

---

# **proplexity**

***Release 0.5.7***

**J. Komorniczak**

**Apr 22, 2024**



## GETTING STARTED

|  |           |
|--|-----------|
| <b>1 Quick start guide</b>               | <b>1</b>  |
| 1.1 Installation . . . . .               | 1         |
| 1.2 Minimal processing example . . . . . | 1         |
| <b>2 Class Imbalance Measures</b>        | <b>5</b>  |
| <b>3 Correlation Measures</b>            | <b>7</b>  |
| <b>4 Dimensionality Measures</b>         | <b>9</b>  |
| <b>5 Feature-based Measures</b>          | <b>11</b> |
| <b>6 Geometry Measures</b>               | <b>15</b> |
| <b>7 Linearity Measures</b>              | <b>17</b> |
| 7.1 Classification measures . . . . .    | 17        |
| 7.2 Regression measures . . . . .        | 18        |
| <b>8 Neighborhood Measures</b>           | <b>21</b> |
| <b>9 Network Measures</b>                | <b>25</b> |
| <b>10 Smoothness Measures</b>            | <b>27</b> |
| <b>11 Complexity Calculator</b>          | <b>29</b> |
| <b>12 About us</b>                       | <b>33</b> |
| <b>13 Citation policy</b>                | <b>35</b> |
| <b>14 Getting started</b>                | <b>37</b> |
| <b>15 API Documentation</b>              | <b>39</b> |
| <b>Python Module Index</b>               | <b>41</b> |
| <b>Index</b>                             | <b>43</b> |



## QUICK START GUIDE

### 1.1 Installation

To use the *proplexity* package, it will be absolutely useful to install it. Fortunately, it is available in the PyPI repository, so you may install it using *pip*:

```
pip install -U proplexity
```

To enable the possibility to modify the measures provided by *proplexity* or in case of necessity to expand it with functions that it does not yet include, it is also possible to install the module directly from the source code. If any modifications are introduced, they propagate to the module currently available to the environment:

```
git clone https://github.com/w4k2/proplexity.git
cd proplexity
make install
```

### 1.2 Minimal processing example

The *proplexity* module is imported in the standard Python fashion. At the same time, for the convenience of implementation, the authors recommend importing it under the *px* alias.

```
# Importing proplexity
import proplexity as px
```

The library is equipped with the *ComplexityCalculator* calculator, which serves as the basic tool for establishing metrics. The following code presents an example of the generation of a synthetic data set – typical for the *scikit-learn* module – and the determination of the value of measures by fitting the complexity model in accordance with the standard API adopted for *scikit-learn* estimators.

```
# Loading benchmark dataset from scikit-learn
from sklearn.datasets import load_breast_cancer
X, y = load_breast_cancer(return_X_y=True)

# Initialize CoplexityCalculator with default parametrization
cc = px.ComplexityCalculator()

# Fit model with data
cc.fit(X,y)
```

As the L1, L2 and L3 measures use the recommended *LinearSVC* implementation from the *svm* module of the *scikit-learn* package in their calculations, the warning “*ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.*” may occur. It is not a problem for the metric calculation – only indicating the lack of linear problem separability.

The complexity calculator object stores a list of all estimated measures that can be read by the model’s *complexity* attribute.

```
cc.complexity
```

```
>>> [0.227 0.064 0.000 0.478 0.012 0.225 0.070 0.042 0.043 0.296 0.084
>>>  0.025 0.178 0.912 0.741 0.268 0.569 0.053 0.002 0.033 0.047 0.122]
```

They appear in the list in the same order as the declarations of the used metrics, which can also be obtained from the hidden method *\_metrics()*.

```
cc._metrics()
```

```
>>> ['f1', 'f1v', 'f2', 'f3', 'f4', 'l1', 'l2', 'l3', 'n1', 'n2', 'n3',
>>>  'n4', 't1', 'lsc', 'density', 'clsCoef', 'hubs', 't2', 't3', 't4',
>>>  'c1', 'c2']
```

The problem difficulty score can also be obtained as a single scalar measure, which is the arithmetic mean of all measures used in the calculation.

```
cc.score()
```

```
>>> 0.203
```

The *proplexity* module, in addition to raw data output, also provides two standard representations of problem analysis. The first is a report in the form of a dictionary presenting the number of patterns (*n\_samples*), attributes (*n\_features*), classes (*classes*), their prior distribution (*prior\_probability*), average metric (*score*) and all member metrics (*complexities*), which can be obtained using the model’s *report()* method:

```
cc.report()
```

```
>>> {
>>>   'n_samples': 569,
>>>   'n_features': 30,
>>>   'n_classes': 2,
>>>   'classes': array([0, 1]),
>>>   'prior_probability': array([0.373, 0.627]),
>>>   'score': 0.214,
>>>   'complexities':
>>>   {
>>>     'f1': 0.227, 'f1v': 0.064, 'f2': 0.001, 'f3': 0.478, 'f4': 0.012,
>>>     'l1': 0.433, 'l2': 0.069, 'l3': 0.049, 'n1': 0.043, 'n2': 0.296,
>>>     'n3': 0.084, 'n4': 0.039, 't1': 0.178, 't2': 0.053, 't3': 0.002,
>>>     't4': 0.033, 'c1': 0.047, 'c2': 0.122,
>>>     'lsc': 0.912, 'density': 0.741, 'clsCoef': 0.268, 'hubs': 0.569
>>>   }
>>> }
```

The second form of reporting is a graph which, in the polar projection, collates all metrics, grouped into categories using color codes:

- *red* – feature based measures,
- *orange* – linearity measures,
- *yellow* – neighborhood measures,
- *green* – network measures,
- *teal* – dimensionality measures,
- *blue* – class imbalance measures.

