SeqMetrics

Release 1.3.4

Ather Abbas

Nov 15, 2022

CONTENTS

1 Installation **1**
- 1.1 using pip . . . 1
- 1.2 using github link . . . 1
- 1.3 using setup.py file . . . 1

2 Quick Start **3**
- 2.1 RegressionMetrics . . . 3
- 2.2 ClassificationMetrics . . . 3

3 Metrics **5**
- 3.1 Metrics . . . 5

4 Regression Metrics **9**
- 4.1 RegressionMetrics . . . 9

5 Classification Metrics **21**
- 5.1 ClassificationMetrics . . . 21

6 utility functions **27**
- 6.1 Utils . . . 27

7 Indices and tables **29**

Index **31**

CHAPTER

ONE

INSTALLATION

1.1 using pip

The most easy way to install ai4water is using `pip`

```
pip install SeqMetrics
```

1.2 using github link

You can use github link for install SeqMetrics.

```
python -m pip install git+https://github.com/AtrCheema/SeqMetrics.git
```

1.3 using setup.py file

go to folder where repository is downloaded

```
python setup.py install
```

CHAPTER

TWO

QUICK START

2.1 RegressionMetrics

```
>>> import numpy as np
>>> from SeqMetrics import RegressionMetrics

>>> true = np.random.random((20, 1))
>>> pred = np.random.random((20, 1))

>>> er = RegressionMetrics(true, pred)

>>> for m in er.all_methods: print("{:20}".format(m)) # get names of all availabe methods

>>> er.nse()   # calculate Nash Sutcliff efficiency

>>> er.calculate_all(verbose=True)  # or calculate errors using all available methods
```

2.2 ClassificationMetrics

```
>>> import numpy as np
>>> from SeqMetrics import ClassificationMetrics

using boolean array

>>> t = np.array([True, False, False, False])
>>> p = np.array([True, True, True, True])
>>> metrics = ClassificationMetrics(t, p)
>>> accuracy = metrics.accuracy()

binary classification with numerical labels

>>> true = np.array([1, 0, 0, 0])
>>> pred = np.array([1, 1, 1, 1])
>>> metrics = ClassificationMetrics(true, pred)
>>> accuracy = metrics.accuracy()

multiclass classification with numerical labels

```

(continues on next page)

(continued from previous page)

```
>>> true = np.random.randint(1, 4, 100)
>>> pred = np.random.randint(1, 4, 100)
>>> metrics = ClassificationMetrics(true, pred)
>>> accuracy = metrics.accuracy()

You can also provide logits instead of labels.

>>> predictions = np.array([[0.25, 0.25, 0.25, 0.25],
>>>                         [0.01, 0.01, 0.01, 0.96]])
>>> targets = np.array([[0, 0, 0, 1],
>>>                     [0, 0, 0, 1]])
>>> metrics = ClassificationMetrics(targets, predictions, multiclass=True)
>>> metrics.cross_entropy()
...  0.71355817782
```

CHAPTER

THREE

METRICS

The `Metrics` class does some preprocessing of arrays if required.

3.1 Metrics

class SeqMetrics.**Metrics**(*true: Union[ndarray, list]*, *predicted: Union[ndarray, list]*, *replace_nan: Optional[Union[int, float]] = None*, *replace_inf: Optional[Union[int, float]] = None*, *remove_zero: bool = False*, *remove_neg: bool = False*, *metric_type: str = 'regression'*, *np_errstate: Optional[dict] = None*)

Bases: `object`

This class does some pre-processign and handles metadata regaring true and predicted arrays.

The arguments other than *true* and *predicted* are dynamic i.e. they can be changed from outside the class. This means the user can change their value after creating the class. This will be useful if we want to calculate an error once by ignoring NaN and then by not ignoring the NaNs. However, the user has to run the method *treat_arrays* in order to have the changed values impact on true and predicted arrays. For ducussion about impact of performance metric see German climate computing website.

__init__(*true: Union[ndarray, list]*, *predicted: Union[ndarray, list]*, *replace_nan: Optional[Union[int, float]] = None*, *replace_inf: Optional[Union[int, float]] = None*, *remove_zero: bool = False*, *remove_neg: bool = False*, *metric_type: str = 'regression'*, *np_errstate: Optional[dict] = None*)

Parameters

- **true** (*array like,*) – ture/observed/actual/target values
- **predicted** (*array like,*) – simulated values
- **replace_nan** (*default None. if not None, then NaNs in true*) – and predicted will be replaced by this value.
- **replace_inf** (*default None, if not None, then inf vlaues in true and*) – predicted will be replaced by this value.
- **remove_zero** (*default False, if True, the zero values in true*) – or predicted arrays will be removed. If a zero is found in one array, the corresponding value in the other array will also be removed.
- **remove_neg** (*default False, if True, the negative values in true*) – or predicted arrays will be removed.
- **metric_type** (*type of metric.*) –
- **np_errstate** (*dict*) – any keyword options for np.errstate() to calculate np.log1p

calculate_all(*statistics=False*, *verbose=False*, *write=False*, *name=None*) → dict

calculates errors using all available methods except brier_score.. write: bool, if True, will write the calculated errors in file. name: str, if not None, then must be path of the file in which to write.

Parameters

- **verbose** (*bool, optional*) – if True, will print the calculated errors. The default is False.
- **write** (*bool, optional*) – if True, will write the calculated errors in file. The default is False.
- **name** (*str, optional*) – if not None, then must be path of the file in which to write. The default is None.

Returns
dictionary of calculated errors.

Return type
dict

Examples

```
>>> import numpy as np
>>> from SeqMetrics import RegressionMetrics
>>> true = np.random.random(100)
>>> predicted = np.random.random(100)
>>> metrics = RegressionMetrics(true, predicted)
>>> metrics.calculate_all()
```

calculate_minimal() → dict

Calculates some basic metrics.

