.HS(average="weighted"))
```

4.14 Lift Score (LS)

In the multi-class and multi-label case, this is the average of the LS score of each class with weighting depending on the average parameter.

- Higher is better (No best value), Range = [0, +inf)
- http://rasbt.github.io/mlxtend/user_guide/evaluate/lift_score/
- https://neptune.ai/blog/evaluation-metrics-binary-classification

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]
```

(continues on next page)

(continued from previous page)

```
cm = ClassificationMetric(y_true, y_pred)

print(cm.lift_score(average=None))
print(cm.LS(average="micro"))
print(cm.LS(average="macro"))
print(cm.LS(average="weighted"))
```

4.15 Jaccard Similarity Index (JSI)

- Best possible score is 1.0, higher value is better. Range = [0, 1]

The Jaccard similarity index, also known as the Jaccard similarity coefficient or Jaccard index, is a commonly used evaluation metric in binary and multiclass classification problems. It measures the similarity between the predicted labels y_pred and the true labels y_true, and is defined as the ratio of the number of true positive (TP) predictions to the number of true positive and false positive (FP) predictions.

In a binary classification problem, a prediction is considered true positive if both the true label and the predicted label are positive, and false positive if the true label is negative and the predicted label is positive. False negative (FN) predictions are those where the true label is positive and the predicted label is negative. True negative (TN) predictions are those where both the true label and the predicted label are negative.

To calculate the Jaccard similarity index, the formula is

$$J = TP/(TP + FP)$$

Where TP (True Positives) is the number of instances where the true label and the predicted label are both positive, and FP (False Positives) is the number of instances where the true label is negative and the predicted label is positive.

In a multiclass classification problem, the Jaccard similarity index can be calculated for each class individually, and then averaged over all classes to obtain the overall Jaccard similarity index. The weighted average of the Jaccard similarity indices can also be calculated, with the weights given by the number of instances in each class.

The Jaccard similarity index ranges from 0 to 1, with a value of 1 indicating perfect agreement between the predicted labels and the true labels, and a value of 0 indicating complete disagreement. The Jaccard similarity index is a useful evaluation metric in situations where the class distribution is imbalanced, as it takes into account only true positive and true negative predictions and not false positive or false negative predictions.

It's important to note that the Jaccard similarity index is sensitive to the number of instances in each class, so it's recommended to use this metric in combination with other evaluation metrics, such as precision, recall, and F1-score, to get a complete picture of the performance of a classification model.

Example:

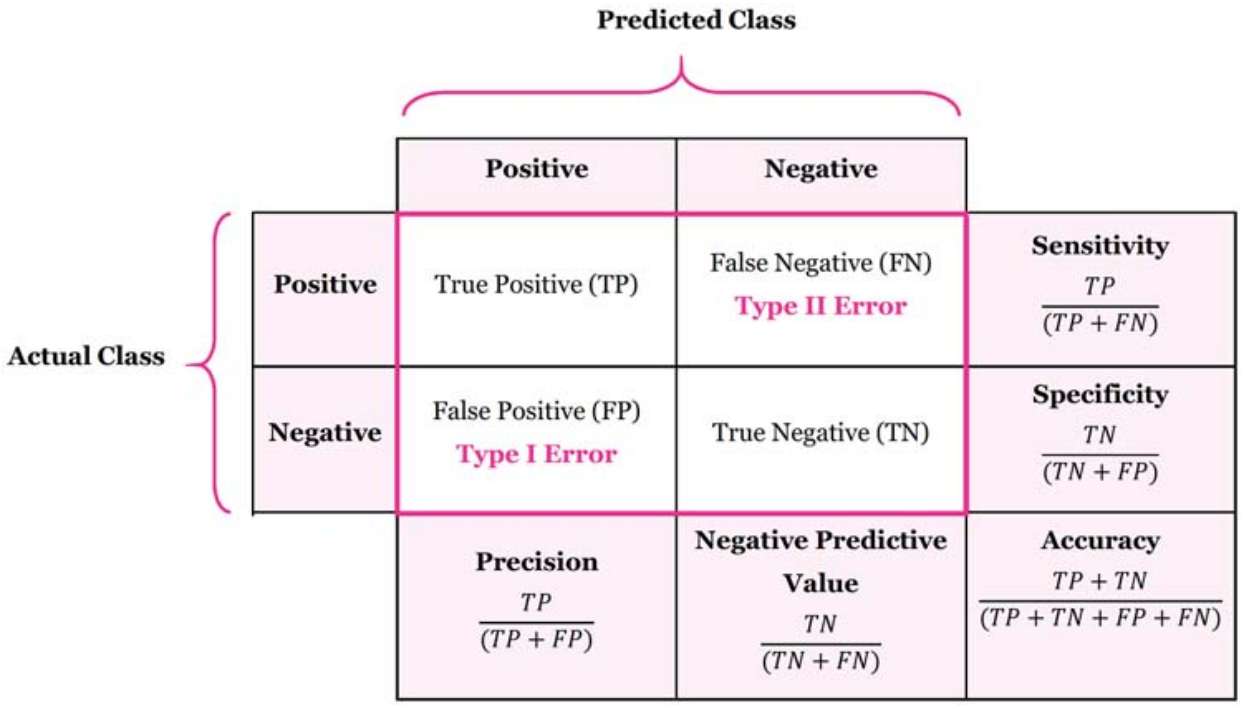
```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]

# y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird", "bird"]
# y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "ant"]
```

(continues on next page)

(continued from previous page)

```
cm = ClassificationMetric(y_true, y_pred)

print(cm.jaccard_similarity_index(average=None))
print(cm.jaccard_similarity_coefficient(average="micro"))
print(cm.JSI(average="macro"))
print(cm.JSC(average="weighted"))
```

4.16 ROC-AUC



ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) is a metric used to evaluate the performance of a binary classification model. It is a measure of how well the model is able to distinguish between positive and negative classes.

A ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The TPR is the ratio of the number of true positives to the total number of positives, while the FPR is the ratio of the number of false positives to the total number of negatives.

The AUC is the area under the ROC curve. It ranges between 0 and 1, where a value of 0.5 represents a model that performs no better than random guessing, and a value of 1 represents a model that makes perfect predictions. A higher AUC value indicates that the model is better at distinguishing between the positive and negative classes.

Interpretation of the ROC curve and AUC value depends on the specific problem and domain. In general, a model with an AUC value of 0.7 to 0.8 is considered acceptable, while a value greater than 0.8 is considered good. However, the interpretation may vary depending on the specific use case and the cost of false positives and false negatives.

In the multi-class and multi-label case, this is the average of the AS score of each class with weighting depending on the average parameter.

In a multiclass classification problem, ROC-AUC can still be used as a metric, but it requires some modifications to account for the multiple classes.

One approach is to use the one-vs-all (OvA) strategy, where we train a binary classifier for each class, treating it as the positive class and all other classes as the negative class. For each class, we calculate the ROC curve and AUC value, and then average the AUC values across all classes to obtain a single metric.

Another approach is to use the one-vs-one (OvO) strategy, where we train a binary classifier for each pair of classes, treating one class as the positive class and the other as the negative class. For each pair of classes, we calculate the ROC curve and AUC value, and then average the AUC values across all pairs to obtain a single metric.

In either case, it is important to ensure that the classes are balanced, meaning that the number of examples in each class is roughly equal, or to use appropriate sampling techniques to handle class imbalance.

It is worth noting that ROC-AUC may not always be the best metric for multiclass problems, especially when the classes are highly imbalanced or the cost of false positives and false negatives varies across classes. In such cases, other metrics such as precision, recall, F1-score, or weighted average of these metrics may be more appropriate.

- Best possible score is 1.0, higher value is better. Range = [0, 1]
- https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/
- There is no "micro" average mode in ROC-AUC metric

Example:

```
from numpy import array
from permetrics.classification import ClassificationMetric

## For integer labels or categorical labels
y_true = [0, 1, 0, 0, 1, 0]
y_score = [0, 1, 0, 0, 0, 1]

y_true = np.array([0, 1, 2, 1, 2, 0, 0, 1])
y_score = np.array([[0.8, 0.1, 0.1],
            [0.2, 0.5, 0.3],
            [0.1, 0.3, 0.6],
            [0.3, 0.7, 0.0],
            [0.4, 0.3, 0.3],
            [0.6, 0.2, 0.2],
            [0.9, 0.1, 0.0],
            [0.1, 0.8, 0.1]])

cm = ClassificationMetric(y_true, y_pred)

print(cm.roc_auc_score(y_true, y_score, average=None))
print(cm.ROC(y_true, y_score))
print(cm.AUC(y_true, y_score, average="macro"))
print(cm.RAS(y_true, y_score, average="weighted"))
```

CHAPTER

FIVE

CLUSTERING METRICS

STT	Metric	Metric Fullname	Characteristics
1	BHI	Ball Hall Index	Smaller is better (Best = 0), Range=[0, +inf)
2	XBI	Xie Beni Index	Smaller is better (Best = 0), Range=[0, +inf)
3	DBI	Davies Bouldin Index	Smaller is better (Best = 0), Range=[0, +inf)
4	BRI	Banfeld Raftery Index	Smaller is better (No best value), Range=(-inf, inf)
5	KDI	Ksq Detw Index	Smaller is better (No best value), Range=(-inf, +inf)
6	DRI	Det Ratio Index	Bigger is better (No best value), Range=[0, +inf)
7	DI	Dunn Index	Bigger is better (No best value), Range=[0, +inf)
8	CHI	Calinski Harabasz Index	Bigger is better (No best value), Range=[0, inf)
9	LDRI	Log Det Ratio Index	Bigger is better (No best value), Range=(-inf, +inf)
10	LSRI	Log SS Ratio Index	Bigger is better (No best value), Range=(-inf, +inf)
11	SI	Silhouette Index	Bigger is better (Best = 1), Range = [-1, +1]
12	SSEI	Sum of Squared Error Index	Smaller is better (Best = 0), Range = [0, +inf)
13	MSEI	Mean Squared Error Index	Smaller is better (Best = 0), Range = [0, +inf)
14	DHI	Duda-Hart Index	Smaller is better (Best = 0), Range = [0, +inf)
15	BI	Beale Index	Smaller is better (Best = 0), Range = [0, +inf)
16	RSI	R-squared Index	Bigger is better (Best=1), Range = (-inf, 1]
17	DBCVI	Density-based Clustering Validation Index	Bigger is better (Best=0), Range = [0, 1]
18	HI	Hartigan Index	Bigger is better (best=0), Range = [0, +inf)
19	MIS	Mutual Info Score	Bigger is better (No best value), Range = [0, +inf)
20	NMIS	Normalized Mutual Info Score	Bigger is better (Best = 1), Range = [0, 1]
21	RaS	Rand Score	Bigger is better (Best = 1), Range = [0, 1]
22	ARS	Adjusted Rand Score	Bigger is better (Best = 1), Range = [-1, 1]
23	FMS	Fowlkes Mallows Score	Bigger is better (Best = 1), Range = [0, 1]
24	HS	Homogeneity Score	Bigger is better (Best = 1), Range = [0, 1]
25	CS	Completeness Score	Bigger is better (Best = 1), Range = [0, 1]
26	VMS	V-Measure Score	Bigger is better (Best = 1), Range = [0, 1]
27	PrS	Precision Score	Bigger is better (Best = 1), Range = [0, 1]
28	ReS	Recall Score	Bigger is better (Best = 1), Range = [0, 1]
29	FmS	F-Measure Score	Bigger is better (Best = 1), Range = [0, 1]
30	CDS	Czekanowski Dice Score	Bigger is better (Best = 1), Range = [0, 1]
31	HGS	Hubert Gamma Score	Bigger is better (Best = 1), Range=[-1, +1]
32	JS	Jaccard Score	Bigger is better (Best = 1), Range = [0, 1]
33	KS	Kulczynski Score	Bigger is better (Best = 1), Range = [0, 1]
34	MNS	Mc Nemar Score	Bigger is better (No best value), Range=(-inf, +inf)
35	PhS	Phi Score	Bigger is better (No best value), Range = (-inf, +inf)
36	RTS	Rogers Tanimoto Score	Bigger is better (Best = 1), Range = [0, 1]
37	RRS	Russel Rao Score	Bigger is better (Best = 1), Range = [0, 1]

continues on next page

Table 1 – continued from previous page

STT	Metric	Metric Fullname	Characteristics
38	SS1S	Sokal Sneath1 Score	Bigger is better (Best = 1), Range = [0, 1]
39	SS2S	Sokal Sneath2 Score	Bigger is better (Best = 1), Range = [0, 1]
40	PuS	Purity Score	Bigger is better (Best = 1), Range = [0, 1]
41	ES	Entropy Score	Smaller is better (Best = 0), Range = [0, +inf)
42	TS	Tau Score	Bigger is better (No best value), Range = (-inf, +inf)
43	GAS	Gamma Score	Bigger is better (Best = 1), Range = [-1, 1]
44	GPS	Gplus Score	Smaller is better (Best = 0), Range = [0, 1]

Most of the clustering metrics is implemented based on the paper [20]

There are several types of clustering metrics that are commonly used to evaluate the quality of clustering results.

- Internal evaluation metrics: These are metrics that evaluate the clustering results based solely on the data and the clustering algorithm used, without any external information. Examples of internal evaluation metrics include Silhouette Coefficient, Calinski-Harabasz Index, and Davies-Bouldin Index.
- External evaluation metrics: These are metrics that evaluate the clustering results by comparing them to some external reference, such as expert labels or a gold standard. Examples of external evaluation metrics include Adjusted Rand score, Normalized Mutual Information score, and Fowlkes-Mallows score.

It's important to choose the appropriate clustering metrics based on the specific problem and data at hand.

In this library, metrics that belong to the internal evaluation category will have a metric name suffix of "index" On the other hand, metrics that belong to the external evaluation category will have a metric name suffix of "score"

5.1 Duda Hart Index (DHI)

The Duda index, also known as the D-index or Duda-Hart index, is a clustering evaluation metric that measures the compactness and separation of clusters. It was proposed by Richard O. Duda and Peter E. Hart in their book "Pattern Classification and Scene Analysis."

The Duda index is defined as the ratio between the average pairwise distance within clusters and the average pairwise distance between clusters. A lower value of the Duda index indicates better clustering, indicating that the clusters are more compact and well-separated. Here's the formula to calculate the Duda index:

```
Duda Index = (Average pairwise intra-cluster distance) / (Average pairwise inter-cluster␣
↪distance)
```

To calculate the Duda index, you need the following steps:

```
Compute the average pairwise distance within each cluster (intra-cluster distance).
Compute the average pairwise distance between different clusters (inter-cluster␣
↪distance).
Divide the average intra-cluster distance by the average inter-cluster distance to␣
↪obtain the Duda index.
```

The Duda index is a useful metric for evaluating clustering results, particularly when the compactness and separation of clusters are important. However, it's worth noting that the Duda index assumes Euclidean distance and may not work well with all types of data or distance metrics.

When implementing the Duda index, you'll need to calculate the pairwise distances between data points within and between clusters. You can use distance functions like Euclidean distance or other suitable distance metrics based on your specific problem and data characteristics.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.duda_hart_index())
print(cm.DHI())
```

5.2 Sum of Squared Error Index (SSEI)

Sum of Squared Error (SSE) is a commonly used metric to evaluate the quality of clustering in unsupervised learning problems. SSE measures the sum of squared distances between each data point and its corresponding centroid or cluster center. It quantifies the compactness of the clusters.

Here's how you can calculate the SSE in a clustering problem:

```
1) Assign each data point to its nearest centroid or cluster center based on some␣
↪distance metric (e.g., Euclidean distance).
2) For each data point, calculate the squared Euclidean distance between the data point␣
↪and its assigned centroid.
3) Sum up the squared distances for all data points to obtain the SSE.
```

Higher SSE values indicate higher dispersion or greater variance within the clusters, while lower SSE values indicate more compact and well-separated clusters. Therefore, minimizing the SSE is often a goal in clustering algorithms.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.sum_squared_error_index())
print(cm.SSEI())
```

5.3 Beale Index (BI)

The Beale Index is a clustering validation metric that measures the quality of a clustering solution by computing the ratio of the within-cluster sum of squares to the between-cluster sum of squares. It is also known as the "variance ratio criterion" or the "F-ratio".

The within-cluster sum of squares is a measure of the variability of the data points within each cluster, while the between-cluster sum of squares is a measure of the variability between the clusters. The idea is that a good clustering solution should have low within-cluster variation and high between-cluster variation, which results in a high Beale Index value.

The Beale Index can be calculated using the following formula:

```
Beale Index = (sum of squared errors within clusters / degrees of freedom within␣
↪clusters) / (sum of squared errors between clusters / degrees of freedom between␣
↪clusters)
```

where the degrees of freedom are the number of data points minus the number of clusters, and the sum of squared errors is the sum of the squared distances between each data point and the centroid of its assigned cluster.