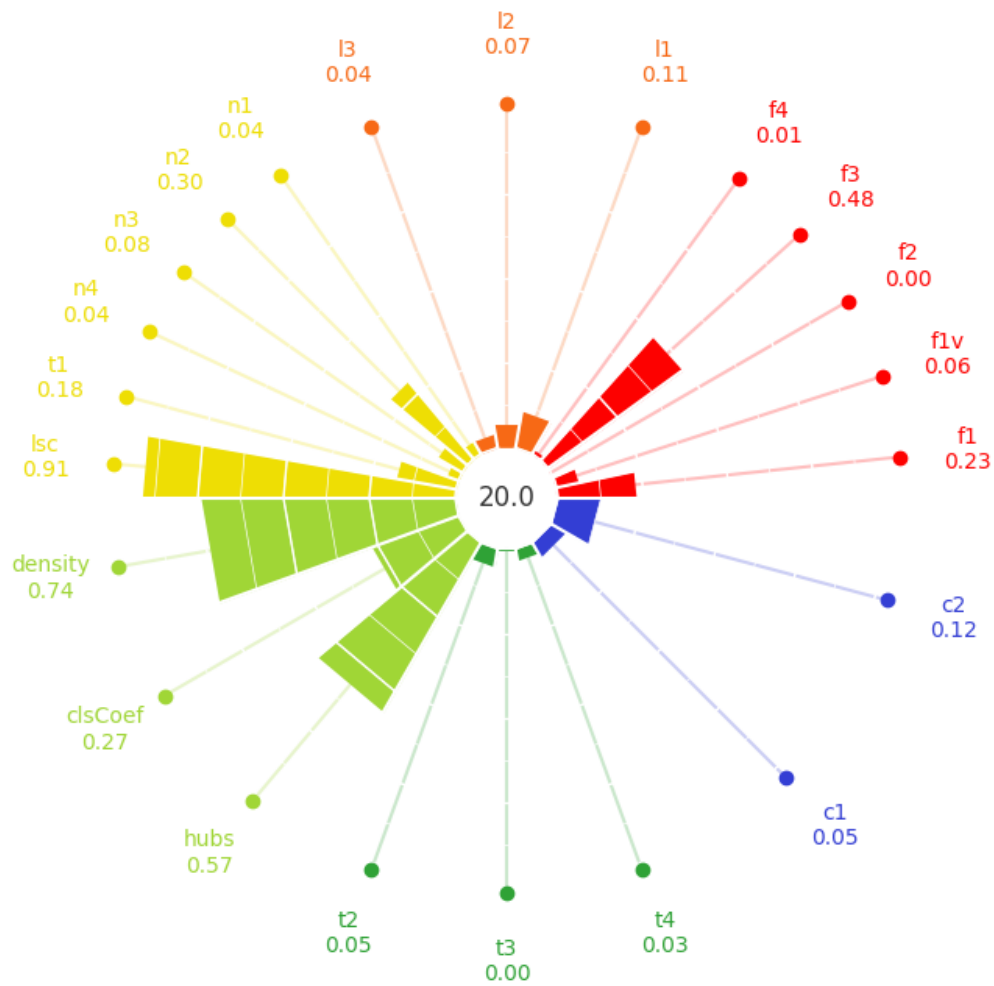
Each problem difficulty category occupies the same graph area, meaning that contexts that are less numerous in metrics (class imbalance) are not dominated in this presentation by categories described by many metrics (neighborhood). The illustration is built with the standard tools of the *matplotlib* module as a subplot of a figure and can be generated with the following source code.

```
# Import matplotlib
import matplotlib.pyplot as plt

# Prepare figure
fig = plt.figure(figsize=(7,7))

# Generate plot describing the dataset
cc.plot(fig, (1,1,1))
```

An example of a complexity graph is shown below.





## CLASS IMBALANCE MEASURES

|                       |  |
|-----------------------|--|
| <code>c1(X, y)</code> | Calculates the Entropy of Class Proportions (C1) metric. |
| <code>c2(X, y)</code> | Calculates the Imbalance Ratio (C2) metric.              |

`proplexity.classification.c1(X, y)`

Calculates the Entropy of Class Proportions (C1) metric.

$$C1 = 1 + \frac{1}{\log(n_c)} \sum_{i=1}^{n_c} p_{c_i} \log(p_{c_i})$$

### Parameters

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

### Return type

float

### Returns

C1 score

`proplexity.classification.c2(X, y)`

Calculates the Imbalance Ratio (C2) metric.

$$C2 = 1 - \frac{1}{IR}$$

### Parameters

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

### Return type

float

### Returns

C2 score



## CORRELATION MEASURES

|  |   |
|--|---|
| <code>c1(X, y[, normalize])</code>               | Calculates the maximum feature correlation to the output (C1) metric. |
| <code>c2(X, y[, normalize])</code>               | Calculates the average feature correlation to the output (C2) metric. |
| <code>c3(X, y[, is_optimized, normalize])</code> | Calculates the individual feature efficiency (C3) metric.             |
| <code>c4(X, y[, normalize])</code>               | Calculates the collective feature efficiency (C4) metric.             |

`proplexity.regression.c1(X, y, normalize=True)`

Calculates the maximum feature correlation to the output (C1) metric.