Returns
Dictionary with all metrics

Return type
dict

Examples

```
>>> import numpy as np
>>> from SeqMetrics import RegressionMetrics
>>> true = np.random.random(100)
>>> predicted = np.random.random(100)
>>> metrics = RegressionMetrics(true, predicted)
>>> metrics.calculate_minimal()
```

calculate_scale_dependent_metrics() → dict

Calculates scale dependent metrics

Returns
Dictionary with all metrics

Return type
dict

Examples

```
>>> import numpy as np
>>> from SeqMetrics import RegressionMetrics
>>> true = np.random.random(100)
>>> predicted = np.random.random(100)
>>> metrics = RegressionMetrics(true, predicted)
>>> metrics.calculate_scale_dependent_metrics()
```

calculate_scale_independent_metrics() → dict

Calculates scale independent metrics

Returns
Dictionary with all metrics

Return type
dict

Examples

```
>>> import numpy as np
>>> from SeqMetrics import RegressionMetrics
>>> true = np.random.random(100)
>>> predicted = np.random.random(100)
>>> metrics = RegressionMetrics(true, predicted)
>>> metrics.calculate_scale_independent_metrics()
```

composite_metrics()

property log1p_p

property log1p_t

property log_p

property log_t

mse(*weights=None*) → float

mean square error

percentage_metrics()

relative_metrics()

property remove_neg

property remove_zero

property replace_inf

property replace_nan

scale_dependent_metrics()

stats(*verbose: bool = False*) → dict

returs some important stats about true and predicted values.

`treat_values()`

This function is applied by default at the start/at the time of initiating the class. However, it can used any time after that. This can be handy if we want to calculate error first by ignoring nan and then by no ignoring nan. Adopting from HydroErr . Removes the nan, negative, and inf values in two numpy arrays

CHAPTER

FOUR

REGRESSION METRICS

4.1 RegressionMetrics

class SeqMetrics.**RegressionMetrics**(**args*, ***kwargs*)

Bases: *Metrics*

Calculates more than 100 regression performance metrics related to sequence data.

Example

```
>>> import numpy as np
>>> from SeqMetrics import RegressionMetrics
>>> t = np.random.random(10)
>>> p = np.random.random(10)
>>> errors = RegressionMetrics(t,p)
>>> all_errors = errors.calculate_all()
```

__init__(**args*, ***kwargs*)

Initializes Metrics.

args and kwargs go to parent class *SeqMetrics.Metrics*.

JS() → float

Jensen-shannon divergence

abs_pbias() → float

Absolute Percent bias

acc() → float

Anomaly correction coefficient. See Langland et al., 2012; Miyakoda et al., 1972 and Murphy et al., 1989.

adjusted_r2() → float

Adjusted R squared.

agreement_index() → float

Agreement Index (d) developed by Willmott, 1981.

It detects additive and pro-portional differences in the observed and simulated means and vari-ances Moriasi et al., 2015. It is overly sensitive to extreme values due to the squared differences. It can also be used as a

substitute for R2 to identify the degree to which model predic-tions are error-free.

$$d = 1 - \frac{\sum_{i=1}^{N}(e_i - s_i)^2}{\sum_{i=1}^{N}(|s_i - \bar{e}| + |e_i - \bar{e}|)^2}$$

aic(*p=1*) → float

Akaike Information Criterion. Modifying from this source

aitchison(*center='mean'*) → float

Aitchison distance. used in Zhang et al., 2020

amemiya_adj_r2() → float

Amemiya's Adjusted R-squared

amemiya_pred_criterion() → float

Amemiya's Prediction Criterion

bias() → float

Bias as and given by Gupta1998 et al., 1998

$$Bias = \frac{1}{N}\sum_{i=1}^{N}(e_i - s_i)$$

bic(*p=1*) → float

Bayesian Information Criterion

Minimising the BIC is intended to give the best model. The model chosen by the BIC is either the same as that chosen by the AIC, or one with fewer terms. This is because the BIC penalises the number of parameters more heavily than the AIC. Modified after RegscorePy.

brier_score() → float

Adopted from SkillMetrics Calculates the Brier score (BS), a measure of the mean-square error of probability forecasts for a dichotomous (two-category) event, such as the occurrence/non-occurrence of precipitation. The score is calculated using the formula:

$$BS = sum_(n = 1)^{N}(f_n - o_n)^2/N$$

where f is the forecast probabilities, o is the observed probabilities (0 or 1), and N is the total number of values in f & o. Note that f & o must have the same number of values, and those values must be in the range [0,1].

Returns
BS : Brier score

Return type
float

References

Glenn W. Brier, 1950: Verification of forecasts expressed in terms of probabilities. Mon. We. Rev., 78, 1-23. D. S. Wilks, 1995: Statistical Methods in the Atmospheric Sciences. Cambridge Press. 547 pp.

calculate_hydro_metrics()

Calculates all metrics for hydrological data.

Returns
Dictionary with all metrics

Return type
dict

centered_rms_dev() → float

Modified after SkillMetrics. Calculates the centered root-mean-square (RMS) difference between true and predicted using the formula: (E')^2 = sum_(n=1)^N [(p_n - mean(p))(r_n - mean(r))]^2/N where p is the predicted values, r is the true values, and N is the total number of values in p & r.

Output: CRMSDIFF : centered root-mean-square (RMS) difference (E')^2

corr_coeff() → float

Pearson correlation coefficient. It measures linear correlatin between true and predicted arrays. It is sensitive to outliers. Reference: Pearson, K 1895.

$$r = \frac{\sum_{i=1}^{n}(e_i - \bar{e})(s_i - \bar{s})}{\sqrt{\sum_{i=1}^{n}(e_i - \bar{e})^2}\sqrt{\sum_{i=1}^{n}(s_i - \bar{s})^2}}$$

cosine_similarity() → float

It is a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors oriented at 90° relative to each other have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. See

covariance() → float

Covariance
Covariance = frac{1}{N} sum_{i=1}^{N}((e_{i} - bar{e}) * (s_{i} - bar{s}))

cronbach_alpha() → float

It is a measure of internal consitency of data. See ucla and stackoverflow pages for more info.

decomposed_mse() → float

Decomposed MSE developed by Kobayashi and Salam (2000)

$$dMSE = (\frac{1}{N}\sum_{i=1}^{N}(e_i - s_i))^2 + SDSD + LCSSDSD = (\sigma(e) - \sigma(s))^2 LCS = 2\sigma(e)\sigma(s) * (1 - \frac{\sum_{i=1}^{n}(e_i - \bar{e})($$
$$\sqrt{\sum_{i=1}^{n}(e_i - \bar{e})^2}\sqrt{\sum$$

euclid_distance() → float

Euclidian distance

Referneces: Kennard et al., 2010

exp_var_score(*weights=None*) → Optional[float]

Explained variance score . Best value is 1, lower values are less accurate.

expanded_uncertainty(*cov_fact=1.96*) → float

By default it calculates uncertainty with 95% confidence interval. 1.96 is the coverage factor corresponding 95% confidence level .This indicator is used in order to show more information about the model deviation. Using formula from by Behar et al., 2015 and Gueymard et al., 2014.

fdc_fhv(*h: float = 0.02*) → float

modified Kratzert2018 code. Peak flow bias of the flow duration curve (Yilmaz 2008). used in kratzert et al., 2018

Parameters
h (*float*) – Must be between 0 and 1.