The Beale Index ranges from 0 to infinity, with higher values indicating better clustering solutions. However, the Beale Index has a tendency to favor solutions with more clusters, so it's important to consider other metrics in conjunction with it.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.beale_index())
print(cm.BI())
```

5.4 R-Squared Index (RSI)

The R-squared index is another clustering validation metric that is used to measure the quality of a clustering solution. It is based on the idea of comparing the variance of the data before and after clustering. The R-squared index measures the proportion of the total variance in the data that is explained by the clustering solution.

The R-squared index is calculated using the following formula:

```
R-squared = (total variance - variance within clusters) / total variance
```

where total variance is the variance of the entire dataset, and variance within clusters is the sum of the variances of each cluster. The R-squared index ranges from -inf to 1, with higher values indicating better clustering solutions. A negative value indicates that the clustering solution is worse than random, while a value of 0 indicates that the clustering solution explains no variance beyond chance. A value of 1 indicates that the clustering solution perfectly explains all the variance in the data.

Note that the R-squared index has some limitations, as it can be biased towards solutions with more clusters. It is also sensitive to the scale and dimensionality of the data, and may not be appropriate for all clustering problems. Therefore, it's important to consider multiple validation metrics when evaluating clustering solutions.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.r_squared_index())
print(cm.RSI())
```

5.5 Density-Based Clustering Validation Index (DBCVI)

The Density-Based Clustering Validation (DBCV) metric is another clustering validation metric that is used to evaluate the quality of a clustering solution, particularly for density-based clustering algorithms such as DBSCAN.

The DBCV metric measures the average ratio of the distances between the data points and their cluster centroids, to the distances between the data points and the nearest data points in other clusters. The idea is that a good clustering solution should have compact and well-separated clusters, so the ratio of these distances should be high.

The DBCV metric is calculated using the following formula:

```
DBCV = (1 / n) * sum_{i=1}^n (sum_{j=1}^n (d(i,j) / max{d(i,k), k!=j}))
```

where n is the number of data points, d(i,j) is the Euclidean distance between data points i and j, and max{d(i,k), k!=j} is the maximum distance between data point i and any other data point in a different cluster.

The DBCV metric ranges from 0 to 1, with lower values indicating better clustering solutions. A value of 0 indicates a perfect clustering solution, where all data points belong to their own cluster and the distances between clusters are maximized.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.density_based_clustering_validation_index())
print(cm.DBCVI())
```

5.6 Calinski-Harabasz Index

The Calinski-Harabasz Index is a clustering evaluation metric used to measure the quality of clusters obtained from clustering algorithms. It aims to quantify the separation between clusters and the compactness within clusters.

The CH Index is calculated as the ratio of the between-cluster variance to the within-cluster variance, multiplied by a scaling factor. The formula for the Calinski-Harabasz Index for a dataset with n data points, k clusters, and where B represents the between-cluster variance and W represents the within-cluster variance, is as follows:

$$CH = \frac{B}{W} \times \frac{n-k}{k-1}$$

Where:

- B is the sum of squared distances between cluster centroids and the overall dataset mean, weighted by the number of data points in each cluster.
- W is the sum of squared distances between data points and their respective cluster centroids.
- n is the total number of data points.
- k is the number of clusters.

A higher Calinski-Harabasz Index indicates better clustering, as it signifies that the clusters are well-separated and compact. The term $\frac{n-k}{k-1}$ acts as a correction factor to avoid overfitting and make the index more reliable.

In practice, you can use the Calinski-Harabasz Index along with other clustering evaluation metrics to assess the performance of clustering algorithms and select the best number of clusters for your dataset.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.calinski_harabasz_index())
print(cm.CHI())
```

5.7 Ball Hall Index

The Ball Hall Index is a clustering validity index that measures the compactness and separation of clusters in a clustering result. It provides a quantitative measure of how well-separated and tight the clusters are.

The formula for calculating the Ball Hall Index is as follows:

```
BHI = Xichma(1 / (2 * n_i) * Xichma(d(x, c_i)) / n
```

Where:

n is the total number of data points n_i is the number of data points in cluster i d(x, c_i) is the Euclidean distance between a data point x and the centroid c_i of cluster i

The Ball Hall Index computes the average distance between each data point and its cluster centroid and then averages this across all clusters. The index is inversely proportional to the compactness and separation of the clusters. A smaller BHI value indicates better-defined and well-separated clusters.

A lower BHI value indicates better clustering, as it signifies that the data points are closer to their own cluster centroid than to the centroids of other clusters, indicating a clear separation between clusters.

The Ball Hall Index is often used as an internal evaluation metric for clustering algorithms to compare different clustering results or to determine the optimal number of clusters. However, it should be noted that it is not without limitations and should be used in conjunction with other evaluation metrics and domain knowledge for a comprehensive assessment of clustering results.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.ball_hall_index())
print(cm.BHI())
```

5.8 Dunn Index (DI)

The Dunn Index, which is a measure used to evaluate the performance of clustering algorithms. The Dunn Index aims to quantify the compactness and separation between clusters in a clustering solution. It helps assess the quality of the clustering by considering both the distance between points within the same cluster (intra-cluster distance) and the distance between points in different clusters (inter-cluster distance).

Let us denote by d_{min} the minimal distance between points of different clusters and d_{max} the largest within-cluster distance.

The distance between clusters C_k and $C_{k'}$ is measured by the distance between their closest points:

$$d_{kk'} = \min_{\substack{i \in I_k \\ j \in I_{k'}}} ||M_i^{\{k\}} - M_j^{\{k'\}}||$$

and d_{min} is the smallest of these distances $d_{kk'}$:

$$d_{min} = \min_{k \neq k'} d_{kk'}$$

For each cluster C_k, let us denote by D_k the largest distance separating two distinct points in the cluster (sometimes called the diameter of the cluster):

$$D_k = \max_{\substack{i,j \in I_k \\ i \neq j}} ||M_i^{\{k\}} - M_j^{\{k\}}||.$$

Then d_{max} is the largest of these distances D_k:

$$d_{max} = \max_{1 \leq k \leq K} D_k$$

The Dunn index is defined as the quotient of d_{min} and d_{max}:

$$\boxed{\mathcal{C} = \frac{d_{min}}{d_{max}}}$$

A higher Dunn Index value indicates better clustering quality – it suggests that the clusters are well separated from each other while being compact internally. Conversely, a lower Dunn Index value may indicate that the clusters are too spread out or not well separated.

However, like any clustering evaluation metric, the Dunn Index has its limitations and should be used in conjunction with other metrics and domain knowledge. It's worth noting that the choice of clustering algorithm, distance metric, and dataset characteristics can influence the interpretation of the Dunn Index.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.dunn_index())
```

(continues on next page)

(continued from previous page)

```
print(cm.DI())
```

5.9 Hartigan Index (HI)

The Hartigan index, also known as the Hartigan's criterion, is a measure used for evaluating the quality of clustering solutions. It is specifically designed for assessing the goodness of fit of a clustering algorithm, particularly the k-means algorithm.

Hartigan's Index, which is a clustering performance metric used to evaluate the quality of a clustering solution. It's named after its creator, John A. Hartigan. The Hartigan's Index measures the ratio of the sum of the squares of the distances between the data points within each cluster to the sum of the squares of the distances between the data points and their respective cluster centroids. In other words, it quantifies how tightly the data points are clustered within their respective clusters compared to how spread out they are from the cluster centers.

Here's the formula for calculating Hartigan's Index for a set of clusters:

H = ∑(d(xi, cj))^2 / ∑(d(xi, ck))^2

Where:

H is the Hartigan's Index for the clustering solution.
xi represents a data point.
cj represents the centroid of the cluster to which xi belongs.
ck represents the centroid of the closest cluster other than the one to which xi belongs.
d(a, b) is the distance between data points a and b.
A lower value of Hartigan's Index indicates a better clustering solution because it suggests that the data points are tightly clustered within their respective clusters and are distant from other cluster centroids. In other words, smaller values of H indicate that the clusters are more compact and well-separated.

It's worth noting that while Hartigan's Index provides a useful measure of cluster quality, it's not the only metric available for evaluating clustering performance. Other metrics such as silhouette score, Davies-Bouldin index, and within-cluster sum of squares (WCSS) are also commonly used for assessing the quality of clustering solutions. The choice of metric often depends on the characteristics of the data and the goals of the clustering analysis.

The Hartigan index quantifies the compactness of clusters and the separation between clusters in a clustering solution.

It aims to find a balance between minimizing the within-cluster variance (compactness) and maximizing the between-cluster variance (separation).

While the Hartigan index is a useful measure, it is not as widely used as other clustering evaluation indices like the Silhouette coefficient or Dunn index. Nevertheless, it can provide insights into the quality of a clustering solution, particularly when comparing different clustering algorithms or determining the optimal number of clusters.

The goal of the Hartigan index is to minimize this ratio. Lower values of the Hartigan index indicate better clustering solutions with lower within-cluster variance and higher separation between clusters.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
data = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
y_pred = np.array([0, 0, 1, 1, 1])

cm = ClusteringMetric(X=data, y_pred=y_pred)

print(cm.hartigan_index())
print(cm.HI())
```

5.10 Entropy Score (ES)

Entropy is a metric used to evaluate the quality of clustering results, particularly when the ground truth labels of the data points are known. It measures the amount of uncertainty or disorder within the clusters produced by a clustering algorithm.

Here's how the Entropy score is calculated:

```
1) For each cluster, compute the class distribution by counting the occurrences of each␣
↪class label within the cluster.
2) Normalize the class distribution by dividing the count of each class label by the␣
↪total number of data points in the cluster.
3) Compute the entropy for each cluster using the normalized class distribution.
4) Weight the entropy of each cluster by its relative size (proportion of data points in␣
↪the whole dataset).
5) Sum up the weighted entropies of all clusters.
```

The resulting value is the Entropy score, which typically ranges from 0 to 1. A lower Entropy score indicates better clustering, as it implies more purity and less uncertainty within the clusters.

Entropy score considers both the composition of each cluster and the distribution of classes within the clusters. It provides a more comprehensive evaluation of clustering performance compared to simple metrics like Purity.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
y_true = np.array([0, 0, 1, 1, 1, 2, 2, 1])
y_pred = np.array([0, 0, 1, 1, 2, 2, 2, 2])
```

(continues on next page)

(continued from previous page)

```
cm = ClusteringMetric(y_true=y_true, y_pred=y_pred)

print(cm.entropy_score())
print(cm.ES())
```

5.11 Purity Score (PuS)

Purity is a metric used to evaluate the quality of clustering results, particularly in situations where the ground truth labels of the data points are known. It measures the extent to which the clusters produced by a clustering algorithm match the true class labels of the data. Here's how Purity is calculated:

```
1) For each cluster, find the majority class label among the data points in that cluster.
2) Sum up the sizes of the clusters that belong to the majority class label.
3) Divide the sum by the total number of data points.
```

The resulting value is the Purity score, which ranges from 0 to 1. A Purity score of 1 indicates a perfect clustering, where each cluster contains only data points from a single class.

Purity is a simple and intuitive metric but has some limitations. It does not consider the actual structure or distribution of the data within the clusters and is sensitive to the number of clusters and class imbalance. Therefore, it may not be suitable for evaluating clustering algorithms in all scenarios.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
y_true = np.array([0, 0, 1, 1, 1, 2, 2, 1])
y_pred = np.array([0, 0, 1, 1, 2, 2, 2, 2])

cm = ClusteringMetric(y_true=y_true, y_pred=y_pred)

print(cm.purity_score())
print(cm.PuS())
```

5.12 Tau Score (TS)

The Tau index, also known as the Tau coefficient, is a measure of agreement or similarity between two clustering solutions. It is commonly used to compare the similarity of two different clusterings or to evaluate the stability of a clustering algorithm.

The Tau index is based on the concept of concordance, which measures the extent to which pairs of objects are assigned to the same clusters in two different clustering solutions. The index ranges from -1 to 1, where 1 indicates perfect agreement, 0 indicates random agreement, and -1 indicates perfect disagreement or inversion of the clustering solutions.

The calculation of the Tau index involves constructing a contingency table that counts the number of pairs of objects that are concordant (i.e., assigned to the same cluster in both solutions) and discordant (i.e., assigned to different clusters in the two solutions).

The formula for calculating the Tau index is as follows:

```
Tau = (concordant_pairs - discordant_pairs) / (concordant_pairs + discordant_pairs)
```

A higher value of the Tau index indicates greater similarity or agreement between the two clusterings, while a lower value indicates less agreement. It's important to note that the interpretation of the Tau index depends on the specific clustering algorithm and the data being clustered.

The Tau index can be useful in various applications, such as evaluating the stability of clustering algorithms, comparing different clustering solutions, or assessing the robustness of a clustering method to perturbations in the data. However, like any clustering evaluation measure, it has its limitations and should be used in conjunction with other evaluation techniques to gain a comprehensive understanding of the clustering performance.

Example:

```
import numpy as np
from permetrics import ClusteringMetric

## For integer labels or categorical labels
y_true = np.array([0, 0, 1, 1, 1, 2, 2, 1])
y_pred = np.array([0, 0, 1, 1, 2, 2, 2, 2])

cm = ClusteringMetric(y_true=y_true, y_pred=y_pred)