Measure returns maximum value out of all feature-output Spearman correlation absolute value. Higher values indicate simpler problems.

$$C1 = \max_{j=1,\dots,d} |\rho(x^j, y)|$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

C1 score

`proplexity.regression.c2(X, y, normalize=True)`

Calculates the average feature correlation to the output (C2) metric.

Measure returns average value of all feature-output Spearman correlation absolute value. Higher values indicate simpler problems.

$$C2 = \sum_{j=1}^d \frac{|\rho(x^j, y)|}{d}$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

C2 score

`proplexity.regression.c3(X, y, is_optimized=True, normalize=True)`

Calculates the individual feature efficiency (C3) metric.

Measure is calculated based on a number of examples that have to be removed in order to obtain a high correlation value. Removes samples based on residual value of linear regression model. The `is_optimized` flag value allows using optimized algorithm, based on divide and conquer strategy.

$$C3 = \min_{j=1}^d \frac{n^j}{n}$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

C3 score

`proplexity.regression.c4(X, y, normalize=True)`

Calculates the collective feature efficiency (C4) metric.

It sequentially analyzes the features with the greatest correlation to the output until all the features are used or all instances are removed. Samples with low residual value are removed. A metric is computed based on the number of samples remaining after removal procedure. By default, 0-1 interval normalization is used. The iterations limit of 1000 was introduced.

$$C4 = \frac{\#\{x_i || \epsilon_i| > 0.1\}_{T_l}}{n}$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

C4 score

## DIMENSIONALITY MEASURES

|                       |  |
|-----------------------|--|
| <code>t2(X, y)</code> | Calculates the Average number of features per dimension (T2) metric.             |
| <code>t3(X, y)</code> | Calculates the Average number of PCA dimensions per points (T3) metric.          |
| <code>t4(X, y)</code> | Calculates the Ratio of the PCA dimension to the original dimension (T4) metric. |

`problexity.classification.t2(X, y)`

Calculates the Average number of features per dimension (T2) metric.

To obtain this measure, the number of dimensions describing the dataset is divided by the number of instances.

$$T2 = \frac{m}{n}$$

### Parameters

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

### Return type

float

### Returns

T2 score

`problexity.classification.t3(X, y)`

Calculates the Average number of PCA dimensions per points (T3) metric.

To obtain this measure, first, the number of PCA components needed to represent 95% of data variability is calculated. Then, the value is divided by the instance number in the dataset.

$$T3 = \frac{m'}{n}$$

### Parameters

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

### Return type

float

### Returns

T3 score

`proplexity.classification.t4(X, y)`

Calculates the Ratio of the PCA dimension to the original dimension (T4) metric.

To obtain this measure, the number of PCA components needed to represent 95% of data variability is divided by the original number of dimensions. This measure describes the proportion of relevant dimensions in the dataset.

$$T4 = \frac{m'}{m}$$

**Parameters**

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

T4 score

## FEATURE-BASED MEASURES

|                   |   |
|-------------------|---|
| <i>f1</i> (X, y)  | Calculates the Maximum Fisher's discriminant ratio (F1) metric.                     |
| <i>f1v</i> (X, y) | Calculates the Directional vector maximum Fisher's discriminant ratio (F1v) metric. |
| <i>f2</i> (X, y)  | Calculates the Volume of overlapping region (F2) metric.                            |
| <i>f3</i> (X, y)  | Calculates the Maximum individual feature efficiency (F3) metric.                   |
| <i>f4</i> (X, y)  | Calculates the Collective feature efficiency (F4) metric.                           |

`proplexity.classification.f1(X, y)`

Calculates the Maximum Fisher's discriminant ratio (F1) metric.

Measure describes the overlap of feature values in each class.

$$F1 = \frac{1}{1 + \max_{i=1}^m r_{f_i}}$$

#### Parameters

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

#### Return type

float

#### Returns

F1 score

`proplexity.classification.f1v(X, y)`

Calculates the Directional vector maximum Fisher's discriminant ratio (F1v) metric.

$$F1v = \frac{1}{1 + dF}$$

#### Parameters

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

#### Return type

float

### Returns

F1v score

`proplexity.classification.f2(X, y)`

Calculates the Volume of overlapping region (F2) metric.

Describes the overlap of the feature values within the classes. The measure is determined by the minimum and maximum values of features in the class. The overlap is then calculated and normalized by the range of values in each class.

$$F2 = \prod_i^m \frac{\text{overlap}(f_i)}{\text{range}(f_i)}$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

F2 score

`proplexity.classification.f3(X, y)`

Calculates the Maximum individual feature efficiency (F3) metric.