Return type
Bias of the peak flows

fdc_flv(*low_flow: float = 0.3*) → float

bias of the bottom 30 % low flows. modified Kratzert code used in kratzert et al., 2018

Parameters
low_flow (*float, optional*) – Upper limit of the flow duration curve. E.g. 0.3 means the bottom 30% of the flows are considered as low flows, by default 0.3

Return type
float

gmae() → float

Geometric Mean Absolute Error

gmean_diff() → float

Geometric mean difference. First geometric mean is calculated for each of two samples and their difference is calculated.

gmrae(*benchmark: Optional[ndarray] = None*) → float

Geometric Mean Relative Absolute Error

inrse() → float

Integral Normalized Root Squared Error

irmse() → float

Inertial RMSE. RMSE divided by standard deviation of the gradient of true.

kendaull_tau(*return_p=False*) → Union[float, tuple]

Kendall's tau .used in Probst et al., 2019.

kge(*return_all=False*)

Kling-Gupta Efficiency Gupta, Kling, Yilmaz, Martinez, 2009, Decomposition of the mean squared error and NSE performance

criteria: Implications for improving hydrological modelling

output:
kge: Kling-Gupta Efficiency cc: correlation alpha: ratio of the standard deviation beta: ratio of the mean

kge_bound() → float

Bounded Version of the Original Kling-Gupta **Efficiency_**

kge_mod(*return_all=False*)

Modified Kling-Gupta **Efficiency_** .

kge_np(*return_all=False*)

Non parametric Kling-Gupta Efficiency

output:
kge: Kling-Gupta Efficiency cc: correlation alpha: ratio of the standard deviation beta: ratio of the mean

References

Pool, Vis, and Seibert, 2018 Evaluating model performance: towards a non-parametric variant of the Kling-Gupta efficiency, Hydrological Sciences Journal. https://doi.org/10.1080/02626667.2018.1552002

kgenp_bound()

Bounded Version of the Non-Parametric Kling-Gupta Efficiency

kgeprime_c2m() → float

Bounded Version of the Modified Kling-Gupta **Efficiency_**

kl_sym() → Optional[float]

Symmetric kullback-leibler divergence

lm_index(*obs_bar_p=None*) → float

Legate-McCabe Efficiency Index. Less sensitive to outliers in the data. The larger, the better

Parameters
obs_bar_p (*float,*) – Seasonal or other selected average. If None, the mean of the observed array will be used.

log_nse(*epsilon=0.0*) → float

log Nash-Sutcliffe model efficiency

$$NSE = 1 - \frac{\sum_{i=1}^{N}(log(e_i) - log(s_i))^2}{\sum_{i=1}^{N}(log(e_i) - log(\bar{e})^2} - 1) * -1$$

log_prob() → float

Logarithmic probability distribution

maape() → float

Mean Arctangent Absolute Percentage Error Note: result is NOT multiplied by 100

mae(*true=None*, *predicted=None*) → float

Mean Absolute Error. It is less sensitive to outliers as compared to mse/rmse.

mapd() → float

Mean absolute percentage deviation.

mape() → float

Mean Absolute Percentage Error. The MAPE is often used when the quantity to predict is known to remain way above zero. It is useful when the size or size of a prediction variable is significant in evaluating the accuracy of a prediction. It has advantages of scale-independency and interpretability. However, it has the significant disadvantage that it produces infinite or undefined values for zero or close-to-zero actual values.

mare() → float

Mean Absolute Relative Error. When expressed in %age, it is also known as mape.

mase(*seasonality: int = 1*)

Mean Absolute Scaled Error. Baseline (benchmark) is computed with naive forecasting (shifted by @seasonality) modified after[11]. It is the ratio of MAE of used model and MAE of naive forecast.

[11] https://gist.github.com/bshishov/5dc237f59f019b26145648e2124ca1c9

References

Hyndman, R. J. (2006). Another look at forecast-accuracy metrics for intermittent demand. Foresight: The International Journal of Applied Forecasting, 4(4), 43-46.

max_error() → float

maximum absolute error

mb_r() → float

Mielke-Berry R value. Berry and Mielke, 1988.

References

Mielke, P. W., & Berry, K. J. (2007). Permutation methods: a distance function approach. Springer Science & Business Media.

mbe() → float

Mean bias error. This indicator expresses a tendency of model to underestimate (negative value) or overestimate (positive value) global radiation, while the MBE values closest to zero are desirable. The drawback of this test is that it does not show the correct performance when the model presents overestimated and underestimated values at the same time, since overestimation and underestimation values cancel each other. [1]

mbrae(*benchmark: Optional[ndarray] = None*) → float

Mean Bounded Relative Absolute Error

mda() → float

Mean Directional Accuracy modified after

mdape() → float

Median Absolute Percentage Error

mde() → float

Median Error

mdrae(*benchmark: Optional[ndarray] = None*) → float

Median Relative Absolute Error

me()

Mean error

mean_bias_error() → float

Mean Bias Error It represents overall bias error or systematic error. It shows average interpolation bias; i.e. average over- or underestimation. [1][2].This indicator expresses a tendency of model to underestimate (negative value) or overestimate (positive value) global radiation, while the MBE values closest to zero are desirable. The drawback of this test is that it does not show the correct performance when the model presents overestimated and underestimated values at the same time, since overestimation and underestimation values cancel each other.