print(cm.tau_score())
print(cm.TS())
```

CHAPTER

SIX

PERMETRICS LIBRARY

6.1 permetrics.utils package

6.1.1 permetrics.utils.classifier_util module

permetrics.utils.classifier_util.**calculate_class_weights**(*y_true*, *y_pred=None*, *y_score=None*)

permetrics.utils.classifier_util.**calculate_confusion_matrix**(*y_true=None*, *y_pred=None*, *labels=None*, *normalize=None*)

Generate a confusion matrix for multiple classification

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **normalize** (*'true', 'pred', 'all', None*) – Normalizes confusion matrix over the true (rows), predicted (columns) conditions or all the population.

Returns a 2-dimensional list of pairwise counts imap (dict): a map between label and index of confusion matrix imap_count (dict): a map between label and number of true label in y_true

Return type matrix (np.ndarray)

permetrics.utils.classifier_util.**calculate_roc_curve**(*y_true*, *y_score*)

permetrics.utils.classifier_util.**calculate_single_label_metric**(*matrix*, *imap*, *imap_count*, *beta=1.0*)

Generate a dictionary of supported metrics for each label

Parameters

- **matrix** (*np.ndarray*) – a 2-dimensional list of pairwise counts
- **imap** (*dict*) – a map between label and index of confusion matrix
- **imap_count** (*dict*) – a map between label and number of true label in y_true
- **beta** (*float*) – to calculate the f-beta score

Returns a dictionary of supported metrics

Return type dict_metrics (dict)

6.1.2 permetrics.utils.cluster_util module

permetrics.utils.cluster_util.**calculate_adjusted_rand_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_ball_hall_index**(*X=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_banfeld_raftery_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*)

permetrics.utils.cluster_util.**calculate_beale_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*)

permetrics.utils.cluster_util.**calculate_calinski_harabasz_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*)

Parameters

- **X** – The X matrix features
- **y_pred** – The predicted results
- **force_finite** – Make result as finite number
- **finite_value** – The value that used to replace the infinite value or NaN value.

Returns The Calinski Harabasz Index

permetrics.utils.cluster_util.**calculate_completeness_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_czekanowski_dice_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_davies_bouldin_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*)

permetrics.utils.cluster_util.**calculate_density_based_clustering_validation_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=1.0*)

permetrics.utils.cluster_util.**calculate_det_ratio_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*)

permetrics.utils.cluster_util.**calculate_duda_hart_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*)

permetrics.utils.cluster_util.**calculate_dunn_index**(*X=None*, *y_pred=None*, *use_modified=True*, *force_finite=True*, *finite_value=0.0*)

permetrics.utils.cluster_util.**calculate_entropy_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_f_measure_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_fowlkes_mallows_score**(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*)

permetrics.utils.cluster_util.**calculate_gamma_score**(*y_true=None*, *y_pred=None*)
Cluster Validation for Mixed-Type Data: Paper

permetrics.utils.cluster_util.**calculate_gplus_score**(*y_true=None*, *y_pred=None*)
Cluster Validation for Mixed-Type Data: Paper

permetrics.utils.cluster_util.**calculate_hartigan_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*)

permetrics.utils.cluster_util.**calculate_homogeneity_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_hubert_gamma_score**(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 1.0*)

permetrics.utils.cluster_util.**calculate_jaccard_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_ksq_detw_index**(*X=None*, *y_pred=None*, *use_normalized=True*)

permetrics.utils.cluster_util.**calculate_kulczynski_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_log_det_ratio_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=-10000000000.0*)

permetrics.utils.cluster_util.**calculate_mc_nemar_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_mean_squared_error_index**(*X=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_mutual_info_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_normalized_mutual_info_score**(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*)

permetrics.utils.cluster_util.**calculate_phi_score**(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*)

permetrics.utils.cluster_util.**calculate_precision_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_purity_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_r_squared_index**(*X=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_rand_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_recall_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_rogers_tanimoto_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_russel_rao_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_silhouette_index**(*X=None*, *y_pred=None*, *multi_output=False*, *force_finite=True*, *finite_value=- 1.0*)

Calculates the silhouette score for a given clustering.

Parameters

- **data** – A numpy array of shape (n_samples, n_features) representing the data points.
- **labels** – A numpy array of shape (n_samples,) containing the cluster labels for each data point.

Returns The silhouette score, a value between -1 and 1.

permetrics.utils.cluster_util.**calculate_silhouette_index_ver1**(*X=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_silhouette_index_ver2**(*X=None*, *y_pred=None*, *multi_output=False*, *force_finite=True*, *finite_value=-1.0*)

permetrics.utils.cluster_util.**calculate_silhouette_index_ver3**(*X=None*, *y_pred=None*, *multi_output=False*, *force_finite=True*, *finite_value=-1.0*)

permetrics.utils.cluster_util.**calculate_sokal_sneath1_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_sokal_sneath2_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_sum_squared_error_index**(*X=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_tau_score**(*y_true=None*, *y_pred=None*)
Cluster Validation for Mixed-Type Data: Paper

permetrics.utils.cluster_util.**calculate_v_measure_score**(*y_true=None*, *y_pred=None*)

permetrics.utils.cluster_util.**calculate_xie_beni_index**(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*)

permetrics.utils.cluster_util.**compute_BGSS**(*X*, *labels*)
The between-group dispersion BGSS or between-cluster variance

permetrics.utils.cluster_util.**compute_TSS**(*X*)

permetrics.utils.cluster_util.**compute_WG**(*X*)

permetrics.utils.cluster_util.**compute_WGSS**(*X*, *labels*)
Calculate the pooled within-cluster sum of squares WGSS or The within-cluster variance

permetrics.utils.cluster_util.**compute_barycenters**(*X*, *labels*)
Get the barycenter for each cluster and barycenter for all observations

Parameters

- **X** (*np.ndarray*) – The features of datasets
- **labels** (*np.ndarray*) – The predicted labels

Returns The barycenter for each clusters in form of matrix overall_barycenter (np.ndarray): the barycenter for all observations

Return type barycenters (np.ndarray)

permetrics.utils.cluster_util.**compute_clusters**(*labels*)
Get the dict of clusters and dict of cluster size

permetrics.utils.cluster_util.**compute_conditional_entropy**(*y_true*, *y_pred*)

permetrics.utils.cluster_util.**compute_confusion_matrix**(*y_true*, *y_pred*, *normalize=False*)
Computes the confusion matrix for a clustering problem given the true labels and the predicted labels. http://cran.nexr.com/web/packages/clusterCrit/vignettes/clusterCrit.pdf

permetrics.utils.cluster_util.**compute_contingency_matrix**(*y_true*, *y_pred*)

permetrics.utils.cluster_util.**compute_entropy**(*labels*)

permetrics.utils.cluster_util.**compute_nd_splus_sminus_t**(*y_true=None*, *y_pred=None*)
concordant_discordant

6.1.3 permetrics.utils.data_util module

permetrics.utils.data_util.**format_classification_data**(*y_true: numpy.ndarray*, *y_pred: numpy.ndarray*)

permetrics.utils.data_util.**format_external_clustering_data**(*y_true: numpy.ndarray*, *y_pred: numpy.ndarray*)

Need both of y_true and y_pred to format

permetrics.utils.data_util.**format_internal_clustering_data**(*y_pred: numpy.ndarray*)

permetrics.utils.data_util.**format_regression_data_type**(*y_true: numpy.ndarray*, *y_pred: numpy.ndarray*)

permetrics.utils.data_util.**format_y_score**(*y_true: numpy.ndarray*, *y_score: numpy.ndarray*)

permetrics.utils.data_util.**get_regression_non_zero_data**(*y_true*, *y_pred*, *one_dim=True*, *rule_idx=0*)

Get non-zero data based on rule

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **one_dim** (*bool*) – is y_true has 1 dimensions or not
- **rule_idx** (*int*) – valid values [0, 1, 2] corresponding to [y_true, y_pred, both true and pred]

Returns y_true with positive values based on rule y_pred: y_pred with positive values based on rule

Return type y_true

permetrics.utils.data_util.**get_regression_positive_data**(*y_true*, *y_pred*, *one_dim=True*, *rule_idx=0*)

Get positive data based on rule

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **one_dim** (*bool*) – is y_true has 1 dimensions or not
- **rule_idx** (*int*) – valid values [0, 1, 2] corresponding to [y_true, y_pred, both true and pred]

Returns y_true with positive values based on rule y_pred: y_pred with positive values based on rule

Return type y_true

permetrics.utils.data_util.**is_unique_labels_consecutive_and_start_zero**(*vector*)

6.1.4 permetrics.utils.encoder module

class permetrics.utils.encoder.**LabelEncoder**
Bases: object

fit(*y*)

fit_transform(*y*)

inverse_transform(*y*)

transform(*y*)

6.1.5 permetrics.utils.regressor_util module

permetrics.utils.regressor_util.**calculate_absolute_pcc**(*y_true*, *y_pred*)

permetrics.utils.regressor_util.**calculate_ec**(*y_true*, *y_pred*)

permetrics.utils.regressor_util.**calculate_entropy**(*y_true*, *y_pred*)

permetrics.utils.regressor_util.**calculate_mse**(*y_true*, *y_pred*)

permetrics.utils.regressor_util.**calculate_nse**(*y_true*, *y_pred*)

permetrics.utils.regressor_util.**calculate_pcc**(*y_true*, *y_pred*)

permetrics.utils.regressor_util.**calculate_wi**(*y_true*, *y_pred*)

6.2 permetrics.evaluator module

class permetrics.evaluator.**Evaluator**(*y_true=None*, *y_pred=None*, ***kwargs*)
Bases: object

This is base class for all performance metrics

EPSILON = 1e-10

SUPPORT = {}

get_metric_by_name(*metric_name=<class 'str'>*, *paras=None*) → dict
Get single metric by name, specific parameter of metric by dictionary

Parameters

- **metric_name** (*str*) – Select name of metric
- **paras** (*dict*) – Dictionary of hyper-parameter for that metric

Returns { metric_name: value }

Return type result (dict)

get_metrics_by_dict(*metrics_dict: dict*) → dict
Get results of list metrics by its name and parameters wrapped by dictionary

For example:

{ “RMSE”: {“multi_output”: multi_output}, “MAE”: {“multi_output”: multi_output}

}

Parameters **metrics_dict** (*dict*) – key is metric name and value is dict of parameters

Returns e.g, { "RMSE": 0.3524, "MAE": 0.445263 }

Return type results (dict)

get_metrics_by_list_names(*list_metric_names=<class 'list'>*, *list_paras=None*) → dict
Get results of list metrics by its name and parameters

Parameters

- **list_metric_names** (*list*) – e.g, ["RMSE", "MAE", "MAPE"]
- **list_paras** (*list*) – e.g, [{"multi_output": "raw_values"}, {"multi_output": "raw_values"}, {"multi_output": [2, 3]}]

Returns e.g, { "RMSE": 0.25, "MAE": [0.3, 0.6], "MAPE": 0.15 }

Return type results (dict)

get_output_result(*result=None*, *n_out=None*, *multi_output=None*, *force_finite=None*, *finite_value=None*)
Get final output result based on selected parameter

Parameters

- **result** – The raw result from metric
- **n_out** – The number of column in y_true or y_pred
- **multi_output** – *raw_values* - return multi-output, *weights* - return single output based on weights, else - return mean result
- **force_finite** – Make result as finite number
- **finite_value** – The value that used to replace the infinite value or NaN value.

Returns Final output results based on selected parameter

Return type final_result

get_processed_data(*y_true=None*, *y_pred=None*)

set_keyword_arguments(*kwargs*)

6.3 permetrics.regression module

class permetrics.regression.**RegressionMetric**(*y_true=None*, *y_pred=None*, ***kwargs*)
Bases: *permetrics.evaluator.Evaluator*

Defines a RegressionMetric class that hold all regression metrics (for both regression and time-series problems)

- An extension of scikit-learn metrics section, with the addition of many more regression metrics.
- https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics
- Some methods in scikit-learn can't generate the multi-output metrics, we re-implement all of them and allow multi-output metrics
- Therefore, we support calculate the multi-output metrics for all methods

Parameters

- **y_true** (*tuple, list, np.ndarray, default = None*) – The ground truth values.
- **y_pred** (*tuple, list, np.ndarray, default = None*) – The prediction values.

A10(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
A10 index (A10): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Notes

- a10-index is engineering index for evaluating artificial intelligence models by showing the number of samples
- that fit the prediction values with a deviation of ±10% compared to experimental values
- https://www.mdpi.com/2076-3417/9/18/3715/htm

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns A10 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

A20(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
A20 index (A20): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Notes

- a20-index evaluated metric by showing the number of samples that fit the prediction values with a deviation of ±20% compared to experimental values
- https://www.mdpi.com/2076-3417/9/18/3715/htm

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns A20 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

A30(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
A30 index (A30): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Note: a30-index evaluated metric by showing the number of samples that fit the prediction values with a deviation of ±30% compared to experimental values

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns A30 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

ACOD(*y_true=None*, *y_pred=None*, *X_shape=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Adjusted Coefficient of Determination (ACOD/AR2): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- https://dziganto.github.io/data%20science/linear%20regression/machine%20learning/python/Linear-Regression-101-Metrics/
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is PCC.
- We should denote it as COD or R2 only.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **X_shape** (*tuple, list, np.ndarray*) – The shape of X_train dataset
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns AR2 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

AE(*y_true=None*, *y_pred=None*, ***kwargs*)

Absolute Error (AE): Best possible score is 0.0, smaller value is better. Range = (-inf, +inf) Note: Computes the absolute error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns AE metric

Return type result (np.ndarray)

APCC(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Absolute Pearson's Correlation Coefficient (APCC or AR): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns AR metric for single column or multiple columns

Return type result (float, int, np.ndarray)

AR(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Absolute Pearson's Correlation Coefficient (APCC or AR): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns AR metric for single column or multiple columns

Return type result (float, int, np.ndarray)

AR2(*y_true=None*, *y_pred=None*, *X_shape=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Adjusted Coefficient of Determination (ACOD/AR2): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- https://dziganto.github.io/data%20science/linear%20regression/machine%20learning/python/Linear-Regression-101-Metrics/
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is PCC.
- We should denote it as COD or R2 only.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **X_shape** (*tuple, list, np.ndarray*) – The shape of X_train dataset
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns AR2 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

CE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Cross Entropy (CE): Range = (-inf, 0]. Can't give any comment about this one

Notes

- Greater value of Entropy, the greater the uncertainty for probability distribution and smaller the value the less the uncertainty