Measure describes the efficiency of each feature in the separation of classes. It considers the maximum value among all features.

$$F3 = \min_{i=1}^m \frac{n_o(f_i)}{n}$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

F3 score

`proplexity.classification.f4(X, y)`

Calculates the Collective feature efficiency (F4) metric.

The measure describes an overview of how the features work together. The instances separated by the most discriminant feature that was not used already are excluded from the further analysis. The process continues until all instances are classified or all features are used. The measure is calculated according to the number of instances in the overlapping region after all features were used and the total number of samples.

$$F4 = \frac{n_o(f_{\min}(T_l))}{n}$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels



**Return type**

float

**Returns**

F4 score



## GEOMETRY MEASURES

|                                    |  |
|------------------------------------|--|
| <code>l3(X, y[, normalize])</code> | Calculates the non-linearity of a linear regressor (L3) measure.           |
| <code>s4(X, y[, normalize])</code> | Calculates the non-linearity of a nearest neighbor regressor (S4) measure. |
| <code>t2(X, y)</code>              | Calculates the average number of examples per dimension (T2) measure.      |

`proplexity.regression.l3(X, y, normalize=True)`

Calculates the non-linearity of a linear regressor (L3) measure.

Linearly interpolates both input (X) and output (y) values of each pair of samples with similar output values. Generated  $l=n-1$  synthetic samples and then measures the mean squared error of a linear regressor, fitted with original data and evaluated on synthetic points. By default performs a normalization of samples.

$$L3 = \frac{1}{l} \sum_{i=1}^l (f(x'_i) - y'_i)^2$$

#### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

#### Return type

float

#### Returns

L3 score

`proplexity.regression.s4(X, y, normalize=True)`

Calculates the non-linearity of a nearest neighbor regressor (S4) measure.

Linearly interpolates both input (X) and output (y) values of each pair of samples with similar output values. Generated  $l=n-1$  synthetic samples and then measures the mean squared error of a nearest neighbor regressor, fitted with original data and evaluated on synthetic points. By default performs a normalization of samples.

$$S4 = \frac{1}{l} \sum_{i=1}^l (NN(x'_i) - y'_i)^2$$

#### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset

- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

S4 score

`proplexity.regression.t2(X, y)`

Calculates the average number of examples per dimension (T2) measure.

Returns number of samples per number of features. Higher values indicate simpler problems.

$$T2 = \frac{n}{d}$$

**Parameters**

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

T2 score

## LINEARITY MEASURES

### 7.1 Classification measures

|                 |   |
|-----------------|---|
| <i>l1(X, y)</i> | Calculates the Sum of the error distance by linear programming (L1) metric. |
| <i>l2(X, y)</i> | Calculates the Error rate of linear classifier (L2) metric.                 |
| <i>l3(X, y)</i> | Calculates the Non linearity of linear classifier (L3) metric.              |

`proplexity.classification.l1(X, y)`

Calculates the Sum of the error distance by linear programming (L1) metric.

Uses Linear SVM classifier. Measures distance of incorrectly classified samples from the SVM hyperplane.

$$L1 = \frac{SumErrorDist}{1 + SumErrorDist}$$

**Parameters**

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

**Return type**

float

**Returns**

L1 score

`proplexity.classification.l2(X, y)`

Calculates the Error rate of linear classifier (L2) metric.

Returns error rate of Linear SVM classifier used within the dataset.

$$L2 = \frac{\sum_{i=1}^n I(h(x_i) \neq y_i)}{n}$$

**Parameters**

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

**Return type**

float

### Returns

L2 score

`proplexity.classification.l3(X, y)`

Calculates the Non linearity of linear classifier (L3) metric.

Linearly interpolating instances of each class generate the additional instances of the problem, which are used to calculate this measure. The class of original instances determines the label of an augmented point. The Linear SVM classifier is used to classify the synthesized points of the dataset. The number of synthetic points is equal to the original dataset size.

$$L3 = \frac{1}{l} \sum_{i=1}^l I(h_T(x'_i) \neq y'_i)$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

L3 score

## 7.2 Regression measures

|                                    |   |
|------------------------------------|---|
| <code>l1(X, y[, normalize])</code> | Calculates the mean absolute error (L1) metric. |
| <code>l2(X, y[, normalize])</code> | Calculates the residuals variance (L2) metric.  |

`proplexity.regression.l1(X, y, normalize=True)`

Calculates the mean absolute error (L1) metric.

Measure returns average error of linear regression model. By default performs a 0-1 interval normalization.

$$L1 = \sum_{i=1}^n \frac{|\epsilon_i|}{n}$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

### Return type

float

### Returns

L1 score

`proplexity.regression.l2(X, y, normalize=True)`

Calculates the residuals variance (L2) metric.