References

- **Willmott, C. J., & Matsuura, K. (2006). On the use of dimensioned measures of error to evaluate the performance**
  of spatial interpolators. International Journal of Geographical Information Science, 20(1), 89-102. https://doi.org/10.1080/13658810500286976
- **Valipour, M. (2015). Retracted: Comparative Evaluation of Radiation-Based Methods for Estimation of Potential**
  Evapotranspiration. Journal of Hydrologic Engineering, 20(5), 04014068. https://dx.doi.org/10.1061/(ASCE)HE.1943-5584.0001066
- https://doi.org/10.1016/j.rser.2015.08.035

mean_gamma_deviance(*weights=None*) → float

mean gamma deviance

mean_poisson_deviance(*weights=None*) → float

mean poisson deviance

mean_var() → float

Mean variance

med_seq_error() → float

Median Squared Error Same as mse but it takes median which reduces the impact of outliers.

median_abs_error() → float

median absolute error

mle() → float

Mean log error

mod_agreement_index(*j=1*) → float

Modified agreement of index. j: int, when j==1, this is same as agreement_index. Higher j means more impact of outliers.

mpe() → float

Mean Percentage Error

mrae(*benchmark: Optional[ndarray] = None*)

Mean Relative Absolute Error

msle(*weights=None*) → float

mean square logrithmic error

norm_ae() → float

Normalized Absolute Error

norm_ape() → float

Normalized Absolute Percentage Error

norm_euclid_distance() → float

Normalized Euclidian distance

nrmse() → float

Normalized Root Mean Squared Error

nrmse_ipercentile(*q1=25*, *q2=75*) → float

RMSE normalized by inter percentile range of true. This is least sensitive to outliers. q1: any interger between 1 and 99 q2: any integer between 2 and 100. Should be greater than q1. Reference: Pontius et al., 2008.

nrmse_mean() → float

Mean Normalized RMSE RMSE normalized by mean of true values.This allows comparison between datasets with different scales.

Reference: Pontius et al., 2008

nrmse_range() → float

Range Normalized Root Mean Squared Error. RMSE normalized by true values. This allows comparison between data sets with different scales. It is more sensitive to outliers.

Reference: Pontius et al., 2008

nse() → float

Nash-Sutcliff Efficiency.

It determine how well the model simulates trends for the output response of concern. But cannot help identify model bias and cannot be used to identify differences in timing and magnitude of peak flows and shape of recession curves; in other words, it cannot be used for single-event simulations. It is sensitive to extreme values due to the squared differ-ences [1]. To make it less sensitive to outliers, [2] proposed log and relative nse.

References

- **Moriasi, D. N., Gitau, M. W., Pai, N., & Daggupati, P. (2015). Hydrologic and water quality models:**
  Performance measures and evaluation criteria. Transactions of the ASABE, 58(6), 1763-1785.
- **Krause, P., Boyle, D., & Bäse, F. (2005). Comparison of different efficiency criteria for hydrological**
  model assessment. Adv. Geosci., 5, 89-97. https://dx.doi.org/10.5194/adgeo-5-89-2005.

nse_alpha() → float

Alpha decomposition of the NSE, see Gupta_ et al. 2009 used in kratzert et al., 2018

Returns
Alpha decomposition of the NSE

Return type
float

nse_beta() → float

Beta decomposition of NSE. See Gupta et. al 2009 . used in kratzert et al., 2018

Returns
Beta decomposition of the NSE

Return type
float

nse_bound() → float

Bounded Version of the Nash-Sutcliffe Efficiency (nse)

nse_mod(*j=1*) → float

Gives less weightage of outliers if j=1 and if j>1, gives more weightage to outliers. Reference: Krause et al., 2005

nse_rel() → float

Relative NSE.

pbias() → float

Percent Bias. It determine how well the model simulates the average magnitudes for the output response of interest. It can also determine over and under-prediction. It cannot be used (1) for single-event simula-tions to identify differences in timing and magnitude of peak flows and the shape of recession curves nor (2) to determine how well the model simulates residual variations and/or trends for the output response of interest. It can give a deceiving rating of model performance if the model overpredicts as much as it underpredicts, in which case PBIAS will be close to zero even though the model simulation is poor. [1]

[1] Moriasi et al., 2015

r2() → float

Quantifies the percent of variation in the response that the 'model' explains. The 'model' here is anything from which we obtained predicted array. It is also called coefficient of determination or square of pearson correlation coefficient. More heavily affected by outliers than pearson correlatin r.

r2_score(*weights=None*)

This is not a symmetric function. Unlike most other scores, R^2 score may be negative (it need not actually be the square of a quantity R). This metric is not well-defined for single samples and will return a NaN value if n_samples is less than two.

rae() → float

Relative Absolute Error (aka Approximation Error)

ref_agreement_index() → float

Refined Index of Agreement. From -1 to 1. Larger the better. Refrence: Willmott et al., 2012

rel_agreement_index() → float

Relative index of agreement. from 0 to 1. larger the better.

relative_rmse() → float

Relative Root Mean Squared Error

$$RRMSE = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(e_i - s_i)^2}}{\bar{e}}$$

rmdspe() → float

Root Median Squared Percentage Error

rmse(*weights=None*) → float

root mean square error

rmsle() → float

Root mean square log error.

This error is less sensitive to outliers . Compared to RMSE, RMSLE only considers the relative error between predicted and actual values, and the scale of the error is nullified by the log-transformation. Furthermore, RMSLE penalizes underestimation more than overestimation. This is especially useful in those studies where the underestimation of the target variable is not acceptable but overestimation can be tolerated .

rmspe() → float

Root Mean Square Percentage Error_ .

rmsse(*seasonality: int = 1*) → float

Root Mean Squared Scaled Error

rrse() → float

Root Relative Squared Error

rse() → float

Relative Squared Error

rsr() → float

Moriasi et al., 2007. It incorporates the benefits of error index statistics andincludes a scaling/normalization factor, so that the resulting statistic and reported values can apply to various constitu-ents.

sa() → float

Spectral angle. From -pi/2 to pi/2. Closer to 0 is better. It measures angle between two vectors in hyperspace indicating how well the shape of two arrays match instead of their magnitude. Reference: Robila and Gershman, 2005.

sc() → float

Spectral correlation. It varies from -pi/2 to pi/2. Closer to 0 is better.

sga() → float

Spectral gradient angle. It varies from -pi/2 to pi/2. Closer to 0 is better.

sid() → float

Spectral Information Divergence. From -pi/2 to pi/2. Closer to 0 is better.

skill_score_murphy() → float

Adopted from here . Calculate non-dimensional skill score (SS) between two variables using definition of Murphy (1988) using the formula:

$$SS = 1 - RMSE^2/SDEV^2$$
$$SDEV is the standard deviation of the true values$$
$$SDEV^2 = sum_(n = 1)^N [r_n - mean(r)]^2/(N - 1)$$

where p is the predicted values, r is the reference values, and N is the total number of values in p & r. Note that p & r must have the same number of values. A positive skill score can be interpreted as the percentage of improvement of the new model forecast in comparison to the reference. On the other hand, a negative skill score denotes that the forecast of interest is worse than the referencing forecast. Consequently, a value of zero denotes that both forecasts perform equally [MLAir, 2020].