- https://datascience.stackexchange.com/questions/20296/cross-entropy-loss-explanation

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns CE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

CI(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Confidence Index (or Performance Index): CI (PI): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- Reference evapotranspiration for Londrina, Paraná, Brazil: performance of different estimation methods
- > 0.85, Excellent
- 0.76-0.85, Very good
- 0.66-0.75, Good
- 0.61-0.65, Satisfactory
- 0.51-0.60, Poor
- 0.41-0.50, Bad
- < 0.40, Very bad

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns CI (PI) metric for single column or multiple columns

Return type result (float, int, np.ndarray)

COD(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Coefficient of Determination (COD/R2): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is PCC.
- We should denote it as COD or R2 only.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R2 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

COR(*y_true=None*, *y_pred=None*, *sample=False*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Correlation (COR): Best possible value = 1, bigger value is better. Range = [-1, +1]

- measures the strength of the relationship between variables
- is the scaled measure of covariance. It is dimensionless.
- the correlation coefficient is always a pure value and not measured in any units.

Links:

- https://corporatefinanceinstitute.com/resources/data-science/covariance/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **sample** (*bool*) – sample covariance or population covariance. See the website above for more details
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns COR metric for single column or multiple columns

Return type result (float, int, np.ndarray)

COV(*y_true=None*, *y_pred=None*, *sample=False*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=-10.0*, ***kwargs*)

Covariance (COV): There is no best value, bigger value is better. Range = [-inf, +inf)

- is a measure of the relationship between two random variables
- evaluates how much – to what extent – the variables change together
- does not assess the dependency between variables
- Positive covariance: Indicates that two variables tend to move in the same direction.
- Negative covariance: Reveals that two variables tend to move in inverse directions.

Links:

- https://corporatefinanceinstitute.com/resources/data-science/covariance/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **sample** (*bool*) – sample covariance or population covariance. See the website above for more details
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns COV metric for single column or multiple columns

Return type result (float, int, np.ndarray)

CRM(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Coefficient of Residual Mass (CRM): Best possible value = 0.0, smaller value is better. Range = [-inf, +inf]

Links:

- https://doi.org/10.1016/j.csite.2022.101797

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns CRM metric for single column or multiple columns

Return type result (float, int, np.ndarray)

DRV(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=10.0*, ***kwargs*)

Deviation of Runoff Volume (DRV): Best possible score is 1.0, smaller value is better. Range = [0, +inf)
Link: https://rstudio-pubs-static.s3.amazonaws.com/433152_56d00c1e29724829bad5fc4fd8c8ebff.html

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns DRV metric for single column or multiple columns

Return type result (float, int, np.ndarray)

EC(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Efficiency Coefficient (EC): Best possible value = 1, bigger value is better. Range = [-inf, +1]

Links:

- https://doi.org/10.1016/j.solener.2019.01.037

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns EC metric for single column or multiple columns

Return type result (float, int, np.ndarray)

EVS(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Explained Variance Score (EVS). Best possible score is 1.0, greater value is better. Range = (-inf, 1.0]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns EVS metric for single column or multiple columns

Return type result (float, int, np.ndarray)

GINI(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Gini coefficient (GINI): Best possible score is 1, bigger value is better. Range = [0, 1]

Notes

- This version is based on below repository matlab code.
- https://github.com/benhamner/Metrics/blob/master/MATLAB/metrics/gini.m

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns Gini metric for single column or multiple columns

Return type result (float, int, np.ndarray)

GINI_WIKI(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Gini coefficient (GINI_WIKI): Best possible score is 1, bigger value is better. Range = [0, 1]

Notes

- This version is based on wiki page, may be is the true version
- https://en.wikipedia.org/wiki/Gini_coefficient
- Gini coefficient can theoretically range from 0 (complete equality) to 1 (complete inequality)
- It is sometimes expressed as a percentage ranging between 0 and 100.
- If negative values are possible, then the Gini coefficient could theoretically be more than 1.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns Gini metric for single column or multiple columns

Return type result (float, int, np.ndarray)

JSD(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Jensen-Shannon Divergence (JSD): Best possible score is 0.0 (identical), smaller value is better . Range = [0, +inf) Link: https://machinelearningmastery.com/divergence-between-probability-distributions/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns JSD metric (bits) for single column or multiple columns

Return type result (float, int, np.ndarray)

KGE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Kling-Gupta Efficiency (KGE): Best possible score is 1, bigger value is better. Range = (-inf, 1] Link: https://rstudio-pubs-static.s3.amazonaws.com/433152_56d00c1e29724829bad5fc4fd8c8ebff.html

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns KGE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

KLD(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Kullback-Leibler Divergence (KLD): Best possible score is 0.0 . Range = (-inf, +inf) Link: https://machinelearningmastery.com/divergence-between-probability-distributions/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns KLD metric (bits) for single column or multiple columns

Return type result (float, int, np.ndarray)

MAAPE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Arctangent Absolute Percentage Error (MAAPE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MAAPE metric for single column or multiple columns (radian values)

Return type result (float, int, np.ndarray)

MAE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Absolute Error (MAE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MAE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MAPE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Absolute Percentage Error (MAPE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MAPE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MASE(*y_true=None*, *y_pred=None*, *m=1*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Mean Absolute Scaled Error (MASE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)
Link: https://en.wikipedia.org/wiki/Mean_absolute_scaled_error

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **m** (*int*) – m = 1 for non-seasonal data, m > 1 for seasonal data
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MASE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MBE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Mean Bias Error (MBE): Best possible score is 0.0. Range = (-inf, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MBE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

ME(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Max Error (ME): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)

- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns ME metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MPE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Mean Percentage Error (MPE): Best possible score is 0.0. Range = (-inf, +inf) Link: https://www.dataquest.io/blog/understanding-regression-error-metrics/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MPE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MRB(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Mean Relative Error (MRE) - Mean Relative Bias (MRB): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MRE (MRB) metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MRE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Mean Relative Error (MRE) - Mean Relative Bias (MRB): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")

- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MRE (MRB) metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MSE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Squared Error (MSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MSLE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Squared Log Error (MSLE): Best possible score is 0.0, smaller value is better. Range = [0, +inf) Link: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-logarithmic-error-(msle)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MSLE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

MedAE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Median Absolute Error (MedAE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MedAE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

NNSE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Normalize Nash-Sutcliffe Efficiency (NNSE): Best possible score is 1.0, bigger value is better. Range = [0, 1] Link: https://agrimetsoft.com/calculators/Nash%20Sutcliffe%20model%20Efficiency%20coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns NSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

NRMSE(*y_true=None*, *y_pred=None*, *model=0*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Normalized Root Mean Square Error (NRMSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Link: https://medium.com/microsoftazure/how-to-better-evaluate-the-goodness-of-fit-of-regressions-990dbf1c0091

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **model** (*int*) – Normalize RMSE by different ways, (Optional, default = 0, valid values = [0, 1, 2, 3]
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns NRMSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

NSE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Nash-Sutcliffe Efficiency (NSE): Best possible score is 1.0, bigger value is better. Range = (-inf, 1] Link: https://agrimetsoft.com/calculators/Nash%20Sutcliffe%20model%20Efficiency%20coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns NSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

OI(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Overall Index (OI): Best possible value = 1, bigger value is better. Range = [-inf, +1]

Links:

- https://doi.org/10.1016/j.solener.2019.01.037

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns OI metric for single column or multiple columns

Return type result (float, int, np.ndarray)

PCC(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)
Pearson's Correlation Coefficient (PCC or R): Best possible score is 1.0, bigger value is better. Range = [-1, 1] .. rubric:: Notes

- Reference evapotranspiration for Londrina, Paraná, Brazil: performance of different estimation methods
- Remember no absolute in the equations
- https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values

- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R metric for single column or multiple columns

Return type result (float, int, np.ndarray)

PCD(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Prediction of Change in Direction (PCD): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns PCD metric for single column or multiple columns

Return type result (float, int, np.ndarray)

R(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)
Pearson's Correlation Coefficient (PCC or R): Best possible score is 1.0, bigger value is better. Range = [-1, 1] .. rubric:: Notes

- Reference evapotranspiration for Londrina, Paraná, Brazil: performance of different estimation methods
- Remember no absolute in the equations
- https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R metric for single column or multiple columns

Return type result (float, int, np.ndarray)

R2(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Coefficient of Determination (COD/R2): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is PCC.
- We should denote it as COD or R2 only.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R2 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

R2S(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
(Pearson's Correlation Index)^2 = R^2 = R2S = RSQ (R square): Best possible score is 1.0, bigger value is better. Range = [0, 1] .. rubric:: Notes

- Do not misunderstand between R2s and R2 (Coefficient of Determination), they are different
- Most of online tutorials (article, wikipedia,...) or even scikit-learn library are denoted the wrong R2s and R2.
- R^2 = R2s = R squared should be (Pearson's Correlation Index)^2
- Meanwhile, R2 = Coefficient of Determination
- https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R2s metric for single column or multiple columns

Return type result (float, int, np.ndarray)

RAE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Relative Absolute Error (RAE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Notes

- https://stackoverflow.com/questions/59499222/how-to-make-a-function-of-mae-and-rae-without-using-librarymetrics
- https://www.statisticshowto.com/relative-absolute-error

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns RAE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

RB(*y_true=None*, *y_pred=None*, ***kwargs*)
Relative Error (RE): Best possible score is 0.0, smaller value is better. Range = (-inf, +inf) Note: Computes the relative error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns RE metric

Return type result (np.ndarray)

RE(*y_true=None*, *y_pred=None*, ***kwargs*)
Relative Error (RE): Best possible score is 0.0, smaller value is better. Range = (-inf, +inf) Note: Computes the relative error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns RE metric

Return type result (np.ndarray)

RMSE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Root Mean Squared Error (RMSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns RMSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

RSE(*y_true=None*, *y_pred=None*, *n_paras=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Residual Standard Error (RSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Links:

- https://www.statology.org/residual-standard-error-r/
- https://machinelearningmastery.com/degrees-of-freedom-in-machine-learning/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **n_paras** (*int*) – The number of model's parameters
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns RSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

RSQ(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

(Pearson's Correlation Index)^2 = R^2 = R2S = RSQ (R square): Best possible score is 1.0, bigger value is better. Range = [0, 1] .. rubric:: Notes

- Do not misunderstand between R2s and R2 (Coefficient of Determination), they are different
- Most of online tutorials (article, wikipedia,...) or even scikit-learn library are denoted the wrong R2s and R2.
- R^2 = R2s = R squared should be (Pearson's Correlation Index)^2
- Meanwhile, R2 = Coefficient of Determination
- https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values

- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R2s metric for single column or multiple columns

Return type result (float, int, np.ndarray)

SE(*y_true=None*, *y_pred=None*, ***kwargs*)

Squared Error (SE): Best possible score is 0.0, smaller value is better. Range = [0, +inf) Note: Computes the squared error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns SE metric

Return type result (np.ndarray)

SLE(*y_true=None*, *y_pred=None*, ***kwargs*)

Squared Log Error (SLE): Best possible score is 0.0, smaller value is better. Range = [0, +inf) Note: Computes the squared log error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns SLE metric

Return type result (np.ndarray)

SMAPE(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Symmetric Mean Absolute Percentage Error (SMAPE): Best possible score is 0.0, smaller value is better. Range = [0, 1] If you want percentage then multiply with 100%

Link: https://en.wikipedia.org/wiki/Symmetric_mean_absolute_percentage_error

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns SMAPE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