Measure returns average of squared residuals of linear regression model. By default performs a 0-1 interval normalization.

$$L2 = \sum_{i=1}^n \frac{\epsilon_i^2}{n}$$

**Parameters**

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type***float***Returns**

L2 score





## NEIGHBORHOOD MEASURES

|                        |  |
|------------------------|--|
| <code>n1(X, y)</code>  | Calculates the Fraction of borderline points (N1) metric.          |
| <code>n2(X, y)</code>  | Calculates the Ratio of intra/extra class NN distance (N2) metric. |
| <code>n3(X, y)</code>  | Calculates the Error rate of NN classifier (N4) metric.            |
| <code>n4(X, y)</code>  | Calculates the Nonlinearity of NN classifier (N4) metric.          |
| <code>t1(X, y)</code>  | Calculates the Fraction of hyperspheres covering data (T1) metric. |
| <code>lsc(X, y)</code> | Calculates the Local set average cardinality (LSC) metric.         |

`proplexity.classification.lsc(X, y)`

Calculates the Local set average cardinality (LSC) metric.

The measure is dependent on the distances between instances and the distances to the instances' nearest enemies – the nearest sample of the opposite class. The number of cases that lie closer to the sample than its closest enemy is taken into account during calculation.

$$LSC = 1 - \frac{1}{n_2} \sum_{i=1}^n |LS(x_i)|$$

### Parameters

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

### Return type

float

### Returns

LSC score

`proplexity.classification.n1(X, y)`

Calculates the Fraction of borderline points (N1) metric.

The Minimum Spanning Tree is generated over input instances. The measure is computed by calculating the number of edges in the MST between instances of different classes over a total number of samples.

$$N1 = \frac{1}{n} \sum_{i=1}^n I((x_i, x_j) \in MST \wedge y_i \neq y_j)$$

### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

**Return type**

float

**Returns**

N1 score

`proplexity.classification.n2(X, y)`

Calculates the Ratio of intra/extra class NN distance (N2) metric.

The measure depends on the distances of each problem instance to its nearest neighbor of the same class and the distance to the nearest neighbor of a different class. According to the proportions of those values, the final measure is calculated.

$$N2 = \frac{infra\_extra}{1 + infra\_extra}$$

**Parameters**

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

**Return type**

float

**Returns**

N2 score

`proplexity.classification.n3(X, y)`

Calculates the Error rate of NN classifier (N4) metric.

Measure is determined by the error rate of the One Nearest Neighbor Classifier in the Leave One Out evaluation protocol.

$$N3 = \frac{\sum_{i=1}^n I(NN(x_i) \neq y_i)}{n}$$

**Parameters**

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

**Return type**

float

**Returns**

N3 score

`proplexity.classification.n4(X, y)`

Calculates the Nonlinearity of NN classifier (N4) metric.

The measure is determined by the error rate of k - Nearest Neighbor Classifier on synthetic points, generated by linearly interpolating original instances. The Classifier is fitted on original points and evaluated on synthetic instances.

$$N4 = \frac{1}{l} \sum_{i=1}^l I(NN_T(x'_i) \neq y'_i)$$

**Parameters**

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

N4 score

proplexity.classification.t1(X, y)

Calculates the Fraction of hyperspheres covering data (T1) metric.

The measure is described by the number of hyperspheres needed to cover the data divided by a number of instances. First, a hypersphere is generated for each problem sample. A sample lies in the center of the hypersphere. Its radius is dependent on the distance to the instance of another class. The hyperspheres are eliminated if a different hypersphere already covers the center instance. The elimination starts from the hyperspheres with the largest radiuses and continues to the ones with smaller radiuses. The hyperspheres that were not eliminated are taken into account during the calculation of complexity.

$$T1 = \frac{\#Hyperspheres(T)}{n}$$

**Parameters**

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

T1 score



## NETWORK MEASURES

|                       |   |
|-----------------------|---|
| <i>density</i> (X, y) | Calculates the Density metric.                |
| <i>clsCoef</i> (X, y) | Calculates the Clustering Coefficient metric. |
| <i>hubs</i> (X, y)    | Calculates the Hubs metric.                   |

`proplexity.classification.clsCoef(X, y)`

Calculates the Clustering Coefficient metric.

Generates an epsilon-Nearest Neighbours graph. The epsilon value is set to 0.15. The edges are selected based on the Gower distance between samples, normalized to the range between 0 and 1. Edges between instances of distinct classes are removed. The neighborhood of each vertex is calculated – the instances directly connected to it. Then, the number of edges between the sample’s neighbors is calculated and divided by the maximum possible number of edges between them. The final measure is calculated based on the neighborhood of each point.

$$ClsCoef = 1 - \frac{1}{n} \sum_{i=1}^n \frac{2|e_{jk} : v_j, v_k \in N_i|}{k_i(k_i - 1)}$$

#### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

#### Return type

float

#### Returns

Clustering Coefficient score

`proplexity.classification.density(X, y)`

Calculates the Density metric.

Generates an epsilon-Nearest Neighbours graph. The epsilon value is set to 0.15. The edges are selected based on the Gower distance between samples, normalized to the range between 0 and 1. Edges between instances of distinct classes are removed. The measure calculates the number of edges in the final graph divided by the total possible number of edges.

$$Density = 1 - \frac{2|E|}{n(n-1)}$$

#### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset

- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

Density score

`proplexity.classification.hubs(X, y)`

Calculates the Hubs metric.