Returns
flaot

References

Allan H. Murphy, 1988: Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient. Mon. Wea. Rev., 116, 2417-2424. doi: http//dx.doi.org/10.1175/1520-0493(1988)<2417:SSBOTM>2.0.CO;2

smape() → float

Symmetric Mean Absolute Percentage Error_. Adoption from .

smdape() → float

Symmetric Median Absolute Percentage Error Note: result is NOT multiplied by 100

spearmann_corr() → float

Separmann correlation coefficient.

This is a nonparametric metric and assesses how well the relationship between the true and predicted data can be described using a monotonic function.

sse() → float

Sum of squared **errors_** (model vs actual). It is measure of how far off our model's predictions are from the observed values. A value of 0 indicates that all predications are spot on. A non-zero value indicates errors.

This is also called residual sum of squares (RSS) or sum of squared residuals as per tutorialspoint .

std_ratio(***kwargs*) → float

ratio of standard deviations of predictions and trues. Also known as standard ratio, it varies from 0.0 to infinity while 1.0 being the perfect value.

umbrae(*benchmark: Optional[ndarray] = None*)

Unscaled Mean Bounded Relative Absolute Error

ve() → float

Volumetric efficiency. from 0 to 1. Smaller the better. Reference: Criss and Winston 2008.

volume_error() → float

Returns the Volume Error (Ve). It is an indicator of the agreement between the averages of the simulated and observed runoff (i.e. long-term water balance). used in Reynolds paper:

$$Sum(self.predicted - true)/sum(self.predicted)$$

References

Reynolds, J.E., S. Halldin, C.Y. Xu, J. Seibert, and A. Kauffeldt. 2017. "Sub-Daily Runoff Predictions Using Parameters Calibrated on the Basis of Data with a Daily Temporal Resolution." Journal of Hydrology 550 (July):399?411.

wape() → float

weighted absolute percentage error (wape)

It is a variation of mape but more suitable for intermittent and low-volume data.

watt_m() → float

Watterson's M. Refrence: Watterson., 1996

wmape() → float

Weighted Mean Absolute Percent **Error_** .

CHAPTER

FIVE

CLASSIFICATION METRICS

5.1 ClassificationMetrics

class SeqMetrics.**ClassificationMetrics**(*true*, *predicted*, *multiclass: bool = False*, **args*, ***kwargs*)

Bases: *Metrics*

Calculates classification metrics.

Parameters

- **true** (*array-like of shape = [n_samples] or [n_samples, n_classes]*) – True class labels.
- **predicted** (*array-like of shape = [n_samples] or [n_samples, n_classes]*) – Predicted class labels.
- **multiclass** (*boolean, optional*) – If true, it is assumed that the true labels are multi-class.
- ****kwargs** (*optional*) – Additional arguments to be passed to the *Metrics* class.

Examples

```
>>> import numpy as np
>>> from SeqMetrics import ClassificationMetrics
```

boolean array

```
>>> t = np.array([True, False, False, False])
>>> p = np.array([True, True, True, True])
>>> metrics = ClassificationMetrics(t, p)
>>> accuracy = metrics.accuracy()
```

binary classification with numerical labels

```
>>> true = np.array([1, 0, 0, 0])
>>> pred = np.array([1, 1, 1, 1])
>>> metrics = ClassificationMetrics(true, pred)
>>> accuracy = metrics.accuracy()
```

multiclass classification with numerical labels

```
>>> true = np.random.randint(1, 4, 100)
>>> pred = np.random.randint(1, 4, 100)
>>> metrics = ClassificationMetrics(true, pred)
>>> accuracy = metrics.accuracy()
```

You can also provide logits instead of labels.

```
>>> predictions = np.array([[0.25, 0.25, 0.25, 0.25],
>>>                         [0.01, 0.01, 0.01, 0.96]])
>>> targets = np.array([[0, 0, 0, 1],
>>>                     [0, 0, 0, 1]])
>>> metrics = ClassificationMetrics(targets, predictions, multiclass=True)
>>> metrics.cross_entropy()
...  0.71355817782
```

Working with categorical values is seamless

```
>>> true = np.array(['a', 'b', 'b', 'b'])
>>> pred = np.array(['a', 'a', 'a', 'a'])
>>> metrics = ClassificationMetrics(true, pred)
>>> accuracy = metrics.accuracy()
```

same goes for multiclass categorical labels

```
>>> t = np.array(['car', 'truck', 'truck', 'car', 'bike', 'truck'])
>>> p = np.array(['car', 'car',   'bike',  'car', 'bike', 'truck'])
>>> metrics = ClassificationMetrics(targets, predictions, multiclass=True)
>>> print(metrics.calculate_all())
```

__init__(*true*, *predicted*, *multiclass: bool = False*, **args*, ***kwargs*)

Parameters

- **true** (*array like,*) – ture/observed/actual/target values
- **predicted** (*array like,*) – simulated values
- **replace_nan** (*default None. if not None, then NaNs in true*) – and predicted will be replaced by this value.
- **replace_inf** (*default None, if not None, then inf vlaues in true and*) – predicted will be replaced by this value.
- **remove_zero** (*default False, if True, the zero values in true*) – or predicted arrays will be removed. If a zero is found in one array, the corresponding value in the other array will also be removed.
- **remove_neg** (*default False, if True, the negative values in true*) – or predicted arrays will be removed.
- **metric_type** (*type of metric.*) –
- **np_errstate** (*dict*) – any keyword options for np.errstate() to calculate np.log1p

accuracy(*normalize: bool = True*) → float

calculates accuracy

Parameters

normalize (*bool*) –

Return type
float

Examples

```
>>> import numpy as np
>>> from SeqMetrics import ClassificationMetrics
>>> true = np.array([1, 0, 0, 0])
>>> pred = np.array([1, 1, 1, 1])
>>> metrics = ClassificationMetrics(true, pred)
>>> print(metrics.accuracy())
```

balanced_accuracy(*average=None*) → float

balanced accuracy. It performs better on imbalanced datasets.

confusion_matrix(*normalize=False*)

calculates confusion matrix

Parameters
normalize (*str, [None, 'true', 'pred', 'all']*) – If None, no normalization is done.