```
SUPPORT = {'A10': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'A20':
{'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'A30': {'best': '1', 'range':
'[0, 1]', 'type': 'max'}, 'ACOD': {'best': '1', 'range': '(-inf, 1]', 'type':
'max'}, 'AE': {'best': '0', 'range': '(-inf, +inf)', 'type': 'unknown'}, 'APCC':
{'best': '1', 'range': '[-1, 1]', 'type': 'max'}, 'AR': {'best': '1', 'range':
'[-1, 1]', 'type': 'max'}, 'AR2': {'best': '1', 'range': '(-inf, 1]', 'type':
'max'}, 'CE': {'best': 'unknown', 'range': '(-inf, 0]', 'type': 'unknown'}, 'CI':
{'best': '1', 'range': '(-inf, 1]', 'type': 'max'}, 'COD': {'best': '1',
'range': '(-inf, 1]', 'type': 'max'}, 'COR': {'best': '1', 'range': '[-1, 1]',
'type': 'max'}, 'COV': {'best': 'no best', 'range': '(-inf, +inf)', 'type':
'max'}, 'CRM': {'best': '0', 'range': '(-inf, +inf)', 'type': 'min'}, 'DRV':
{'best': '1', 'range': '[1, +inf)', 'type': 'min'}, 'EC': {'best': '1', 'range':
'(-inf, 1]', 'type': 'max'}, 'EVS': {'best': '1', 'range': '(-inf, 1]', 'type':
'max'}, 'GINI': {'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'GINI_WIKI':
{'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'JSD': {'best': '0',
'range': '[0, +inf)', 'type': 'min'}, 'KGE': {'best': '1', 'range': '(-inf, 1]',
'type': 'max'}, 'KLD': {'best': '0', 'range': '(-inf, +inf)', 'type':
'unknown'}, 'MAAPE': {'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'MAE':
{'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'MAPE': {'best': '0',
'range': '[0, +inf)', 'type': 'min'}, 'MASE': {'best': '0', 'range': '[0,
+inf)', 'type': 'min'}, 'MBE': {'best': '0', 'range': '(-inf, +inf)', 'type':
'unknown'}, 'ME': {'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'MPE':
{'best': '0', 'range': '(-inf, +inf)', 'type': 'unknown'}, 'MRB': {'best': '0',
'range': '[0, +inf)', 'type': 'min'}, 'MRE': {'best': '0', 'range': '[0, +inf)',
'type': 'min'}, 'MSE': {'best': '0', 'range': '[0, +inf)', 'type': 'min'},
'MSLE': {'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'MedAE': {'best':
'0', 'range': '[0, +inf)', 'type': 'min'}, 'NNSE': {'best': '1', 'range': '[0,
1]', 'type': 'max'}, 'NRMSE': {'best': '0', 'range': '[0, +inf)', 'type':
'min'}, 'NSE': {'best': '1', 'range': '(-inf, 1]', 'type': 'max'}, 'OI': {'best':
'1', 'range': '(-inf, 1]', 'type': 'max'}, 'PCC': {'best': '1', 'range': '[-1,
1]', 'type': 'max'}, 'PCD': {'best': '1', 'range': '[0, 1]', 'type': 'max'},
'R': {'best': '1', 'range': '[-1, 1]', 'type': 'max'}, 'R2': {'best': '1',
'range': '(-inf, 1]', 'type': 'max'}, 'R2S': {'best': '1', 'range': '[0, 1]',
'type': 'max'}, 'RAE': {'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'RB':
{'best': '0', 'range': '(-inf, +inf)', 'type': 'unknown'}, 'RE': {'best': '0',
'range': '(-inf, +inf)', 'type': 'unknown'}, 'RMSE': {'best': '0', 'range': '[0,
+inf)', 'type': 'min'}, 'RSE': {'best': '0', 'range': '[0, +inf)', 'type':
'min'}, 'RSQ': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'SE': {'best':
'0', 'range': '[0, +inf)', 'type': 'min'}, 'SLE': {'best': '0', 'range': '[0,
+inf)', 'type': 'min'}, 'SMAPE': {'best': '0', 'range': '[0, 1]', 'type':
'min'}, 'VAF': {'best': '100', 'range': '(-inf, 100%)', 'type': 'max'}, 'WI':
{'best': '1', 'range': '[0, 1]', 'type': 'max'}}
```

VAF(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Variance Accounted For between 2 signals (VAF): Best possible score is 100% (identical signal), bigger value is better. Range = (-inf, 100%] Link: https://www.dcsc.tudelft.nl/~jwvanwingerden/lti/doc/html/vaf.html

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns VAF metric for single column or multiple columns

Return type result (float, int, np.ndarray)

WI(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Willmott Index (WI): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Notes

- Reference evapotranspiration for Londrina, Paraná, Brazil: performance of different estimation methods
- https://www.researchgate.net/publication/319699360_Reference_evapotranspiration_for_Londrina_Parana_Brazil_performance_of_different_estimation_methods

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns WI metric for single column or multiple columns

Return type result (float, int, np.ndarray)

a10_index(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

A10 index (A10): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Notes

- a10-index is engineering index for evaluating artificial intelligence models by showing the number of samples
- that fit the prediction values with a deviation of ±10% compared to experimental values
- https://www.mdpi.com/2076-3417/9/18/3715/htm

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns A10 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

a20_index(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

A20 index (A20): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Notes

- a20-index evaluated metric by showing the number of samples that fit the prediction values with a deviation of ±20% compared to experimental values
- https://www.mdpi.com/2076-3417/9/18/3715/htm

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns A20 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

a30_index(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

A30 index (A30): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Note: a30-index evaluated metric by showing the number of samples that fit the prediction values with a deviation of ±30% compared to experimental values

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)

- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns A30 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

absolute_pearson_correlation_coefficient(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Absolute Pearson's Correlation Coefficient (APCC or AR): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns AR metric for single column or multiple columns

Return type result (float, int, np.ndarray)

adjusted_coefficient_of_determination(*y_true=None*, *y_pred=None*, *X_shape=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Adjusted Coefficient of Determination (ACOD/AR2): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- https://dziganto.github.io/data%20science/linear%20regression/machine%20learning/python/Linear-Regression-101-Metrics/
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is PCC.
- We should denote it as COD or R2 only.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **X_shape** (*tuple, list, np.ndarray*) – The shape of X_train dataset
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns AR2 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

coefficient_of_determination(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Coefficient of Determination (COD/R2): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score
- Scikit-learn and other websites denoted COD as R^2 (or R squared), it leads to the misunderstanding of R^2 in which R is PCC.
- We should denote it as COD or R2 only.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R2 metric for single column or multiple columns

Return type result (float, int, np.ndarray)

coefficient_of_residual_mass(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Coefficient of Residual Mass (CRM): Best possible value = 0.0, smaller value is better. Range = [-inf, +inf]

Links:

- https://doi.org/10.1016/j.csite.2022.101797

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns CRM metric for single column or multiple columns

Return type result (float, int, np.ndarray)

confidence_index(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Confidence Index (or Performance Index): CI (PI): Best possible score is 1.0, bigger value is better. Range = (-inf, 1]

Notes

- Reference evapotranspiration for Londrina, Paraná, Brazil: performance of different estimation methods
- > 0.85, Excellent
- 0.76-0.85, Very good
- 0.66-0.75, Good
- 0.61-0.65, Satisfactory
- 0.51-0.60, Poor
- 0.41-0.50, Bad
- < 0.40, Very bad

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns CI (PI) metric for single column or multiple columns

Return type result (float, int, np.ndarray)

correlation(*y_true=None*, *y_pred=None*, *sample=False*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Correlation (COR): Best possible value = 1, bigger value is better. Range = [-1, +1]

- measures the strength of the relationship between variables
- is the scaled measure of covariance. It is dimensionless.
- the correlation coefficient is always a pure value and not measured in any units.

Links:

- https://corporatefinanceinstitute.com/resources/data-science/covariance/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **sample** (*bool*) – sample covariance or population covariance. See the website above for more details
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns COR metric for single column or multiple columns

Return type result (float, int, np.ndarray)

covariance(*y_true=None*, *y_pred=None*, *sample=False*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 10.0*, ***kwargs*)

Covariance (COV): There is no best value, bigger value is better. Range = [-inf, +inf)

- is a measure of the relationship between two random variables
- evaluates how much – to what extent – the variables change together
- does not assess the dependency between variables
- Positive covariance: Indicates that two variables tend to move in the same direction.
- Negative covariance: Reveals that two variables tend to move in inverse directions.

Links:

- https://corporatefinanceinstitute.com/resources/data-science/covariance/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **sample** (*bool*) – sample covariance or population covariance. See the website above for more details
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns COV metric for single column or multiple columns

Return type result (float, int, np.ndarray)

cross_entropy(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Cross Entropy (CE): Range = (-inf, 0]. Can't give any comment about this one

Notes

- Greater value of Entropy, the greater the uncertainty for probability distribution and smaller the value the less the uncertainty
- https://datascience.stackexchange.com/questions/20296/cross-entropy-loss-explanation

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns CE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

deviation_of_runoff_volume(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=10.0*, ***kwargs*)

Deviation of Runoff Volume (DRV): Best possible score is 1.0, smaller value is better. Range = [0, +inf)
Link: https://rstudio-pubs-static.s3.amazonaws.com/433152_56d00c1e29724829bad5fc4fd8c8ebff.html

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns DRV metric for single column or multiple columns

Return type result (float, int, np.ndarray)

efficiency_coefficient(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Efficiency Coefficient (EC): Best possible value = 1, bigger value is better. Range = [-inf, +1]

Links:

- https://doi.org/10.1016/j.solener.2019.01.037

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns EC metric for single column or multiple columns

Return type result (float, int, np.ndarray)

explained_variance_score(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Explained Variance Score (EVS). Best possible score is 1.0, greater value is better. Range = (-inf, 1.0]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns EVS metric for single column or multiple columns

Return type result (float, int, np.ndarray)

get_processed_data(*y_true=None*, *y_pred=None*, ***kwargs*)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns y_true used in evaluation process. y_pred_final: y_pred used in evaluation process n_out: Number of outputs

Return type y_true_final

static get_support(*name=None*, *verbose=True*)

gini_coefficient(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Gini coefficient (GINI): Best possible score is 1, bigger value is better. Range = [0, 1]

Notes

- This version is based on below repository matlab code.
- https://github.com/benhamner/Metrics/blob/master/MATLAB/metrics/gini.m

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns Gini metric for single column or multiple columns

Return type result (float, int, np.ndarray)

gini_coefficient_wiki(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Gini coefficient (GINI_WIKI): Best possible score is 1, bigger value is better. Range = [0, 1]

Notes

- This version is based on wiki page, may be is the true version
- https://en.wikipedia.org/wiki/Gini_coefficient
- Gini coefficient can theoretically range from 0 (complete equality) to 1 (complete inequality)
- It is sometimes expressed as a percentage ranging between 0 and 100.
- If negative values are possible, then the Gini coefficient could theoretically be more than 1.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns Gini metric for single column or multiple columns

Return type result (float, int, np.ndarray)

jensen_shannon_divergence(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Jensen-Shannon Divergence (JSD): Best possible score is 0.0 (identical), smaller value is better . Range = [0, +inf) Link: https://machinelearningmastery.com/divergence-between-probability-distributions/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns JSD metric (bits) for single column or multiple columns

Return type result (float, int, np.ndarray)

kling_gupta_efficiency(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Kling-Gupta Efficiency (KGE): Best possible score is 1, bigger value is better. Range = (-inf, 1] Link: https://rstudio-pubs-static.s3.amazonaws.com/433152_56d00c1e29724829bad5fc4fd8c8ebff.html

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns KGE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

kullback_leibler_divergence(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Kullback-Leibler Divergence (KLD): Best possible score is 0.0 . Range = (-inf, +inf) Link: https://machinelearningmastery.com/divergence-between-probability-distributions/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)

- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns KLD metric (bits) for single column or multiple columns

Return type result (float, int, np.ndarray)

max_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Max Error (ME): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns ME metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_absolute_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Absolute Error (MAE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MAE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_absolute_percentage_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Absolute Percentage Error (MAPE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")

- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MAPE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_absolute_scaled_error(*y_true=None*, *y_pred=None*, *m=1*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Absolute Scaled Error (MASE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)
Link: https://en.wikipedia.org/wiki/Mean_absolute_scaled_error

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **m** (*int*) – m = 1 for non-seasonal data, m > 1 for seasonal data
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MASE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_arctangent_absolute_percentage_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Arctangent Absolute Percentage Error (MAAPE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MAAPE metric for single column or multiple columns (radian values)

Return type result (float, int, np.ndarray)

mean_bias_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Bias Error (MBE): Best possible score is 0.0. Range = (-inf, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MBE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_percentage_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Percentage Error (MPE): Best possible score is 0.0. Range = (-inf, +inf) Link: https://www.dataquest.io/blog/understanding-regression-error-metrics/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MPE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_relative_bias(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Relative Error (MRE) - Mean Relative Bias (MRB): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MRE (MRB) metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_relative_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Relative Error (MRE) - Mean Relative Bias (MRB): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MRE (MRB) metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_squared_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Squared Error (MSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

mean_squared_log_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Mean Squared Log Error (MSLE): Best possible score is 0.0, smaller value is better. Range = [0, +inf) Link: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-logarithmic-error-(msle)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)

- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MSLE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

median_absolute_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Median Absolute Error (MedAE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns MedAE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

nash_sutcliffe_efficiency(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Nash-Sutcliffe Efficiency (NSE): Best possible score is 1.0, bigger value is better. Range = (-inf, 1] Link: https://agrimetsoft.com/calculators/Nash%20Sutcliffe%20model%20Efficiency%20coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns NSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

normalized_nash_sutcliffe_efficiency(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Normalize Nash-Sutcliffe Efficiency (NNSE): Best possible score is 1.0, bigger value is better. Range = [0, 1] Link: https://agrimetsoft.com/calculators/Nash%20Sutcliffe%20model%20Efficiency%20coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns NSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

normalized_root_mean_square_error(*y_true=None*, *y_pred=None*, *model=0*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Normalized Root Mean Square Error (NRMSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Link: https://medium.com/microsoftazure/how-to-better-evaluate-the-goodness-of-fit-of-regressions-990dbf1c0091

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **model** (*int*) – Normalize RMSE by different ways, (Optional, default = 0, valid values = [0, 1, 2, 3]
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns NRMSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

overall_index(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Overall Index (OI): Best possible value = 1, bigger value is better. Range = [-inf, +1]

Links:

- https://doi.org/10.1016/j.solener.2019.01.037

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns OI metric for single column or multiple columns

Return type result (float, int, np.ndarray)

pearson_correlation_coefficient(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Pearson's Correlation Coefficient (PCC or R): Best possible score is 1.0, bigger value is better. Range = [-1, 1] .. rubric:: Notes

- Reference evapotranspiration for Londrina, Paraná, Brazil: performance of different estimation methods
- Remember no absolute in the equations
- https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R metric for single column or multiple columns

Return type result (float, int, np.ndarray)

pearson_correlation_coefficient_square(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

(Pearson's Correlation Index)^2 = R^2 = R2S = RSQ (R square): Best possible score is 1.0, bigger value is better. Range = [0, 1] .. rubric:: Notes

- Do not misunderstand between R2s and R2 (Coefficient of Determination), they are different
- Most of online tutorials (article, wikipedia,...) or even scikit-learn library are denoted the wrong R2s and R2.
- R^2 = R2s = R squared should be (Pearson's Correlation Index)^2
- Meanwhile, R2 = Coefficient of Determination
- https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)

- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns R2s metric for single column or multiple columns

Return type result (float, int, np.ndarray)

prediction_of_change_in_direction(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Prediction of Change in Direction (PCD): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns PCD metric for single column or multiple columns

Return type result (float, int, np.ndarray)

relative_absolute_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Relative Absolute Error (RAE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Notes

- https://stackoverflow.com/questions/59499222/how-to-make-a-function-of-mae-and-rae-without-using-librarymetrics
- https://www.statisticshowto.com/relative-absolute-error

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns RAE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

residual_standard_error(*y_true=None*, *y_pred=None*, *n_paras=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Residual Standard Error (RSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Links:

- https://www.statology.org/residual-standard-error-r/
- https://machinelearningmastery.com/degrees-of-freedom-in-machine-learning/

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **n_paras** (*int*) – The number of model's parameters
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns RSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

root_mean_squared_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)

Root Mean Squared Error (RMSE): Best possible score is 0.0, smaller value is better. Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns RMSE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

single_absolute_error(*y_true=None*, *y_pred=None*, ***kwargs*)

Absolute Error (AE): Best possible score is 0.0, smaller value is better. Range = (-inf, +inf) Note: Computes the absolute error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns AE metric

Return type result (np.ndarray)

single_relative_bias(*y_true=None*, *y_pred=None*, ***kwargs*)
Relative Error (RE): Best possible score is 0.0, smaller value is better. Range = (-inf, +inf) Note: Computes the relative error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns RE metric

Return type result (np.ndarray)

single_relative_error(*y_true=None*, *y_pred=None*, ***kwargs*)
Relative Error (RE): Best possible score is 0.0, smaller value is better. Range = (-inf, +inf) Note: Computes the relative error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns RE metric

Return type result (np.ndarray)

single_squared_error(*y_true=None*, *y_pred=None*, ***kwargs*)
Squared Error (SE): Best possible score is 0.0, smaller value is better. Range = [0, +inf) Note: Computes the squared error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns SE metric

Return type result (np.ndarray)

single_squared_log_error(*y_true=None*, *y_pred=None*, ***kwargs*)
Squared Log Error (SLE): Best possible score is 0.0, smaller value is better. Range = [0, +inf) Note: Computes the squared log error between two numbers, or for element between a pair of list, tuple or numpy arrays.

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns SLE metric

Return type result (np.ndarray)

symmetric_mean_absolute_percentage_error(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=1.0*, ***kwargs*)
Symmetric Mean Absolute Percentage Error (SMAPE): Best possible score is 0.0, smaller value is better. Range = [0, 1] If you want percentage then multiply with 100%

Link: https://en.wikipedia.org/wiki/Symmetric_mean_absolute_percentage_error

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values

- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns SMAPE metric for single column or multiple columns

Return type result (float, int, np.ndarray)

variance_accounted_for(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Variance Accounted For between 2 signals (VAF): Best possible score is 100% (identical signal), bigger value is better. Range = (-inf, 100%] Link: https://www.dcsc.tudelft.nl/~jwvanwingerden/lti/doc/html/vaf.html

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)
- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns VAF metric for single column or multiple columns

Return type result (float, int, np.ndarray)

willmott_index(*y_true=None*, *y_pred=None*, *multi_output='raw_values'*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Willmott Index (WI): Best possible score is 1.0, bigger value is better. Range = [0, 1]

Notes

- Reference evapotranspiration for Londrina, Paraná, Brazil: performance of different estimation methods
- https://www.researchgate.net/publication/319699360_Reference_evapotranspiration_for_Londrina_Parana_Brazil_performance_of_different_estimation_methods

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **multi_output** – Can be "raw_values" or list weights of variables such as [0.5, 0.2, 0.3] for 3 columns, (Optional, default = "raw_values")
- **force_finite** (*bool*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value* (Optional, default = True)

- **finite_value** (*float*) – The finite value used to replace Inf or NaN result (Optional, default = 0.0)

Returns WI metric for single column or multiple columns

Return type result (float, int, np.ndarray)

6.4 permetrics.classification module

class permetrics.classification.**ClassificationMetric**(*y_true=None*, *y_pred=None*, ***kwargs*)

Bases: *permetrics.evaluator.Evaluator*

Defines a ClassificationMetric class that hold all classification metrics (for both binary and multiple classification problem)

Parameters

- **y_true** (*tuple, list, np.ndarray, default = None*) – The ground truth values.
- **y_pred** (*tuple, list, np.ndarray, default = None*) – The prediction values.
- **labels** (*tuple, list, np.ndarray, default = None*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*(str, None): {'micro', 'macro', 'weighted'} or None, default="macro"*) – If None, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

  'micro': Calculate metrics globally by considering each element of the label indicator matrix as a label.

  'macro': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

  'weighted': Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).

AS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate accuracy score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the accuracy score

Return type accuracy (float, dict)

AUC(*y_true=None*, *y_pred=None*, *average='macro'*, ***kwargs*)

Calculates the ROC-AUC score between y_true and y_score. Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – A LIST OF PREDICTED SCORES (NOT LABELS)

- **average** (*str, None*) – {'macro', 'weighted'} or None, default="macro"

Returns The AUC score.

Return type float, dict

BSL(*y_true=None*, *y_pred=None*, ***kwargs*)
Calculates the Brier Score Loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of labels (or predicted scores in case of multi-class)

Returns The Brier Score Loss

Return type float, dict

CEL(*y_true=None*, *y_pred=None*, ***kwargs*)
Calculates the Cross-Entropy loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – A LIST OF PREDICTED SCORES (NOT LABELS)

Returns The Cross-Entropy loss

Return type float

CKS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate Cohen Kappa score for multiple classification problem Higher is better (Best = +1), Range = [-1, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Cohen Kappa score

Return type cks (float, dict)

CM(*y_true=None*, *y_pred=None*, *labels=None*, *normalize=None*, ***kwargs*)
Generate confusion matrix and useful information

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **normalize** (*'true', 'pred', 'all', None*) – Normalizes confusion matrix over the true (rows), predicted (columns) conditions or all the population.

Returns a 2-dimensional list of pairwise counts imap (dict): a map between label and index of confusion matrix imap_count (dict): a map between label and number of true label in y_true

Return type matrix (np.ndarray)

F1S(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate f1 score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the f1 score

Return type f1 (float, dict)

F2S(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate f2 score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the f2 score

Return type f2 (float, dict)

FBS(*y_true=None*, *y_pred=None*, *beta=1.0*, *labels=None*, *average='macro'*, ***kwargs*)
The beta parameter determines the weight of recall in the combined score. beta < 1 lends more weight to precision, while beta > 1 favors recall (beta -> 0 considers only precision, beta -> +inf only recall). Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **beta** (*float*) – the weight of recall in the combined score, default = 1.0
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the fbeta score

Return type fbeta (float, dict)

GINI(*y_true=None*, *y_pred=None*, ***kwargs*)
Calculates the Gini index between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes

Returns The Gini index

Return type float, dict

GMS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Calculates the G-mean (Geometric mean) score between y_true and y_pred. Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns The G-mean score.

Return type float, dict

HL(*y_true=None*, *y_pred=None*, ***kwargs*)

Calculates the Hinge loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of labels (or predicted scores in case of multi-class)

Returns The Hinge loss

Return type float

HS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate hamming score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the hamming score

Return type hl (float, dict)

JSC(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate Jaccard similarity index for multiple classification problem Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes

- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Jaccard similarity index

Return type jsi (float, dict)

JSI(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate Jaccard similarity index for multiple classification problem Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Jaccard similarity index

Return type jsi (float, dict)

KLDL(*y_true=None*, *y_pred=None*, ***kwargs*)

Calculates the Kullback-Leibler divergence loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of labels (or predicted scores in case of multi-class)

Returns The Kullback-Leibler divergence loss

Return type float

LS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate lift score for multiple classification problem Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the lift score

Return type ls (float, dict)

MCC(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate Matthews Correlation Coefficient Higher is better (Best = 1), Range = [-1, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes

- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Matthews correlation coefficient

Return type mcc (float, dict)

NPV(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate negative predictive value for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the negative predictive value

Return type npv (float, dict)

PS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate precision score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro" If None, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

  'micro': Calculate metrics globally by considering each element of the label indicator matrix as a label.

  'macro': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

  'weighted': Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).

Returns the precision score

Return type precision (float, dict)

RAS(*y_true=None*, *y_pred=None*, *average='macro'*, ***kwargs*)

Calculates the ROC-AUC score between y_true and y_score. Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes

- **y_pred** (*tuple, list, np.ndarray*) – A LIST OF PREDICTED SCORES (NOT LABELS)
- **average** (*str, None*) – {'macro', 'weighted'} or None, default="macro"

Returns The AUC score.

Return type float, dict

ROC(*y_true=None*, *y_pred=None*, *average='macro'*, ***kwargs*)
Calculates the ROC-AUC score between y_true and y_score. Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – A LIST OF PREDICTED SCORES (NOT LABELS)
- **average** (*str, None*) – {'macro', 'weighted'} or None, default="macro"

Returns The AUC score.

Return type float, dict

RS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate recall score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the recall score

Return type recall (float, dict)

SS(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate specificity score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the specificity score

Return type ss (float, dict)