Generates an epsilon-Nearest Neighbours graph. The epsilon value is set to 0.15. The edges are selected based on the Gower distance between samples, normalized to the range between 0 and 1. Edges between instances of distinct classes are removed. The neighborhood of each vertex is obtained – the instances directly connected to it. The measure scores each sample by the number of connections to neighbors, weighted by the number of connections the neighbors have.

$$Hubs = 1 - \frac{1}{n} \sum_{i=1}^n hub(v_i)$$

**Parameters**

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

Hubs score

## SMOOTHNESS MEASURES

|                                    |  |
|------------------------------------|--|
| <code>s1(X, y[, normalize])</code> | Calculates the output distribution (S1) measure.                 |
| <code>s2(X, y[, normalize])</code> | Calculates the input distribution (S2) measure.                  |
| <code>s3(X, y[, normalize])</code> | Calculates the error of nearest neighbor regressor (S3) measure. |

`problexity.regression.s1(X, y, normalize=True)`

Calculates the output distribution (S1) measure.

Calculates complexity based on a similarity of instances adjacent in minimum spanning tree (MST). Returns the average difference of labels (y), of samples connected by MST. By default a 0-1 interval normalization is performed.

$$S1 = \frac{1}{n} \sum_{i,j \in MST} |y_i - y_j|$$

#### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

#### Return type

float

#### Returns

S1 score

`problexity.regression.s2(X, y, normalize=True)`

Calculates the input distribution (S2) measure.

Calculates complexity based on a similarity of features (X) of instances with close output values (y). Returns the average euclidean norm of difference of input values, of samples neighbouring after sorting them by output values. By default a 0-1 interval normalization is performed.

$$S2 = \frac{1}{n} \sum_{i=2}^n ||x_i - x_{i-1}||_2$$

#### Parameters

- **X** (array-like, shape (n\_samples, n\_features)) – Dataset
- **y** (array-like, shape (n\_samples)) – Labels

#### Return type

float

**Returns**

S2 score

`proplexity.regression.s3(X, y, normalize=True)`

Calculates the error of nearest neighbor regressor (S3) measure.

Returns mean squared error of a 1-nearest neighbor regressor, established during leave-one-out procedure. By default, the data is normalized with 0-1 interval normalization.

$$S3 = \frac{1}{n} \sum_{i=1}^n (NN(x_i) - y_i)^2$$

**Parameters**

- **X** (*array-like, shape (n\_samples, n\_features)*) – Dataset
- **y** (*array-like, shape (n\_samples)*) – Labels

**Return type**

float

**Returns**

S3 score



## COMPLEXITY CALCULATOR

---

|   |                              |
|---|------------------------------|
| <code>ComplexityCalculator([metrics, colors, ...])</code> | Complexity Calculator Class. |
|---|------------------------------|

---

```
class problemxity.ComplexityCalculator(metrics=None, colors=None, ranges=None, weights=None,
                                       mode='classification', multiclass_strategy='ovo')
```

Bases: `object`

Complexity Calculator Class.

A class that allows to determine all or selected metrics for a given data set. The report can be returned both as a simple vector of metrics, as well as a dictionary containing all set parameters and visualization in the form of a radar.

### Parameters

- **metrics** (*list, optional (default=all the metrics available in problemxity)*) – List of classification complexity measures used to validate a given set.
- **mode** (*string, optional (default=classification)*) – Recognition task for which metrics should be calculated. Might be selected between *classification* and *regression*.
- **multiclass\_strategy** (*string, optional (default=ova)*) – Strategy used for multiclass metric integration. Might be selected between *ova* and *ovo*.
- **ranges** (*dict, optional (default=all the default six groups of metrics)*) – Configuration of radar visualisation, allowing to group metrics by color.
- **colors** (*list, optional (default=six-color palette)*) – List of colors assigned to groups on radar visualisation.
- **weights** (*list, optional (default=list of weights, where weight are equal to 1 for all measures where simpler problems have smaller value, otherwise -1)*) – List of weights taken into account in score() procedure.

### Variables

- **complexity** (*list*) – The list of all the scores acquired with metrics defined by metrics list.
- **n\_samples** (*int*) – The number of samples in the fitted dataset.
- **n\_features** (*int*) – The number of features of the fitted dataset.
- **n\_classes** (*int*) – The number of classes in the fitted dataset.
- **classes** (*array-like, shape (n\_classes, )*) – The class labels.

- **prior\_probability**(array-like, shape (n\_classes, )) – The prior probability of classes.

### Examples

```
>>> from proplexity import ComplexityCalculator
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification()
>>> cc = ComplexityCalculator().fit(X, y)
>>> print(cc.complexity)
[0.3158144010174404, 0.1508882806154997, 0.005974480517635054, 0.57, 0.0,
 0.10518058962953956, 0.1, 0.07, 0.135, 0.48305940839428635, 0.27, 0.11,
 1.0, 0.9642, 0.9892929292929293, 0.9321428571428572, 0.9297111755529109,
 0.2, 0.16, 0.8, 0.0, 0.0]
>>> report = cc.report()
>>> print(report)
{
  'n_samples': 100, 'n_features': 20, 'n_classes': 2,
  'classes': array([0, 1]),
  'prior_probability': array([0.5, 0.5]),
  'score': 0.377,
  'complexities':
  {
    'f1': 0.316, 'f1v': 0.151, 'f2': 0.006, 'f3': 0.57, 'f4': 0.0,
    'l1': 0.105, 'l2': 0.1, 'l3': 0.07,
    'n1': 0.135, 'n2': 0.483, 'n3': 0.27, 'n4': 0.11, 't1': 1.0, 'lsc': 0.964,
    'density': 0.989, 'clsCoef': 0.932, 'hubs': 0.93,
    't2': 0.2, 't3': 0.16, 't4': 0.8, 'c1': 0.0, 'c2': 0.0
  }
}
```

### **fit**(X, y)

Calculates metrics for given dataset.