Returns
confusion matrix

Return type
ndarray

Examples

```
>>> import numpy as np
>>> from SeqMetrics import ClassificationMetrics
>>> true = np.array([1, 0, 0, 0])
>>> pred = np.array([1, 1, 1, 1])
>>> metrics = ClassificationMetrics(true, pred)
>>> metrics.confusion_matrix()
```

multiclass classification

```
>>> true = np.random.randint(1, 4, 100)
>>> pred = np.random.randint(1, 4, 100)
>>> metrics = ClassificationMetrics(true, pred)
>>> metrics.confusion_matrix()
```

cross_entropy(*epsilon=1e-12*) → float

Computes cross entropy between targets (encoded as one-hot vectors) and predictions.

Return type
scalar

error_rate()

Error rate is the number of all incorrect predictions divided by the total number of samples in data.

f1_score(*average=None*) → Union[ndarray, float]

Calculates f1 score according to following formula f1_score = 2 * (precision * recall) / (precision + recall)

Parameters
average (*str, optional*) – It can be `macro` or `weighted`. or `micro`

Return type
array or float

Examples

```
>>> import numpy as np
>>> from SeqMetrics import ClassificationMetrics
>>> true = np.array([1, 0, 0, 0])
>>> pred = np.array([1, 1, 1, 1])
>>> metrics = ClassificationMetrics(true, pred)
>>> calc_f1_score = metrics.f1_score()
...
>>> print(metrics.f1_score(average="macro"))
>>> print(metrics.f1_score(average="weighted"))
```

f2_score(*average=None*)

f2 score

false_discovery_rate()

False discovery rate
FP / (TP + FP)

false_negative_rate()

False Negative Rate or miss rate.

FN / (FN + TP)

false_omission_rate(*average=None*)

False omission rate

FN / (FN + TN)

false_positive_rate()

False positive rate is the number of incorrect positive predictions divided by the total number of negatives. Its best value is 0.0 and worst value is 1.0. It is also called probability of false alarm or fall-out.

TP / (TP + TN)

fowlkes_mallows_index(*average=None*)

Fowlkes–Mallows index

sqrt(PPV * TPR)

PPV is positive predictive value or precision. TPR is true positive rate or recall or sensitivity

https://en.wikipedia.org/wiki/Fowlkes%E2%80%93Mallows_index

mathews_corr_coeff()

Methew's correlation coefficient

negative_likelihood_ratio(*average=None*)

Negative likelihood ratio

1 - sensitivity / specificity

https://en.wikipedia.org/wiki/Likelihood_ratios_in_diagnostic_testing#positive_likelihood_ratio

negative_predictive_value()

Negative Predictive Value TN/(TN+FN)

positive_likelihood_ratio(*average=None*)

Positive likelihood ratio sensitivity / 1-specificity

precision(*average=None*)

Returns precision score, also called positive predictive value. It is number of correct positive predictions divided by the total number of positive predictions. TP/(TP+FP)

Parameters
average (string, [None, `macro`, `weighted`, `micro`]) –

Examples

```
>>> import numpy as np
>>> from SeqMetrics import ClassificationMetrics
>>> true = np.array([1, 0, 0, 0])
>>> pred = np.array([1, 1, 1, 1])
>>> metrics = ClassificationMetrics(true, pred)
>>> print(metrics.precision())
...
>>> print(metrics.precision(average="macro"))
>>> print(metrics.precision(average="weighted"))
```

prevalence_threshold(*average=None*)

Prevalence threshold

sqrt(FPR) / (sqrt(TPR) + sqrt(FPR))

TPR is true positive rate or recall

recall(*average=None*)

It is also called sensitivity or true positive rate. It is number of correct positive predictions divided by the total number of positives Formula :

True Posivitive / True Positive + False Negative

Parameters
average (*str (default=None)*) – one of None, `macro`, `weighted`, or `micro`

specificity(*average=None*)

It is also called true negative rate or selectivity. It is the probability that the predictions are negative when the true labels are also negative. It is number of correct negative predictions divided by the total number of negatives.

It's formula is following TN / TN+FP

Examples

```
>>> import numpy as np
>>> from SeqMetrics import ClassificationMetrics
>>> true = np.array([1, 0, 0, 0])
>>> pred = np.array([1, 1, 1, 1])
>>> metrics = ClassificationMetrics(true, pred)
>>> print(metrics.specificity())
...
>>> print(metrics.specificity(average="macro"))
>>> print(metrics.specificity(average="weighted"))
```

youden_index(*average=None*)

Youden index, also known as informedness

j = TPR + TNR 1 = sensitivity + specificity - 1

https://en.wikipedia.org/wiki/Youden%27s_J_statistic

CHAPTER

SIX

UTILITY FUNCTIONS

6.1 Utils

class SeqMetrics.utils.**plot_metrics**(*metrics: dict*, *ranges: tuple = ((0.0, 1.0), (1.0, 10), (10, 1000))*, *exclude: Optional[list] = None*, *plot_type: str = 'bar'*, *max_metrics_per_fig: int = 15*, *show: bool = True*, *save: bool = False*, *save_path: str = ''*, ***kwargs*)

Plots the metrics given as dictionary as radial or bar plot between specified ranges.