```
SUPPORT = {'AS': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'BSL': {'best':
'0', 'range': '[0, 1]', 'type': 'min'}, 'CEL': {'best': '0', 'range': '[0,
+inf)', 'type': 'min'}, 'CKS': {'best': '1', 'range': '[-1, +1]', 'type':
'max'}, 'F1S': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'F2S': {'best':
'1', 'range': '[0, 1]', 'type': 'max'}, 'FBS': {'best': '1', 'range': '[0, 1]',
'type': 'max'}, 'GINI': {'best': '0', 'range': '[0, 1]', 'type': 'min'}, 'GMS':
{'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'HL': {'best': '0', 'range':
'[0, +inf)', 'type': 'min'}, 'HS': {'best': '1', 'range': '[0, 1]', 'type':
'max'}, 'JSI': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'KLDL': {'best':
'0', 'range': '[0, +inf)', 'type': 'min'}, 'LS': {'best': 'no best', 'range':
'[0, +inf)', 'type': 'max'}, 'MCC': {'best': '1', 'range': '[-1, +1]', 'type':
'max'}, 'NPV': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'PS': {'best':
'1', 'range': '[0, 1]', 'type': 'max'}, 'ROC-AUC': {'best': '1', 'range': '[0,
1]', 'type': 'max'}, 'RS': {'best': '1', 'range': '[0, 1]', 'type': 'max'},
'SS': {'best': '1', 'range': '[0, 1]', 'type': 'max'}}
```

accuracy_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate accuracy score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the accuracy score

Return type accuracy (float, dict)

brier_score_loss(*y_true=None*, *y_pred=None*, ***kwargs*)

Calculates the Brier Score Loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of labels (or predicted scores in case of multi-class)

Returns The Brier Score Loss

Return type float, dict

cohen_kappa_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate Cohen Kappa score for multiple classification problem Higher is better (Best = +1), Range = [-1, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Cohen Kappa score

Return type cks (float, dict)

confusion_matrix(*y_true=None*, *y_pred=None*, *labels=None*, *normalize=None*, ***kwargs*)
Generate confusion matrix and useful information

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **normalize** (*'true', 'pred', 'all', None*) – Normalizes confusion matrix over the true (rows), predicted (columns) conditions or all the population.

Returns a 2-dimensional list of pairwise counts imap (dict): a map between label and index of confusion matrix imap_count (dict): a map between label and number of true label in y_true

Return type matrix (np.ndarray)

crossentropy_loss(*y_true=None*, *y_pred=None*, ***kwargs*)
Calculates the Cross-Entropy loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – A LIST OF PREDICTED SCORES (NOT LABELS)

Returns The Cross-Entropy loss

Return type float

f1_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate f1 score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the f1 score

Return type f1 (float, dict)

f2_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate f2 score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the f2 score

Return type f2 (float, dict)

fbeta_score(*y_true=None*, *y_pred=None*, *beta=1.0*, *labels=None*, *average='macro'*, ***kwargs*)
The beta parameter determines the weight of recall in the combined score. beta < 1 lends more weight to precision, while beta > 1 favors recall (beta -> 0 considers only precision, beta -> +inf only recall). Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **beta** (*float*) – the weight of recall in the combined score, default = 1.0
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the fbeta score

Return type fbeta (float, dict)

g_mean_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Calculates the G-mean (Geometric mean) score between y_true and y_pred. Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns The G-mean score.

Return type float, dict

get_processed_data(*y_true=None*, *y_pred=None*)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction values

Returns y_true used in evaluation process. y_pred_final: y_pred used in evaluation process one_dim: is y_true has 1 dimensions or not

Return type y_true_final

get_processed_data2(*y_true=None*, *y_pred=None*)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values
- **y_pred** (*tuple, list, np.ndarray*) – The prediction scores

Returns y_true used in evaluation process. y_pred_final: y_pred used in evaluation process one_dim: is y_true has 1 dimensions or not

Return type y_true_final

static get_support(*name=None*, *verbose=True*)

gini_index(*y_true=None*, *y_pred=None*, ***kwargs*)

Calculates the Gini index between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes

Returns The Gini index

Return type float, dict

hamming_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate hamming score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the hamming score

Return type hl (float, dict)

hinge_loss(*y_true=None*, *y_pred=None*, ***kwargs*)

Calculates the Hinge loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of labels (or predicted scores in case of multi-class)

Returns The Hinge loss

Return type float

jaccard_similarity_coefficient(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)

Generate Jaccard similarity index for multiple classification problem Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Jaccard similarity index

Return type jsi (float, dict)

jaccard_similarity_index(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate Jaccard similarity index for multiple classification problem Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Jaccard similarity index

Return type jsi (float, dict)

kullback_leibler_divergence_loss(*y_true=None*, *y_pred=None*, ***kwargs*)
Calculates the Kullback-Leibler divergence loss between y_true and y_pred. Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of labels (or predicted scores in case of multi-class)

Returns The Kullback-Leibler divergence loss

Return type float

lift_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate lift score for multiple classification problem Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the lift score

Return type ls (float, dict)

matthews_correlation_coefficient(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate Matthews Correlation Coefficient Higher is better (Best = 1), Range = [-1, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.

- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the Matthews correlation coefficient

Return type mcc (float, dict)

negative_predictive_value(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate negative predictive value for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the negative predictive value

Return type npv (float, dict)

precision_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate precision score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro" If None, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

  'micro': Calculate metrics globally by considering each element of the label indicator matrix as a label.

  'macro': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

  'weighted': Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).

Returns the precision score

Return type precision (float, dict)

recall_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate recall score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the recall score

Return type recall (float, dict)

roc_auc_score(*y_true=None*, *y_pred=None*, *average='macro'*, ***kwargs*)
Calculates the ROC-AUC score between y_true and y_score. Higher is better (Best = +1), Range = [0, +1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – A LIST OF PREDICTED SCORES (NOT LABELS)
- **average** (*str, None*) – {'macro', 'weighted'} or None, default="macro"

Returns The AUC score.

Return type float, dict

specificity_score(*y_true=None*, *y_pred=None*, *labels=None*, *average='macro'*, ***kwargs*)
Generate specificity score for multiple classification problem Higher is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*tuple, list, np.ndarray*) – a list of integers or strings for known classes
- **y_pred** (*tuple, list, np.ndarray*) – a list of integers or strings for y_pred classes
- **labels** (*tuple, list, np.ndarray*) – List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **average** (*str, None*) – {'micro', 'macro', 'weighted'} or None, default="macro"

Returns the specificity score

Return type ss (float, dict)

6.5 permetrics.clustering module

class permetrics.clustering.**ClusteringMetric**(*y_true=None*, *y_pred=None*, *X=None*, *force_finite=True*, *finite_value=None*, ***kwargs*)

Bases: *permetrics.evaluator.Evaluator*

Defines a ClusteringMetric class that hold all internal and external metrics for clustering problems

- An extension of scikit-learn metrics section, with the addition of many more internal metrics.
- https://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation

Parameters

- **y_true** (*tuple, list, np.ndarray, default = None*) – The ground truth values. This is for calculating external metrics
- **y_pred** (*tuple, list, np.ndarray, default = None*) – The prediction values. This is for both calculating internal and external metrics
- **X** (*tuple, list, np.ndarray, default = None*) – The features of datasets. This is for calculating internal metrics
- **force_finite** (*bool, default = True*) – When result is not finite, it can be NaN or Inf. Their result will be replaced by *finite_value*

- **finite_value** (*float, default = None*) – The value that used to replace the infinite value or NaN value.

ARS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Adjusted rand score between two clusterings. Bigger is better (Best = 1), Range = [-1, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Adjusted rand score

Return type result (float)

BHI(*X=None*, *y_pred=None*, ***kwargs*)

The Ball-Hall Index (1995) is the mean of the mean dispersion across all clusters. The **largest difference** between successive clustering levels indicates the optimal number of clusters. Smaller is better (Best = 0), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The Ball-Hall index

Return type result (float)

BI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwarg*)

Computes the Beale Index Smaller is better (Best=0), Range = [0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Beale Index

Return type result (float)

BRI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)

Computes the Banfeld-Raftery Index. Smaller is better (No best value), Range=(-inf, inf) This index is the weighted sum of the logarithms of the traces of the variance covariance matrix of each cluster

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Banfeld-Raftery Index

Return type result (float)

CDS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Czekanowski-Dice score between two clusterings. It is the harmonic mean of the precision and recall coefficients. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Czekanowski-Dice score

Return type result (float)

CHI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Compute the Calinski and Harabasz (1974) index. It is also known as the Variance Ratio Criterion. The score is defined as ratio between the within-cluster dispersion and the between-cluster dispersion. Bigger is better (No best value), Range=[0, inf)

- This metric in scikit-learn library is wrong in calculate the intra_disp variable (WGSS)
- https://github.com/scikit-learn/scikit-learn/blob/7f9bad99d/sklearn/metrics/cluster/_unsupervised.py#L351C1-L351C1

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The resulting Calinski-Harabasz index.

Return type result (float)

CS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the completeness score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the ratio of samples that are correctly assigned to the same cluster to the total number of samples in the data.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The completeness score.

Return type result (float)

DBCVI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=1.0*, ***kwarg*)

Computes the Density-based Clustering Validation Index Smaller is better (Best=0), Range = [0, 1]

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Density-based Clustering Validation Index

Return type result (float)

DBI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)
Computes the Davies-Bouldin index Smaller is better (Best = 0), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Davies-Bouldin index

Return type result (float)

DHI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)
Computes the Duda Index or Duda-Hart index Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Duda-Hart index

Return type result (float)

DI(*X=None*, *y_pred=None*, *use_modified=True*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Computes the Dunn Index Bigger is better (No best value), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **use_modified** (*bool*) – The modified version we proposed to speed up the computational time for this metric, default=True
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Dunn Index

Return type result (float)

DRI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Computes the Det-Ratio index Bigger is better (No best value), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Det-Ratio index

Return type result (float)

ES(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Entropy score Smaller is better (Best = 0), Range = [0, +inf)

Entropy is a metric used to evaluate the quality of clustering results, particularly when the ground truth labels of the data points are known. It measures the amount of uncertainty or disorder within the clusters produced by a clustering algorithm.

Here's how the Entropy score is calculated:

1) For each cluster, compute the class distribution by counting the occurrences of each class label within the cluster.
2) Normalize the class distribution by dividing the count of each class label by the total number of data points in the cluster.
3) Compute the entropy for each cluster using the normalized class distribution.
4) Weight the entropy of each cluster by its relative size (proportion of data points in the whole dataset).
5) Sum up the weighted entropies of all clusters.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Entropy score

Return type result (float)

FMS(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Computes the Fowlkes-Mallows score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Fowlkes-Mallows score

Return type result (float)

FmS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the F-Measure score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It is the harmonic mean of the precision and recall coefficients, given by the formula F = 2PR / (P + R). It provides a single score that summarizes both precision and recall. The Fa-measure is a weighted version

of the F-measure that allows for a trade-off between precision and recall. It is defined as Fa = (1 + a)PR / (aP + R), where a is a parameter that determines the relative importance of precision and recall.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The F-Measure score

Return type result (float)

GAS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Gamma Score between two clustering solutions. Bigger is better (Best = 1), Range = [-1, 1]

Ref: Cluster Validation for Mixed-Type Data (Rabea Aschenbruck and Gero Szepannek)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Gamma Score

Return type result (float)

GPS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Gplus Score between two clustering solutions. Smaller is better (Best = 0), Range = [0, 1]

Ref: Cluster Validation for Mixed-Type Data (Rabea Aschenbruck and Gero Szepannek)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Gplus Score

Return type result (float)

HGS(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Computes the Hubert Gamma score between two clusterings. Bigger is better (Best = 1), Range=[-1, +1]

The Hubert Gamma index ranges from -1 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared, a value of 0 indicates no association between the partitions, and a value of -1 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Hubert Gamma score

Return type result (float)

HI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwarg*)

Computes the Hartigan index for a clustering solution. Smaller is better (best=0), Range = [0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Hartigan index

Return type result (float)

HS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Homogeneity Score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the extent to which each cluster contains only data points that belong to a single class or category. In other words, homogeneity assesses whether all the data points in a cluster are members of the same true class or label. A higher homogeneity score indicates better clustering results, where each cluster corresponds well to a single ground truth class.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Homogeneity Score

Return type result (float)

JS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Jaccard score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It ranges from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

The Jaccard score is similar to the Czekanowski-Dice score, but it is less sensitive to differences in cluster size. However, like the Czekanowski-Dice score, it may not be sensitive to certain types of differences between partitions. Therefore, it is often used in conjunction with other external indices to get a more complete picture of the similarity between partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Jaccard score

Return type result (float)

KDI(*X=None*, *y_pred=None*, *use_normalized=True*, ***kwargs*)

Computes the Ksq-DetW Index Bigger is better (No best value), Range=(-inf, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **use_normalized** (*bool*) – We normalize the scatter matrix before calculate the Det to reduce the value, default=True

Returns The Ksq-DetW Index

Return type result (float)

KS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Kulczynski score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It is the arithmetic mean of the precision and recall coefficients, which means that it takes into account both precision and recall. The Kulczynski index ranges from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Kulczynski score

Return type result (float)

LDRI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*, ***kwargs*)

Computes the Log Det Ratio Index Bigger is better (No best value), Range=(-inf, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Log Det Ratio Index

Return type result (float)

LSRI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*, ***kwargs*)

Computes the Log SS Ratio Index Bigger is better (No best value), Range=(-inf, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Log SS Ratio Index

Return type result (float)

MIS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Mutual Information score between two clusterings. Bigger is better (No best value), Range = [0, +inf)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Mutual Information score

Return type result (float)

MNS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Mc Nemar score between two clusterings. Bigger is better (No best value), Range=(-inf, +inf)

It is an adaptation of the non-parametric McNemar test for the comparison of frequencies between two paired samples. The McNemar index ranges from -inf to inf, where a bigger value indicates perfect agreement between the two partitions being compared

Under the null hypothesis that the discordances between the partitions P1 and P2 are random, the McNemar index follows approximately a normal distribution. The McNemar index can be transformed into a chi-squared distance, which follows a chi-squared distribution with 1 degree of freedom

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Mc Nemar score

Return type result (float)

MSEI(*X=None*, *y_pred=None*, ***kwarg*)

Computes the Mean Squared Error Index Smaller is better (Best = 0), Range = [0, +inf)

MSEI measures the mean of squared distances between each data point and its corresponding centroid or cluster center.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The Mean Squared Error Index

Return type result (float)

NMIS(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Computes the normalized mutual information between two clusterings. It is a variation of the mutual information score that normalizes the result to take values between 0 and 1. It is defined as the mutual information divided by the average entropy of the true and predicted clusterings. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The normalized mutual information score.

Return type result (float)

PhS(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*, ***kwargs*)

Computes the Phi score between two clusterings. Bigger is better (No best value), Range = (-inf, +inf)

It is a classical measure of the correlation between two dichotomous variables, and it can be used to measure the similarity between two partitions. The Phi index ranges from -inf to +inf, where a bigger value indicates

perfect agreement between the two partitions being compared, a value of 0 indicates no association between the partitions, and a smaller value indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Phi score

Return type result (float)

PrS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Precision score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]. It is different than precision score in classification metrics

It measures the proportion of points that are correctly grouped together in P2, given that they are grouped together in P1. It is calculated as the ratio of yy (the number of points that are correctly grouped together in both P1 and P2) to the sum of yy and ny (the number of points that are grouped together in P2 but not in P1). The formula for P is P = yy / (yy + ny).

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Precision score

Return type result (float)

PuS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Purity score Bigger is better (Best = 1), Range = [0, 1]

Purity is a metric used to evaluate the quality of clustering results, particularly in situations where the ground truth labels of the data points are known. It measures the extent to which the clusters produced by a clustering algorithm match the true class labels of the data.

Here's how Purity is calculated:

1) For each cluster, find the majority class label among the data points in that cluster.
2) Sum up the sizes of the clusters that belong to the majority class label.
3) Divide the sum by the total number of data points.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Purity score

Return type result (float)

RRS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Russel-Rao score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the proportion of concordances between the two partitions by computing the proportion of pairs of samples that are in the same cluster in both partitions. The Russel-Rao index ranges from 0 to 1, where

a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Russel-Rao score

Return type result (float)

RSI(*X=None*, *y_pred=None*, ***kwarg*)

Computes the R-squared index Bigger is better (Best=1), Range = (-inf, 1]

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The R-squared index

Return type result (float)

RTS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Rogers-Tanimoto score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the similarity between two partitions by computing the proportion of pairs of samples that are either in the same cluster in both partitions or in different clusters in both partitions, with an adjustment for the number of pairs of samples that are in different clusters in one partition but in the same cluster in the other partition. The Rogers-Tanimoto index ranges from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Rogers-Tanimoto score

Return type result (float)

RaS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Rand score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The rand score.

Return type result (float)

ReS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Recall score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the proportion of points that are correctly grouped together in P2, given that they are grouped together in P1. It is calculated as the ratio of yy to the sum of yy and yn (the number of points that are grouped together in P1 but not in P2). The formula for R is R = yy / (yy + yn).

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Recall score

Return type result (float)

SI(*X=None*, *y_pred=None*, *multi_output=False*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)
Computes the Silhouette Index Bigger is better (Best = 1), Range = [-1, +1]

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **multi_output** (*bool*) – Returned scores for each cluster, default=False
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Silhouette Index

Return type result (float)

SS1S(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Sokal-Sneath 1 score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the similarity between two partitions by computing the proportion of pairs of samples that are in the same cluster in both partitions, with an adjustment for the number of pairs of samples that are in different clusters in one partition but in the same cluster in the other partition. The Sokal-Sneath indices range from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Sokal-Sneath 1 score

Return type result (float)

SS2S(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Sokal-Sneath 2 score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the similarity between two partitions by computing the proportion of pairs of samples that are in the same cluster in both partitions, with an adjustment for the number of pairs of samples that are in different clusters in one partition but in the same cluster in the other partition. The Sokal-Sneath indices range from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Sokal-Sneath 2 score

Return type result (float)

SSEI(*X=None*, *y_pred=None*, ***kwarg*)

Computes the Sum of Squared Error Index Smaller is better (Best = 0), Range = [0, +inf)

SSEI measures the sum of squared distances between each data point and its corresponding centroid or cluster center. It quantifies the compactness of the clusters. Here's how you can calculate the SSE in a clustering problem:

1) Assign each data point to its nearest centroid or cluster center based on some distance metric (e.g., Euclidean distance).
2) For each data point, calculate the squared Euclidean distance between the data point and its assigned centroid.
3) Sum up the squared distances for all data points to obtain the SSE.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The Sum of Squared Error Index

Return type result (float)