#### Parameters

- **X**(array-like, shape (n\_samples, n\_features)) – The training input samples.
- **y**(array-like, shape (n\_samples, )) – The training input labels.

#### Return type

ComplexityCalculator class object

#### Returns

ComplexityCalculator class object.

### **plot**(figure, spec=(1, 1, 1))

Returns integrated score of problem complexity

#### Parameters

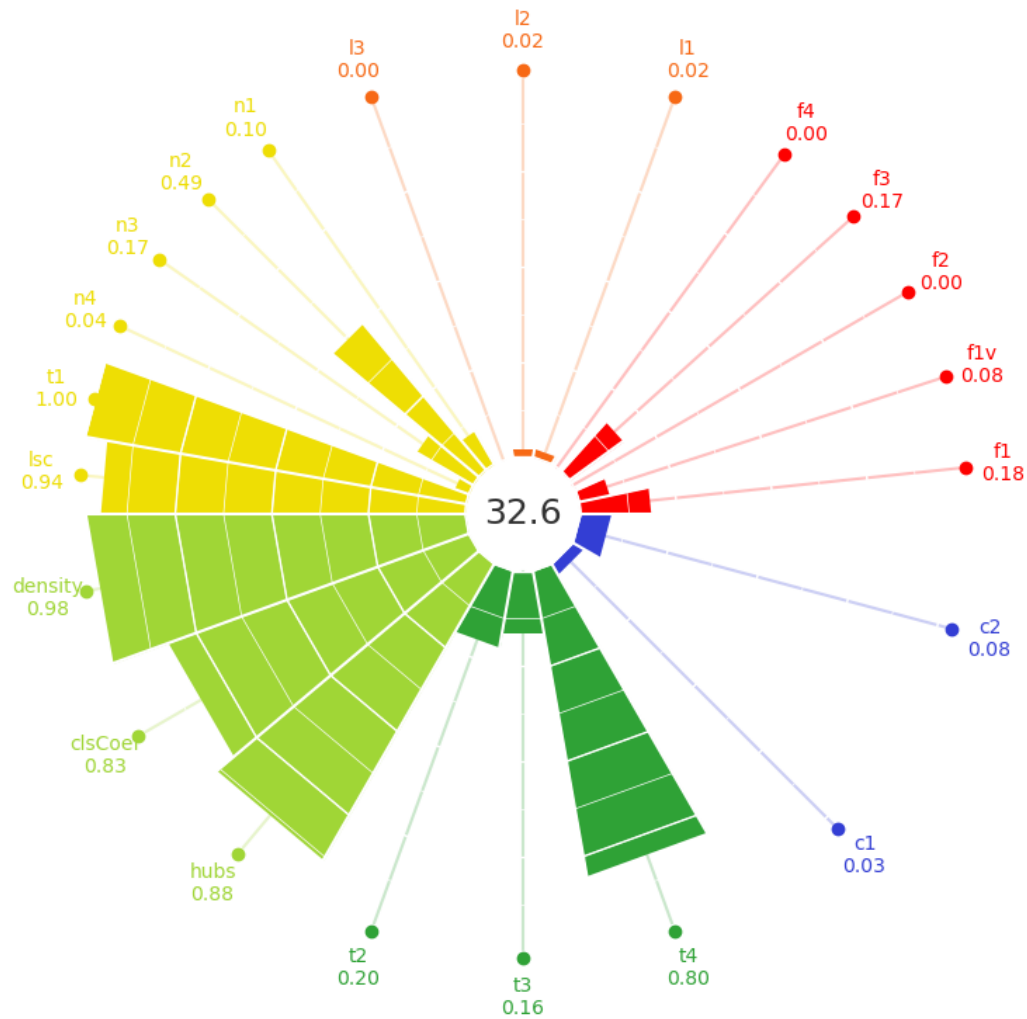
- **weights**(matplotlib figure object) – Figure to draw radar on.
- **spec**(tuple, optional (default=(1, 1, 1))) – Matplotlib subplot location.

#### Return type

object

#### Returns

Matplotlib axis object.