Parameters

- **metrics** – dict dictionary whose keys are names are erros and values are error values.
- **ranges** – tuple of tuples defining range of errors to plot in one plot
- **exclude** – list List of metrics to be excluded from plotting.
- **max_metrics_per_fig** – int maximum number of metrics to show in one figure.
- **plot_type** – either of `radial` or `bar`.
- **show** – If, then figure will be shown/drawn
- **save** – if True, the figure will be saved.
- **save_path** – if given, the figure will the saved at this location.
- **kwargs** – keyword arguments for plotting

Retuns: None

Examples

```
>>> import numpy as np
>>> from SeqMetrics import RegressionMetrics
>>> from SeqMetrics import plot_metrics
>>> t = np.random.random((20, 1))
>>> p = np.random.random((20, 1))
>>> er = RegressionMetrics(t, p)
>>> all_errors = er.calculate_all()
>>> plot_metrics(all_errors, plot_type='bar', max_metrics_per_fig=50)
>>> # or draw the radial plot
>>> plot_metrics(all_errors, plot_type='radial', max_metrics_per_fig=50)
```

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

__init__() (*SeqMetrics.ClassificationMetrics method*), 22
__init__() (*SeqMetrics.Metrics method*), 5
__init__() (*SeqMetrics.RegressionMetrics method*), 9

A

abs_pbias() (*SeqMetrics.RegressionMetrics method*), 9
acc() (*SeqMetrics.RegressionMetrics method*), 9
accuracy() (*SeqMetrics.ClassificationMetrics method*), 22
adjusted_r2() (*SeqMetrics.RegressionMetrics method*), 9
agreement_index() (*SeqMetrics.RegressionMetrics method*), 9
aic() (*SeqMetrics.RegressionMetrics method*), 10
aitchison() (*SeqMetrics.RegressionMetrics method*), 10
amemiya_adj_r2() (*SeqMetrics.RegressionMetrics method*), 10
amemiya_pred_criterion() (*SeqMetrics.RegressionMetrics method*), 10

B

balanced_accuracy() (*SeqMetrics.ClassificationMetrics method*), 23
bias() (*SeqMetrics.RegressionMetrics method*), 10
bic() (*SeqMetrics.RegressionMetrics method*), 10
brier_score() (*SeqMetrics.RegressionMetrics method*), 10

C

calculate_all() (*SeqMetrics.Metrics method*), 5
calculate_hydro_metrics() (*SeqMetrics.RegressionMetrics method*), 10
calculate_minimal() (*SeqMetrics.Metrics method*), 6
calculate_scale_dependent_metrics() (*SeqMetrics.Metrics method*), 6
calculate_scale_independent_metrics() (*SeqMetrics.Metrics method*), 7
centered_rms_dev() (*SeqMetrics.RegressionMetrics method*), 11
ClassificationMetrics (*class in SeqMetrics*), 21
composite_metrics() (*SeqMetrics.Metrics method*), 7
confusion_matrix() (*SeqMetrics.ClassificationMetrics method*), 23
corr_coeff() (*SeqMetrics.RegressionMetrics method*), 11
cosine_similarity() (*SeqMetrics.RegressionMetrics method*), 11
covariance() (*SeqMetrics.RegressionMetrics method*), 11
cronbach_alpha() (*SeqMetrics.RegressionMetrics method*), 11
cross_entropy() (*SeqMetrics.ClassificationMetrics method*), 23

D

decomposed_mse() (*SeqMetrics.RegressionMetrics method*), 11

E

error_rate() (*SeqMetrics.ClassificationMetrics method*), 23
euclid_distance() (*SeqMetrics.RegressionMetrics method*), 11
exp_var_score() (*SeqMetrics.RegressionMetrics method*), 11
expanded_uncertainty() (*SeqMetrics.RegressionMetrics method*), 11

F

f1_score() (*SeqMetrics.ClassificationMetrics method*), 23
f2_score() (*SeqMetrics.ClassificationMetrics method*), 24
false_discovery_rate() (*SeqMetrics.ClassificationMetrics method*), 24
false_negative_rate() (*SeqMetrics.ClassificationMetrics method*), 24
false_omission_rate() (*SeqMetrics.ClassificationMetrics method*), 24
false_positive_rate() (*SeqMetrics.ClassificationMetrics method*), 24

fdc_fhv() (*SeqMetrics.RegressionMetrics method*), 11
fdc_flv() (*SeqMetrics.RegressionMetrics method*), 12
fowlkes_mallows_index() (*SeqMetrics.ClassificationMetrics method*), 24

G

gmae() (*SeqMetrics.RegressionMetrics method*), 12
gmean_diff() (*SeqMetrics.RegressionMetrics method*), 12
gmrae() (*SeqMetrics.RegressionMetrics method*), 12

I

inrse() (*SeqMetrics.RegressionMetrics method*), 12
irmse() (*SeqMetrics.RegressionMetrics method*), 12

J

JS() (*SeqMetrics.RegressionMetrics method*), 9

K

kendaull_tau() (*SeqMetrics.RegressionMetrics method*), 12
kge() (*SeqMetrics.RegressionMetrics method*), 12
kge_bound() (*SeqMetrics.RegressionMetrics method*), 12
kge_mod() (*SeqMetrics.RegressionMetrics method*), 12
kge_np() (*SeqMetrics.RegressionMetrics method*), 12
kgenp_bound() (*SeqMetrics.RegressionMetrics method*), 13
kgeprime_c2m() (*SeqMetrics.RegressionMetrics method*), 13
kl_sym() (*SeqMetrics.RegressionMetrics method*), 13

L

lm_index() (*SeqMetrics.RegressionMetrics method*), 13
log1p_p (*SeqMetrics.Metrics property*), 7
log1p_t (*SeqMetrics.Metrics property*), 7
log_nse() (*SeqMetrics.RegressionMetrics method*), 13
log_p (*SeqMetrics.Metrics property*), 7
log_prob() (*SeqMetrics.RegressionMetrics method*), 13
log_t (*SeqMetrics.Metrics property*), 7