```
SUPPORT = {'ARS': {'best': '1', 'range': '[-1, 1]', 'type': 'max'}, 'BHI':
{'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'BI': {'best': '0', 'range':
'[0, +inf)', 'type': 'min'}, 'BRI': {'best': 'no best', 'range': '(-inf, +inf)',
'type': 'min'}, 'CDS': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'CHI':
{'best': 'no best', 'range': '[0, +inf)', 'type': 'max'}, 'CS': {'best': '1',
'range': '[0, 1]', 'type': 'max'}, 'DBCVI': {'best': '0', 'range': '[0, 1]',
'type': 'min'}, 'DBI': {'best': '0', 'range': '[0, +inf)', 'type': 'min'},
'DHI': {'best': '0', 'range': '[0, +inf)', 'type': 'min'}, 'DI': {'best': 'no
best', 'range': '[0, +inf)', 'type': 'max'}, 'DRI': {'best': 'no best', 'range':
'[0, +inf)', 'type': 'max'}, 'ES': {'best': '0', 'range': '[0, +inf)', 'type':
'min'}, 'FMS': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'FmS': {'best':
'1', 'range': '[0, 1]', 'type': 'max'}, 'GAS': {'best': '1', 'range': '[-1, 1]',
'type': 'max'}, 'GPS': {'best': '0', 'range': '[0, 1]', 'type': 'min'}, 'HGS':
{'best': '1', 'range': '[-1, 1]', 'type': 'max'}, 'HI': {'best': '0', 'range':
'[0, +inf)', 'type': 'min'}, 'HS': {'best': '1', 'range': '[0, 1]', 'type':
'max'}, 'JS': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'KDI': {'best':
'no best', 'range': '(-inf, +inf)', 'type': 'max'}, 'KS': {'best': '1', 'range':
'[0, 1]', 'type': 'max'}, 'LDRI': {'best': 'no best', 'range': '(-inf, +inf)',
'type': 'max'}, 'LSRI': {'best': 'no best', 'range': '(-inf, +inf)', 'type':
'max'}, 'MIS': {'best': 'no best', 'range': '[0, +inf)', 'type': 'max'}, 'MNS':
{'best': 'no best', 'range': '(-inf, +inf)', 'type': 'max'}, 'MSEI': {'best':
'0', 'range': '[0, +inf)', 'type': 'min'}, 'NMIS': {'best': '1', 'range': '[0,
1]', 'type': 'max'}, 'PhS': {'best': 'no best', 'range': '(-inf, +inf)', 'type':
'max'}, 'PrS': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'PuS': {'best':
'1', 'range': '[0, 1]', 'type': 'max'}, 'RRS': {'best': '1', 'range': '[0, 1]',
'type': 'max'}, 'RSI': {'best': '1', 'range': '(-inf, +1]', 'type': 'max'},
'RTS': {'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'RaS': {'best': '1',
'range': '[0, 1]', 'type': 'max'}, 'ReS': {'best': '1', 'range': '[0, 1]',
'type': 'max'}, 'SI': {'best': '1', 'range': '[-1, +1]', 'type': 'max'}, 'SS1S':
{'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'SS2S': {'best': '1', 'range':
'[0, 1]', 'type': 'max'}, 'SSEI': {'best': '0', 'range': '[0, +inf)', 'type':
'min'}, 'TS': {'best': 'no best', 'range': '(-inf, +inf)', 'type': 'max'}, 'VMS':
{'best': '1', 'range': '[0, 1]', 'type': 'max'}, 'XBI': {'best': '0', 'range':
'[0, +inf)', 'type': 'min'}}
```

TS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Tau Score between two clustering solutions. Bigger is better (No best value), Range = (-inf, +inf)

Ref: Cluster Validation for Mixed-Type Data (Rabea Aschenbruck and Gero Szepannek)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Tau Score

Return type result (float)

VMS(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the V measure score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It is a combination of two other metrics: homogeneity and completeness. Homogeneity measures whether all the data points in a given cluster belong to the same class. Completeness measures whether all the data points of a certain class are assigned to the same cluster. The V-measure combines these two metrics into a single score.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The V measure score

Return type result (float)

XBI(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)

Computes the Xie-Beni index. Smaller is better (Best = 0), Range=[0, +inf)

The Xie-Beni index is an index of fuzzy clustering, but it is also applicable to crisp clustering. The numerator is the mean of the squared distances of all of the points with respect to their barycenter of the cluster they belong to. The denominator is the minimal squared distances between the points in the clusters. The **minimum** value indicates the best number of clusters.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Xie-Beni index

Return type result (float)

adjusted_rand_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Adjusted rand score between two clusterings. Bigger is better (Best = 1), Range = [-1, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Adjusted rand score

Return type result (float)

ball_hall_index(*X=None*, *y_pred=None*, ***kwargs*)

The Ball-Hall Index (1995) is the mean of the mean dispersion across all clusters. The **largest difference** between successive clustering levels indicates the optimal number of clusters. Smaller is better (Best = 0), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The Ball-Hall index

Return type result (float)

banfeld_raftery_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)

Computes the Banfeld-Raftery Index. Smaller is better (No best value), Range=(-inf, inf) This index is the weighted sum of the logarithms of the traces of the variance covariance matrix of each cluster

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Banfeld-Raftery Index

Return type result (float)

beale_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwarg*)

Computes the Beale Index Smaller is better (Best=0), Range = [0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Beale Index

Return type result (float)

calinski_harabasz_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Compute the Calinski and Harabasz (1974) index. It is also known as the Variance Ratio Criterion. The score is defined as ratio between the within-cluster dispersion and the between-cluster dispersion. Bigger is better (No best value), Range=[0, inf)

- This metric in scikit-learn library is wrong in calculate the intra_disp variable (WGSS)
- https://github.com/scikit-learn/scikit-learn/blob/7f9bad99d/sklearn/metrics/cluster/_unsupervised.py#L351C1-L351C1

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The resulting Calinski-Harabasz index.

Return type result (float)

check_X(*X*)

completeness_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the completeness score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the ratio of samples that are correctly assigned to the same cluster to the total number of samples in the data.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The completeness score.

Return type result (float)

czekanowski_dice_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Czekanowski-Dice score between two clusterings. It is the harmonic mean of the precision and recall coefficients. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Czekanowski-Dice score

Return type result (float)

davies_bouldin_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)

Computes the Davies-Bouldin index Smaller is better (Best = 0), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Davies-Bouldin index

Return type result (float)

density_based_clustering_validation_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=1.0*, ***kwarg*)

Computes the Density-based Clustering Validation Index Smaller is better (Best=0), Range = [0, 1]

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Density-based Clustering Validation Index

Return type result (float)

det_ratio_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Computes the Det-Ratio index Bigger is better (No best value), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.

- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Det-Ratio index

Return type result (float)

duda_hart_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)
Computes the Duda Index or Duda-Hart index Smaller is better (Best = 0), Range = [0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Duda-Hart index

Return type result (float)

dunn_index(*X=None*, *y_pred=None*, *use_modified=True*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)
Computes the Dunn Index Bigger is better (No best value), Range=[0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **use_modified** (*bool*) – The modified version we proposed to speed up the computational time for this metric, default=True
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Dunn Index

Return type result (float)

entropy_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Entropy score Smaller is better (Best = 0), Range = [0, +inf)

Entropy is a metric used to evaluate the quality of clustering results, particularly when the ground truth labels of the data points are known. It measures the amount of uncertainty or disorder within the clusters produced by a clustering algorithm.

Here's how the Entropy score is calculated:

1) For each cluster, compute the class distribution by counting the occurrences of each class label within the cluster.
2) Normalize the class distribution by dividing the count of each class label by the total number of data points in the cluster.
3) Compute the entropy for each cluster using the normalized class distribution.
4) Weight the entropy of each cluster by its relative size (proportion of data points in the whole dataset).

5) Sum up the weighted entropies of all clusters.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Entropy score

Return type result (float)

f_measure_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the F-Measure score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It is the harmonic mean of the precision and recall coefficients, given by the formula F = 2PR / (P + R). It provides a single score that summarizes both precision and recall. The Fa-measure is a weighted version of the F-measure that allows for a trade-off between precision and recall. It is defined as Fa = (1 + a)PR / (aP + R), where a is a parameter that determines the relative importance of precision and recall.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The F-Measure score

Return type result (float)

fowlkes_mallows_score(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Computes the Fowlkes-Mallows score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Fowlkes-Mallows score

Return type result (float)

gamma_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Gamma Score between two clustering solutions. Bigger is better (Best = 1), Range = [-1, 1]

Ref: Cluster Validation for Mixed-Type Data (Rabea Aschenbruck and Gero Szepannek)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Gamma Score

Return type result (float)

get_processed_external_data(*y_true=None*, *y_pred=None*, *force_finite=None*, *finite_value=None*)

Parameters

- **y_true** (*tuple, list, np.ndarray*) – The ground truth values

- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **force_finite** (*bool*) – Force the result as finite number
- **finite_value** (*float*) – The finite number

Returns y_true used in evaluation process. y_pred_final: y_pred used in evaluation process le: label encoder object force_finite: Force the result as finite number finite_value: The finite number

Return type y_true_final

get_processed_internal_data(*y_pred=None*, *force_finite=None*, *finite_value=None*)

Parameters

- **y_pred** (*tuple, list, np.ndarray*) – The prediction values
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns y_pred used in evaluation process le: label encoder object force_finite finite_value

Return type y_pred_final

static get_support(*name=None*, *verbose=True*)

gplus_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Gplus Score between two clustering solutions. Smaller is better (Best = 0), Range = [0, 1]

Ref: Cluster Validation for Mixed-Type Data (Rabea Aschenbruck and Gero Szepannek)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Gplus Score

Return type result (float)

hartigan_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwarg*)

Computes the Hartigan index for a clustering solution. Smaller is better (best=0), Range = [0, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Hartigan index

Return type result (float)

homogeneity_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Homogeneity Score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the extent to which each cluster contains only data points that belong to a single class or category. In other words, homogeneity assesses whether all the data points in a cluster are members of the same

true class or label. A higher homogeneity score indicates better clustering results, where each cluster corresponds well to a single ground truth class.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Homogeneity Score

Return type result (float)

hubert_gamma_score(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)
Computes the Hubert Gamma score between two clusterings. Bigger is better (Best = 1), Range=[-1, +1]

The Hubert Gamma index ranges from -1 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared, a value of 0 indicates no association between the partitions, and a value of -1 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Hubert Gamma score

Return type result (float)

jaccard_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Jaccard score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It ranges from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

The Jaccard score is similar to the Czekanowski-Dice score, but it is less sensitive to differences in cluster size. However, like the Czekanowski-Dice score, it may not be sensitive to certain types of differences between partitions. Therefore, it is often used in conjunction with other external indices to get a more complete picture of the similarity between partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Jaccard score

Return type result (float)

ksq_detw_index(*X=None*, *y_pred=None*, *use_normalized=True*, ***kwargs*)
Computes the Ksq-DetW Index Bigger is better (No best value), Range=(-inf, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **use_normalized** (*bool*) – We normalize the scatter matrix before calculate the Det to reduce the value, default=True

Returns The Ksq-DetW Index

Return type result (float)

kulczynski_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Kulczynski score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It is the arithmetic mean of the precision and recall coefficients, which means that it takes into account both precision and recall. The Kulczynski index ranges from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Kulczynski score

Return type result (float)

log_det_ratio_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*, ***kwargs*)

Computes the Log Det Ratio Index Bigger is better (No best value), Range=(-inf, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Log Det Ratio Index

Return type result (float)

log_ss_ratio_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*, ***kwargs*)

Computes the Log SS Ratio Index Bigger is better (No best value), Range=(-inf, +inf)

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Log SS Ratio Index

Return type result (float)

mc_nemar_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Mc Nemar score between two clusterings. Bigger is better (No best value), Range=(-inf, +inf)

It is an adaptation of the non-parametric McNemar test for the comparison of frequencies between two paired samples. The McNemar index ranges from -inf to inf, where a bigger value indicates perfect agreement between the two partitions being compared

Under the null hypothesis that the discordances between the partitions P1 and P2 are random, the McNemar index follows approximately a normal distribution. The McNemar index can be transformed into a chi-squared distance, which follows a chi-squared distribution with 1 degree of freedom

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Mc Nemar score

Return type result (float)

mean_squared_error_index(*X=None*, *y_pred=None*, ***kwarg*)

Computes the Mean Squared Error Index Smaller is better (Best = 0), Range = [0, +inf)

MSEI measures the mean of squared distances between each data point and its corresponding centroid or cluster center.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The Mean Squared Error Index

Return type result (float)

mutual_info_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Mutual Information score between two clusterings. Bigger is better (No best value), Range = [0, +inf)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Mutual Information score

Return type result (float)

normalized_mutual_info_score(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=0.0*, ***kwargs*)

Computes the normalized mutual information between two clusterings. It is a variation of the mutual information score that normalizes the result to take values between 0 and 1. It is defined as the mutual information divided by the average entropy of the true and predicted clusterings. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The normalized mutual information score.

Return type result (float)

phi_score(*y_true=None*, *y_pred=None*, *force_finite=True*, *finite_value=- 10000000000.0*, ***kwargs*)
Computes the Phi score between two clusterings. Bigger is better (No best value), Range = (-inf, +inf)

It is a classical measure of the correlation between two dichotomous variables, and it can be used to measure the similarity between two partitions. The Phi index ranges from -inf to +inf, where a bigger value indicates perfect agreement between the two partitions being compared, a value of 0 indicates no association between the partitions, and a smaller value indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Phi score

Return type result (float)

precision_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Precision score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]. It is different than precision score in classification metrics

It measures the proportion of points that are correctly grouped together in P2, given that they are grouped together in P1. It is calculated as the ratio of yy (the number of points that are correctly grouped together in both P1 and P2) to the sum of yy and ny (the number of points that are grouped together in P2 but not in P1). The formula for P is P = yy / (yy + ny).

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Precision score

Return type result (float)

purity_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Purity score Bigger is better (Best = 1), Range = [0, 1]

Purity is a metric used to evaluate the quality of clustering results, particularly in situations where the ground truth labels of the data points are known. It measures the extent to which the clusters produced by a clustering algorithm match the true class labels of the data.

Here's how Purity is calculated:

1) For each cluster, find the majority class label among the data points in that cluster.
2) Sum up the sizes of the clusters that belong to the majority class label.
3) Divide the sum by the total number of data points.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Purity score

Return type result (float)

r_squared_index(*X=None*, *y_pred=None*, ***kwarg*)
Computes the R-squared index Bigger is better (Best=1), Range = (-inf, 1]

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The R-squared index

Return type result (float)

rand_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Rand score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The rand score.

Return type result (float)

recall_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Recall score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the proportion of points that are correctly grouped together in P2, given that they are grouped together in P1. It is calculated as the ratio of yy to the sum of yy and yn (the number of points that are grouped together in P1 but not in P2). The formula for R is R = yy / (yy + yn).

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Recall score

Return type result (float)

rogers_tanimoto_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Rogers-Tanimoto score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the similarity between two partitions by computing the proportion of pairs of samples that are either in the same cluster in both partitions or in different clusters in both partitions, with an adjustment for the number of pairs of samples that are in different clusters in one partition but in the same cluster in the other partition. The Rogers-Tanimoto index ranges from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Rogers-Tanimoto score

Return type result (float)

russel_rao_score(*y_true=None*, *y_pred=None*, ***kwargs*)
Computes the Russel-Rao score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the proportion of concordances between the two partitions by computing the proportion of pairs of samples that are in the same cluster in both partitions. The Russel-Rao index ranges from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Russel-Rao score

Return type result (float)

silhouette_index(*X=None*, *y_pred=None*, *multi_output=False*, *force_finite=True*, *finite_value=- 1.0*, ***kwargs*)

Computes the Silhouette Index Bigger is better (Best = 1), Range = [-1, +1]

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **multi_output** (*bool*) – Returned scores for each cluster, default=False
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Silhouette Index

Return type result (float)

sokal_sneath1_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Sokal-Sneath 1 score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the similarity between two partitions by computing the proportion of pairs of samples that are in the same cluster in both partitions, with an adjustment for the number of pairs of samples that are in different clusters in one partition but in the same cluster in the other partition. The Sokal-Sneath indices range from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Sokal-Sneath 1 score

Return type result (float)

sokal_sneath2_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Sokal-Sneath 2 score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It measures the similarity between two partitions by computing the proportion of pairs of samples that are in the same cluster in both partitions, with an adjustment for the number of pairs of samples that are in different clusters in one partition but in the same cluster in the other partition. The Sokal-Sneath indices range from 0 to 1, where a value of 1 indicates perfect agreement between the two partitions being compared. A value of 0 indicates complete disagreement between the two partitions.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Sokal-Sneath 2 score

Return type result (float)

sum_squared_error_index(*X=None*, *y_pred=None*, ***kwarg*)

Computes the Sum of Squared Error Index Smaller is better (Best = 0), Range = [0, +inf)

SSEI measures the sum of squared distances between each data point and its corresponding centroid or cluster center. It quantifies the compactness of the clusters. Here's how you can calculate the SSE in a clustering problem:

1) Assign each data point to its nearest centroid or cluster center based on some distance metric (e.g., Euclidean distance).
2) For each data point, calculate the squared Euclidean distance between the data point and its assigned centroid.
3) Sum up the squared distances for all data points to obtain the SSE.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.

Returns The Sum of Squared Error Index

Return type result (float)

tau_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the Tau Score between two clustering solutions. Bigger is better (No best value), Range = (-inf, +inf)

Ref: Cluster Validation for Mixed-Type Data (Rabea Aschenbruck and Gero Szepannek)

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The Tau Score

Return type result (float)

v_measure_score(*y_true=None*, *y_pred=None*, ***kwargs*)

Computes the V measure score between two clusterings. Bigger is better (Best = 1), Range = [0, 1]

It is a combination of two other metrics: homogeneity and completeness. Homogeneity measures whether all the data points in a given cluster belong to the same class. Completeness measures whether all the data points of a certain class are assigned to the same cluster. The V-measure combines these two metrics into a single score.

Parameters

- **y_true** (*array-like*) – The true labels for each sample.
- **y_pred** (*array-like*) – The predicted cluster labels for each sample.

Returns The V measure score

Return type result (float)

xie_beni_index(*X=None*, *y_pred=None*, *force_finite=True*, *finite_value=10000000000.0*, ***kwargs*)
Computes the Xie-Beni index. Smaller is better (Best = 0), Range=[0, +inf)

The Xie-Beni index is an index of fuzzy clustering, but it is also applicable to crisp clustering. The numerator is the mean of the squared distances of all of the points with respect to their barycenter of the cluster they belong to. The denominator is the minimal squared distances between the points in the clusters. The **minimum** value indicates the best number of clusters.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – A list of *n_features*-dimensional data points. Each row corresponds to a single data point.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels for each sample.
- **force_finite** (*bool*) – Make result as finite number
- **finite_value** (*float*) – The value that used to replace the infinite value or NaN value.

Returns The Xie-Beni index

Return type result (float)

CHAPTER

SEVEN

CITATION REQUEST

Please include these citations if you plan to use this library:

```
@article{Thieu_PerMetrics_A_Framework_2024,
        author = {Thieu, Nguyen Van},
        doi = {10.21105/joss.06143},
        journal = {Journal of Open Source Software},
        month = mar,
        number = {95},
        pages = {6143},
        title = {{PerMetrics: A Framework of Performance Metrics for Machine Learning␣
↪Models}},
        url = {https://joss.theoj.org/papers/10.21105/joss.06143},
        volume = {9},
        year = {2024}
}

@article{van2023mealpy,
        title={MEALPY: An open-source library for latest meta-heuristic algorithms in␣
↪Python},
        author={Van Thieu, Nguyen and Mirjalili, Seyedali},
        journal={Journal of Systems Architecture},
        year={2023},
        publisher={Elsevier},
        doi={10.1016/j.sysarc.2023.102871}
}
```

If you have an open-ended or a research question, you can contact me via nguyenthieu2102@gmail.com

CHAPTER

EIGHT

ALL PERFORMANCE METRICS

The list of all available performance metrics in this library are as follows:

STT	Metric	Metric Fullname	Characteristics
1	EVS	Explained Variance Score	Bigger is better (Best = 1), Range=(-inf, 1.0]
2	ME	Max Error	Smaller is better (Best = 0), Range=[0, +inf)
3	MBE	Mean Bias Error	Best = 0, Range=(-inf, +inf)
4	MAE	Mean Absolute Error	Smaller is better (Best = 0), Range=[0, +inf)
5	MSE	Mean Squared Error	Smaller is better (Best = 0), Range=[0, +inf)
6	RMSE	Root Mean Squared Error	Smaller is better (Best = 0), Range=[0, +inf)
7	MSLE	Mean Squared Log Error	Smaller is better (Best = 0), Range=[0, +inf)
8	MedAE	Median Absolute Error	Smaller is better (Best = 0), Range=[0, +inf)
9	MRE / MRB	Mean Relative Error / Mean Relative Bias	Smaller is better (Best = 0), Range=[0, +inf)
10	MPE	Mean Percentage Error	Best = 0, Range=(-inf, +inf)
11	MAPE	Mean Absolute Percentage Error	Smaller is better (Best = 0), Range=[0, +inf)
12	SMAPE	Symmetric Mean Absolute Percentage Error	Smaller is better (Best = 0), Range=[0, 1]
13	MAAPE	Mean Arctangent Absolute Percentage Error	Smaller is better (Best = 0), Range=[0, +inf)
14	MASE	Mean Absolute Scaled Error	Smaller is better (Best = 0), Range=[0, +inf)
15	NSE	Nash-Sutcliffe Efficiency Coefficient	Bigger is better (Best = 1), Range=(-inf, 1]
16	NNSE	Normalized Nash-Sutcliffe Efficiency Coefficient	Bigger is better (Best = 1), Range=[0, 1]
17	WI	Willmott Index	Bigger is better (Best = 1), Range=[0, 1]
18	R / PCC	Pearson's Correlation Coefficient	Bigger is better (Best = 1), Range=[-1, 1]
19	AR / APCC	Absolute Pearson's Correlation Coefficient	Bigger is better (Best = 1), Range=[-1, 1]
20	RSQ/R2S	(Pearson's Correlation Index) ^ 2	Bigger is better (Best = 1), Range=[0, 1]
21	R2 / COD	Coefficient of Determination	Bigger is better (Best = 1), Range=(-inf, 1]
22	AR2 / ACOD	Adjusted Coefficient of Determination	Bigger is better (Best = 1), Range=(-inf, 1]
23	CI	Confidence Index	Bigger is better (Best = 1), Range=(-inf, 1]
24	DRV	Deviation of Runoff Volume	Smaller is better (Best = 1.0), Range=[1, +inf)
25	KGE	Kling-Gupta Efficiency	Bigger is better (Best = 1), Range=(-inf, 1]
26	GINI	Gini Coefficient	Smaller is better (Best = 0), Range=[0, +inf)
27	GINI_WIKI	Gini Coefficient on Wikipage	Smaller is better (Best = 0), Range=[0, +inf)
28	PCD	Prediction of Change in Direction	Bigger is better (Best = 1.0), Range=[0, 1]
29	CE	Cross Entropy	Range(-inf, 0], Can't give comment about this
30	KLD	Kullback Leibler Divergence	Best = 0, Range=(-inf, +inf)
31	JSD	Jensen Shannon Divergence	Smaller is better (Best = 0), Range=[0, +inf)
32	VAF	Variance Accounted For	Bigger is better (Best = 100%), Range=(-inf, 100%]
33	RAE	Relative Absolute Error	Smaller is better (Best = 0), Range=[0, +inf)
34	A10	A10 Index	Bigger is better (Best = 1), Range=[0, 1]
35	A20	A20 Index	Bigger is better (Best = 1), Range=[0, 1]

continues on next page

Table 1 – continued from previous page

STT	Metric	Metric Fullname	Characteristics
36	A30	A30 Index	Bigger is better (Best = 1), Range=[0, 1]
37	NRMSE	Normalized Root Mean Square Error	Smaller is better (Best = 0), Range=[0, +inf)
38	RSE	Residual Standard Error	Smaller is better (Best = 0), Range=[0, +inf)
39	RE / RB	Relative Error / Relative Bias	Best = 0, Range=(-inf, +inf)
40	AE	Absolute Error	Best = 0, Range=(-inf, +inf)
41	SE	Squared Error	Smaller is better (Best = 0), Range=[0, +inf)
42	SLE	Squared Log Error	Smaller is better (Best = 0), Range=[0, +inf)
43	COV	Covariance	Bigger is better (No best value), Range=(-inf, +inf)
44	COR	Correlation	Bigger is better (Best = 1), Range=[-1, +1]
45	EC	Efficiency Coefficient	Bigger is better (Best = 1), Range=(-inf, +1]
46	OI	Overall Index	Bigger is better (Best = 1), Range=(-inf, +1]
47	CRM	Coefficient of Residual Mass	Smaller is better (Best = 0), Range=(-inf, +inf)

STT	Metric	Metric Fullname	Characteristics
1	PS	Precision Score	Bigger is better (Best = 1), Range = [0, 1]
2	NPV	Negative Predictive Value	Bigger is better (Best = 1), Range = [0, 1]
3	RS	Recall Score	Bigger is better (Best = 1), Range = [0, 1]
4	AS	Accuracy Score	Bigger is better (Best = 1), Range = [0, 1]
5	F1S	F1 Score	Bigger is better (Best = 1), Range = [0, 1]
6	F2S	F2 Score	Bigger is better (Best = 1), Range = [0, 1]
7	FBS	F-Beta Score	Bigger is better (Best = 1), Range = [0, 1]
8	SS	Specificity Score	Bigger is better (Best = 1), Range = [0, 1]
9	MCC	Matthews Correlation Coefficient	Bigger is better (Best = 1), Range = [-1, +1]
10	HS	Hamming Score	Bigger is better (Best = 1), Range = [0, 1]
11	CKS	Cohen's kappa score	Bigger is better (Best = +1), Range = [-1, +1]
12	JSI	Jaccard Similarity Coefficient	Bigger is better (Best = +1), Range = [0, +1]
13	GMS	Geometric Mean Score	Bigger is better (Best = +1), Range = [0, +1]
14	ROC-AUC	ROC-AUC	Bigger is better (Best = +1), Range = [0, +1]
15	LS	Lift Score	Bigger is better (No best value), Range = [0, +inf)
16	GINI	GINI Index	Smaller is better (Best = 0), Range = [0, +1]
17	CEL	Cross Entropy Loss	Smaller is better (Best = 0), Range=[0, +inf)
18	HL	Hinge Loss	Smaller is better (Best = 0), Range=[0, +inf)
19	KLDL	Kullback Leibler Divergence Loss	Smaller is better (Best = 0), Range=[0, +inf)
20	BSL	Brier Score Loss	Smaller is better (Best = 0), Range=[0, +1]

STT	Metric	Metric Fullname	Characteristics
1	BHI	Ball Hall Index	Smaller is better (Best = 0), Range=[0, +inf)
2	XBI	Xie Beni Index	Smaller is better (Best = 0), Range=[0, +inf)
3	DBI	Davies Bouldin Index	Smaller is better (Best = 0), Range=[0, +inf)
4	BRI	Banfeld Raftery Index	Smaller is better (No best value), Range=(-inf, inf)
5	KDI	Ksq Detw Index	Smaller is better (No best value), Range=(-inf, +inf)
6	DRI	Det Ratio Index	Bigger is better (No best value), Range=[0, +inf)
7	DI	Dunn Index	Bigger is better (No best value), Range=[0, +inf)
8	CHI	Calinski Harabasz Index	Bigger is better (No best value), Range=[0, inf)
9	LDRI	Log Det Ratio Index	Bigger is better (No best value), Range=(-inf, +inf)
10	LSRI	Log SS Ratio Index	Bigger is better (No best value), Range=(-inf, +inf)
11	SI	Silhouette Index	Bigger is better (Best = 1), Range = [-1, +1]
12	SSEI	Sum of Squared Error Index	Smaller is better (Best = 0), Range = [0, +inf)

continues on next page

Table 2 – continued from previous page

STT	Metric	Metric Fullname	Characteristics
13	MSEI	Mean Squared Error Index	Smaller is better (Best = 0), Range = [0, +inf)
14	DHI	Duda-Hart Index	Smaller is better (Best = 0), Range = [0, +inf)
15	BI	Beale Index	Smaller is better (Best = 0), Range = [0, +inf)
16	RSI	R-squared Index	Bigger is better (Best=1), Range = (-inf, 1]
17	DBCVI	Density-based Clustering Validation Index	Bigger is better (Best=0), Range = [0, 1]
18	HI	Hartigan Index	Bigger is better (best=0), Range = [0, +inf)
19	MIS	Mutual Info Score	Bigger is better (No best value), Range = [0, +inf)
20	NMIS	Normalized Mutual Info Score	Bigger is better (Best = 1), Range = [0, 1]
21	RaS	Rand Score	Bigger is better (Best = 1), Range = [0, 1]
22	ARS	Adjusted Rand Score	Bigger is better (Best = 1), Range = [-1, 1]
23	FMS	Fowlkes Mallows Score	Bigger is better (Best = 1), Range = [0, 1]
24	HS	Homogeneity Score	Bigger is better (Best = 1), Range = [0, 1]
25	CS	Completeness Score	Bigger is better (Best = 1), Range = [0, 1]
26	VMS	V-Measure Score	Bigger is better (Best = 1), Range = [0, 1]
27	PrS	Precision Score	Bigger is better (Best = 1), Range = [0, 1]
28	ReS	Recall Score	Bigger is better (Best = 1), Range = [0, 1]
29	FmS	F-Measure Score	Bigger is better (Best = 1), Range = [0, 1]
30	CDS	Czekanowski Dice Score	Bigger is better (Best = 1), Range = [0, 1]
31	HGS	Hubert Gamma Score	Bigger is better (Best = 1), Range=[-1, +1]
32	JS	Jaccard Score	Bigger is better (Best = 1), Range = [0, 1]
33	KS	Kulczynski Score	Bigger is better (Best = 1), Range = [0, 1]
34	MNS	Mc Nemar Score	Bigger is better (No best value), Range=(-inf, +inf)
35	PhS	Phi Score	Bigger is better (No best value), Range = (-inf, +inf)
36	RTS	Rogers Tanimoto Score	Bigger is better (Best = 1), Range = [0, 1]
37	RRS	Russel Rao Score	Bigger is better (Best = 1), Range = [0, 1]
38	SS1S	Sokal Sneath1 Score	Bigger is better (Best = 1), Range = [0, 1]
39	SS2S	Sokal Sneath2 Score	Bigger is better (Best = 1), Range = [0, 1]
40	PuS	Purity Score	Bigger is better (Best = 1), Range = [0, 1]
41	ES	Entropy Score	Smaller is better (Best = 0), Range = [0, +inf)
42	TS	Tau Score	Bigger is better (No best value), Range = (-inf, +inf)
43	GAS	Gamma Score	Bigger is better (Best = 1), Range = [-1, 1]
44	GPS	Gplus Score	Smaller is better (Best = 0), Range = [0, 1]

CHAPTER

NINE

OFFICIAL LINKS

- Official source code repo: https://github.com/thieu1995/permetrics
- Official document: https://permetrics.readthedocs.io/
- Download releases: https://pypi.org/project/permetrics/
- Issue tracker: https://github.com/thieu1995/permetrics/issues
- Notable changes log: https://github.com/thieu1995/permetrics/blob/master/ChangeLog.md
- Official chat group: https://t.me/+fRVCJGuGJg1mNDg1
- **This project also related to our another projects which are "optimization" and "machine learning", check it here:**
  - https://github.com/thieu1995/mealpy
  - https://github.com/thieu1995/metaheuristics
  - https://github.com/thieu1995/opfunu
  - https://github.com/thieu1995/enoppy
  - https://github.com/thieu1995/MetaCluster
  - https://github.com/thieu1995/pfevaluator
  - https://github.com/thieu1995/mafese
  - https://github.com/aiir-team

CHAPTER

TEN

REFERENCE DOCUMENTS

1) https://www.debadityachakravorty.com/ai-ml/cmatrix/
2) https://neptune.ai/blog/evaluation-metrics-binary-classification
3) https://danielyang1009.github.io/model-performance-measure/
4) https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2
5) http://cran.nexr.com/web/packages/clusterCrit/vignettes/clusterCrit.pdf
6) https://publikationen.bibliothek.kit.edu/1000120412/79692380
7) https://torchmetrics.readthedocs.io/en/latest/
8) http://rasbt.github.io/mlxtend/user_guide/evaluate/lift_score/
9) https://www.baeldung.com/cs/multi-class-f1-score
10) https://kavita-ganesan.com/how-to-compute-precision-and-recall-for-a-multi-class-classification-problem/#.YoXMSqhBy3A
11) https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/

CHAPTER

ELEVEN

LICENSE

The project is licensed under GNU General Public License (GPL) V3 license.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Thieu Nguyen, Giang Nguyen, and Binh Minh Nguyen. Eo-cnn: an enhanced cnn model trained by equilibrium optimization for traffic transportation prediction. *Procedia Computer Science*, 176:800–809, 2020.

[2] Thieu Nguyen, Nhuan Tran, Binh Minh Nguyen, and Giang Nguyen. A resource usage prediction system using functional-link and genetic algorithm neural network for multivariate cloud metrics. In *2018 IEEE 11th conference on service-oriented computing and applications (SOCA)*, 49–56. IEEE, 2018.

[3] Takeyoshi Kato. Prediction of photovoltaic power generation output and network operation. In *Integration of Distributed Energy Resources in Power Systems*, pages 77–108. Elsevier, 2016.

[4] Thieu Nguyen, Binh Minh Nguyen, and Giang Nguyen. Building resource auto-scaler with functional-link neural network and adaptive bacterial foraging optimization. In *International Conference on Theory and Applications of Models of Computation*, 501–517. Springer, 2019.

[5] Timothy O Hodson, Thomas M Over, and Sydney S Foks. Mean squared error, deconstructed. *Journal of Advances in Modeling Earth Systems*, 13(12):e2021MS002681, 2021.

[6] Binh Minh Nguyen, Trung Tran, Thieu Nguyen, and Giang Nguyen. An improved sea lion optimization for workload elasticity prediction with neural networks. *International Journal of Computational Intelligence Systems*, 15(1):90, 2022.

[7] Thieu Nguyen, Bao Hoang, Giang Nguyen, and Binh Minh Nguyen. A new workload prediction model using extreme learning machine and enhanced tug of war optimization. *Procedia Computer Science*, 170:362–369, 2020.

[8] Tu Nguyen Thieu Nguyen, Binh Minh Nguyen, and Giang Nguyen. Efficient time-series forecasting using neural network and opposition-based coral reefs optimization. *International Journal of Computational Intelligence Systems*, 12(2):1144–1161, 2019.

[9] Chengyu Xie, Hoang Nguyen, Xuan-Nam Bui, Van-Thieu Nguyen, and Jian Zhou. Predicting roof displacement of roadways in underground coal mines using adaptive neuro-fuzzy inference system optimized by various physics-based optimization algorithms. *Journal of Rock Mechanics and Geotechnical Engineering*, 13(6):1452–1465, 2021.

[10] Ali Najah Ahmed, To Van Lam, Nguyen Duy Hung, Nguyen Van Thieu, Ozgur Kisi, and Ahmed El-Shafie. A comprehensive comparison of recent developed meta-heuristic algorithms for streamflow time series forecasting problem. *Applied Soft Computing*, 105:107282, 2021.

[11] Rodrigo Dlugosz da Silva, Marcelo Augusto de Aguiar, Marcelo Giovanetti Canteri, Juliandra Rodrigues Rosisca, Nilson Aparecido Vieira Junio, and others. Reference evapotranspiration for londrina, paraná, brazil: performance of different estimation methods. *Semina: Ciências Agrárias*, 38(4):2363–2374, 2017.

[12] Nguyen Van Thieu, Surajit Deb Barma, To Van Lam, Ozgur Kisi, and Amai Mahesha. Groundwater level modeling using augmented artificial ecosystem optimization. *Journal of Hydrology*, 617:129034, 2023.

[13] Binh Minh Nguyen, Bao Hoang, Thieu Nguyen, and Giang Nguyen. Nqsv-net: a novel queuing search variant for global space search and workload modeling. *Journal of Ambient Intelligence and Humanized Computing*, 12:27–46, 2021.

[14] Kapil Gupta. An integrated batting performance analytics model for women's cricket using principal component analysis and gini scores. *Decision Analytics Journal*, 4:100109, 2022.

[15] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, IV–317. IEEE, 2007.

[16] Bent Fuglede and Flemming Topsoe. Jensen-shannon divergence and hilbert space embedding. In *International symposium onInformation theory, 2004. ISIT 2004. Proceedings.*, 31. IEEE, 2004.

[17] Karl D Stephen and Alireza Kazemi. Improved normalization of time-lapse seismic data using normalized root mean square repeatability data to improve automatic production and seismic history matching in the nelson field. *Geophysical Prospecting*, 62(5):1009–1027, 2014.

[18] Karl G Jöreskog. Structural analysis of covariance and correlation matrices. *Psychometrika*, 43(4):443–477, 1978.

[19] Rolla Almodfer, Mohamed E Zayed, Mohamed Abd Elaziz, Moustafa M Aboelmaaref, Mohammed Mudhsh, and Ammar H Elsheikh. Modeling of a solar-powered thermoelectric air-conditioning system using a random vector functional link network integrated with jellyfish search algorithm. *Case Studies in Thermal Engineering*, 31:101797, 2022.

[20] Bernard Desgraupes. Clustering indices. *University of Paris Ouest-Lab Modal'X*, 1(1):34, 2013.

PYTHON MODULE INDEX

p

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

V

W

X