**report**(*precision=3*)

Returns report of problem complexity

**Parameters**

**precision** (*int*, *optional* (*default=3*)) – The rounding precision.

**Return type**

dict

**Returns**

Dictionary with complexity report

**Examples**

```
>>> from proplexity import ComplexityCalculator
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification()
>>> cc = ComplexityCalculator().fit(X, y)
>>> report = cc.report()
```

(continues on next page)

(continued from previous page)

```
>>> print(report)
{
  'n_samples': 100, 'n_features': 20, 'n_classes': 2,
  'classes': array([0, 1]),
  'prior_probability': array([0.5, 0.5]),
  'score': 0.377,
  'complexities':
  {
    'f1': 0.316, 'f1v': 0.151, 'f2': 0.006, 'f3': 0.57, 'f4': 0.0,
    'l1': 0.105, 'l2': 0.1, 'l3': 0.07,
    'n1': 0.135, 'n2': 0.483, 'n3': 0.27, 'n4': 0.11, 't1': 1.0, 'lsc': 0.
↪964,
    'density': 0.989, 'clsCoef': 0.932, 'hubs': 0.93,
    't2': 0.2, 't3': 0.16, 't4': 0.8, 'c1': 0.0, 'c2': 0.0
  }
}
```

### score()

Returns integrated score of problem complexity

#### Parameters

**weights** (array-like, optional (default=None), shape (n\_metrics)) – Optional weights of metrics.

#### Return type

float

#### Returns

Single score for integrated metrics

## ABOUT US

The **proplexity** package was created for the needs of the [Department of Systems and Computer Networks](#), *Wrocław University of Science and Technology*, as part of research projects regarding Multiobjective Optimization and its code is used for experimental evaluation since 2022.

The main author and maintainer of its current version is the employees of the unit, namely [J. Komorniczak](#).





## CITATION POLICY

If you use `proplexity` in a scientific publication, We would appreciate citation to the following papers, including introduction of library and original introduction of used measures:

```
@article{komorniczak2023proplexity,  
title={proplexity-An open-source Python library for supervised learning problem ↵  
↵complexity assessment},  
author={Komorniczak, Joanna and Ksieniewicz, Pawe{\l}},  
journal={Neurocomputing},  
volume={521},  
pages={126--136},  
year={2023},  
publisher={Elsevier}  
}  
  
@article{lorena2018complex,  
title={How complex is your classification problem},  
author={Lorena, A and Garcia, L and Lehmann, Jens and Souto, M and Ho, T},  
journal={A survey on measuring classification complexity. arXiv},  
year={2018}  
}
```

The `proplexity` is a python library containing the implementation of metrics describing the complexity of the classification problem. The implementation was based on the publication of Lorena et al.

You can read more about it in the User Guide.





## GETTING STARTED

To use the *proplexity* package, it will be absolutely useful to install it. Fortunately, it is available in the PyPI repository, so you may install it using *pip*:

```
pip install -U proplexity
```

You can also install the module cloned from Github using the `setup.py` file if you have a strange, but perhaps legitimate need:

```
git clone https://github.com/w4k2/proplexity.git
cd proplexity
make install
```



## API DOCUMENTATION

Precise API description of all the classes and functions implemented in the module.

See the [README](#) for more information.



## PYTHON MODULE INDEX

### p

`proplexity`, [29](#)  
`proplexity.classification`, [25](#)  
`proplexity.regression`, [27](#)



## C

c1() (in module *proplexity.classification*), 5  
 c1() (in module *proplexity.regression*), 7  
 c2() (in module *proplexity.classification*), 5  
 c2() (in module *proplexity.regression*), 7  
 c3() (in module *proplexity.regression*), 8  
 c4() (in module *proplexity.regression*), 8  
 clsCoef() (in module *proplexity.classification*), 25  
 ComplexityCalculator (class in *proplexity*), 29

## D

density() (in module *proplexity.classification*), 25

## F

f1() (in module *proplexity.classification*), 11  
 f1v() (in module *proplexity.classification*), 11  
 f2() (in module *proplexity.classification*), 12  
 f3() (in module *proplexity.classification*), 12  
 f4() (in module *proplexity.classification*), 12  
 fit() (*proplexity.ComplexityCalculator* method), 30

## H

hubs() (in module *proplexity.classification*), 26

## L

l1() (in module *proplexity.classification*), 17  
 l1() (in module *proplexity.regression*), 18  
 l2() (in module *proplexity.classification*), 17  
 l2() (in module *proplexity.regression*), 18  
 l3() (in module *proplexity.classification*), 18  
 l3() (in module *proplexity.regression*), 15  
 lsc() (in module *proplexity.classification*), 21

## M

module  
     *proplexity*, 29  
     *proplexity.classification*, 5, 9, 11, 17, 21, 25  
     *proplexity.regression*, 7, 15, 18, 27

## N

n1() (in module *proplexity.classification*), 21

n2() (in module *proplexity.classification*), 22  
 n3() (in module *proplexity.classification*), 22  
 n4() (in module *proplexity.classification*), 22

## P

plot() (*proplexity.ComplexityCalculator* method), 30  
*proplexity*  
     module, 29  
*proplexity.classification*  
     module, 5, 9, 11, 17, 21, 25  
*proplexity.regression*  
     module, 7, 15, 18, 27

## R

report() (*proplexity.ComplexityCalculator* method), 31

## S

s1() (in module *proplexity.regression*), 27  
 s2() (in module *proplexity.regression*), 27  
 s3() (in module *proplexity.regression*), 28  
 s4() (in module *proplexity.regression*), 15  
 score() (*proplexity.ComplexityCalculator* method), 32

## T

t1() (in module *proplexity.classification*), 23  
 t2() (in module *proplexity.classification*), 9  
 t2() (in module *proplexity.regression*), 16  
 t3() (in module *proplexity.classification*), 9  
 t4() (in module *proplexity.classification*), 9