M

maape() (*SeqMetrics.RegressionMetrics method*), 13
mae() (*SeqMetrics.RegressionMetrics method*), 13
mapd() (*SeqMetrics.RegressionMetrics method*), 13
mape() (*SeqMetrics.RegressionMetrics method*), 13
mare() (*SeqMetrics.RegressionMetrics method*), 13
mase() (*SeqMetrics.RegressionMetrics method*), 13
mathews_corr_coeff() (*SeqMetrics.ClassificationMetrics method*), 24
max_error() (*SeqMetrics.RegressionMetrics method*), 14
mb_r() (*SeqMetrics.RegressionMetrics method*), 14
mbe() (*SeqMetrics.RegressionMetrics method*), 14
mbrae() (*SeqMetrics.RegressionMetrics method*), 14
mda() (*SeqMetrics.RegressionMetrics method*), 14
mdape() (*SeqMetrics.RegressionMetrics method*), 14
mde() (*SeqMetrics.RegressionMetrics method*), 14
mdrae() (*SeqMetrics.RegressionMetrics method*), 14
me() (*SeqMetrics.RegressionMetrics method*), 14
mean_bias_error() (*SeqMetrics.RegressionMetrics method*), 14
mean_gamma_deviance() (*SeqMetrics.RegressionMetrics method*), 15
mean_poisson_deviance() (*SeqMetrics.RegressionMetrics method*), 15
mean_var() (*SeqMetrics.RegressionMetrics method*), 15
med_seq_error() (*SeqMetrics.RegressionMetrics method*), 15
median_abs_error() (*SeqMetrics.RegressionMetrics method*), 15
Metrics (*class in SeqMetrics*), 5
mle() (*SeqMetrics.RegressionMetrics method*), 15
mod_agreement_index() (*SeqMetrics.RegressionMetrics method*), 15
mpe() (*SeqMetrics.RegressionMetrics method*), 15
mrae() (*SeqMetrics.RegressionMetrics method*), 15
mse() (*SeqMetrics.Metrics method*), 7
msle() (*SeqMetrics.RegressionMetrics method*), 15

N

negative_likelihood_ratio() (*SeqMetrics.ClassificationMetrics method*), 24
negative_predictive_value() (*SeqMetrics.ClassificationMetrics method*), 25
norm_ae() (*SeqMetrics.RegressionMetrics method*), 15
norm_ape() (*SeqMetrics.RegressionMetrics method*), 15
norm_euclid_distance() (*SeqMetrics.RegressionMetrics method*), 15
nrmse() (*SeqMetrics.RegressionMetrics method*), 15
nrmse_ipercentile() (*SeqMetrics.RegressionMetrics method*), 15
nrmse_mean() (*SeqMetrics.RegressionMetrics method*), 16
nrmse_range() (*SeqMetrics.RegressionMetrics method*), 16
nse() (*SeqMetrics.RegressionMetrics method*), 16
nse_alpha() (*SeqMetrics.RegressionMetrics method*), 16
nse_beta() (*SeqMetrics.RegressionMetrics method*), 16
nse_bound() (*SeqMetrics.RegressionMetrics method*), 16
nse_mod() (*SeqMetrics.RegressionMetrics method*), 16
nse_rel() (*SeqMetrics.RegressionMetrics method*), 17

P

pbias() (*SeqMetrics.RegressionMetrics method*), 17

percentage_metrics() (*SeqMetrics.Metrics method*), 7
plot_metrics (*class in SeqMetrics.utils*), 27
positive_likelihood_ratio() (*SeqMetrics.ClassificationMetrics method*), 25
precision() (*SeqMetrics.ClassificationMetrics method*), 25
prevalence_threshold() (*SeqMetrics.ClassificationMetrics method*), 25

R

r2() (*SeqMetrics.RegressionMetrics method*), 17
r2_score() (*SeqMetrics.RegressionMetrics method*), 17
rae() (*SeqMetrics.RegressionMetrics method*), 17
recall() (*SeqMetrics.ClassificationMetrics method*), 25
ref_agreement_index() (*SeqMetrics.RegressionMetrics method*), 17
RegressionMetrics (*class in SeqMetrics*), 9
rel_agreement_index() (*SeqMetrics.RegressionMetrics method*), 17
relative_metrics() (*SeqMetrics.Metrics method*), 7
relative_rmse() (*SeqMetrics.RegressionMetrics method*), 17
remove_neg (*SeqMetrics.Metrics property*), 7
remove_zero (*SeqMetrics.Metrics property*), 7
replace_inf (*SeqMetrics.Metrics property*), 7
replace_nan (*SeqMetrics.Metrics property*), 7
rmdspe() (*SeqMetrics.RegressionMetrics method*), 17
rmse() (*SeqMetrics.RegressionMetrics method*), 17
rmsle() (*SeqMetrics.RegressionMetrics method*), 17
rmspe() (*SeqMetrics.RegressionMetrics method*), 17
rmsse() (*SeqMetrics.RegressionMetrics method*), 18
rrse() (*SeqMetrics.RegressionMetrics method*), 18
rse() (*SeqMetrics.RegressionMetrics method*), 18
rsr() (*SeqMetrics.RegressionMetrics method*), 18

S

sa() (*SeqMetrics.RegressionMetrics method*), 18
sc() (*SeqMetrics.RegressionMetrics method*), 18
scale_dependent_metrics() (*SeqMetrics.Metrics method*), 7
sga() (*SeqMetrics.RegressionMetrics method*), 18
sid() (*SeqMetrics.RegressionMetrics method*), 18
skill_score_murphy() (*SeqMetrics.RegressionMetrics method*), 18
smape() (*SeqMetrics.RegressionMetrics method*), 18
smdape() (*SeqMetrics.RegressionMetrics method*), 18
spearmann_corr() (*SeqMetrics.RegressionMetrics method*), 19
specificity() (*SeqMetrics.ClassificationMetrics method*), 25
sse() (*SeqMetrics.RegressionMetrics method*), 19
stats() (*SeqMetrics.Metrics method*), 7
std_ratio() (*SeqMetrics.RegressionMetrics method*), 19

T

treat_values() (*SeqMetrics.Metrics method*), 7

U

umbrae() (*SeqMetrics.RegressionMetrics method*), 19

V

ve() (*SeqMetrics.RegressionMetrics method*), 19
volume_error() (*SeqMetrics.RegressionMetrics method*), 19

W

wape() (*SeqMetrics.RegressionMetrics method*), 19
watt_m() (*SeqMetrics.RegressionMetrics method*), 19
wmape() (*SeqMetrics.RegressionMetrics method*), 19

Y

youden_index() (*SeqMetrics.ClassificationMetrics method*), 26