

# *proplexity* — an open-source *Python* library for binary classification problem complexity assessment

Joanna Komorniczak and Pawel Ksieniewicz

*Department of Systems and Computer Networks,  
Faculty of Information and Communication Technology,  
Wrocław University of Science and Technology,  
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland*

---

## Abstract

The classification problem’s complexity assessment is an essential element of many topics in the supervised learning domain. It plays a significant role in meta-learning – becoming the basis for determining meta-attributes or multi-criteria optimization – allowing the evaluation of the training set resampling without needing to rebuild the recognition model. The tools currently available for the academic community, which would enable the calculation of problem complexity measures, are available only as libraries of the *C++* and *R* languages. This paper describes the software module that allows for the estimation of 22 complexity measures for the *Python* language – compatible with the *scikit-learn* programming interface – allowing for the implementation of research using them in the most popular programming environment of the *machine learning* community.

*Keywords:* Problem complexity, Classification, *Python*

---

## 1. Motivation and significance

Proper evaluation of the algorithms and methods proposed for solving the pattern classification task, in accordance with good practices adopted in the machine learning research environment [1], requires extensive experiments performed on a large pool of appropriately diverse data sets [2]. In a typical approach to designing an experimental evaluation procedure, the usefulness of a selected group of problems is most often assessed by basic reporting of the sets dimensionality, the number of examples describing them, the

number of problem classes [3] and their prior distribution – in the case of non-uniformity of their representation [4]. However, it is necessary to remember that these are only simple measures, briefly describing the problem difficulty without giving the researcher sufficient insight into the actual complexity of the task. A true complexity of a problem is contained not only in the basic characteristics of problem space or class imbalance but also in problem-specific distribution, understood as its linearity, neighborhood characteristic, geometrical and topological complexity, and feature dependency [5].

The classification task is the main issue of supervised machine learning, finding its applications in almost every branch of life and science, starting from economics, through epidemiology and medicine, to machine vision and predictive maintenance systems [6]. Such a variety of undertaken problems leads to a plethora of various difficulties, expressed in a multiplicity of class-building clusters, a significant imbalance ratio, or significant class overlap, to enumerate a few. The considered complexity metrics demonstrate the potential for assessing the diversity of collected data sets which were precisely described and spanned over the taxonomy by Lorena et al. [7].

The computations of the problem complexity do not only find their application in the proper selection of benchmark datasets for the experimental evaluation. Their application in *meta-learning* solutions has been particularly popular in recent years [8], being one of the primary sources of meta-features for problem identification. Such an approach allows for the induction of task representations [9] and the automation of the deep neural networks structure configuration [10]. The measures of problem complexity are also used in geospatial data – in filtering the predictor noise [11] – or in the difficulty metrics dedicated to spectral data [12]. Preliminary works assessing the usefulness of such measures in data stream processing are also present in the literature [13].

All the above-mentioned examples concern research from the narrow span of the last few years, in which researchers carried out elements of problem complexity analysis with the use of two problem-complexity libraries currently available to the scientific community.

The first library – *DCoL* – was developed in 2010 by the team of *Universitat Ramon Llull* and *Bell Laboratories* [14] and is an implementation of 14 basic complexity metrics for the *C++* language. In 2017 its source code was

made available on the GitHub<sup>1</sup>.

A second library, *ECoL*, published as supporting software for Lorena et al. [7] publication, has been developed since September 2016 and is an implementation of 22 metrics in *R* language, currently being the most comprehensive solution for this type available to the scientific community. It reached its current version (0.3.0) in December 2020. Its entire version history is available in public GitHub repository<sup>2</sup>.

It is important to emphasize that in recent years, the *Python* programming language has started playing a much more significant role in the development of machine learning methods than any other experimental environment [15].

This publication presents a *proplexity* library, containing the implementation of 22 problem complexity measures divided to six categories: (i) feature-based, (ii) linearity, (iii) neighborhood, (iv) network, (v) dimensionality, and (vi) class imbalance as well as a *ComplexityCalculator* class, introducing additional utilities facilitating research and enabling simple expansion of the module with additional metrics. By proposing this library, new methods considering the classification complexity can be developed, which will further impact the evolution of machine learning algorithms both in the meta-learning field and in other research applications from recent years.

The measures of the problem complexity can be used as a substitute criterion in optimization tasks. We can expect that the quality of the classification will depend on the problem’s difficulty expressed in measures of complexity. Testing the classifier’s ability to recognize objects, often using cross-validation and induction algorithms with computational overhead significantly larger than measure computation, will be time consuming. We can potentially speed up the optimization process by problem complexity assessment as an alternative criterion.

## 2. Software description

This chapter contains the software description of the library. It will present the package structure, a minimal processing example, and an exemplary analysis of the results. The chapter will illustrate the implemented measures of problem complexity and the *ComplexityCalculator* model.

---

<sup>1</sup><https://github.com/nmacia/dcol>

<sup>2</sup><https://github.com/lpfgarcia/ECoL/>

### 2.1. *Software Architecture*

The library consists of two main elements – the *measures* submodule and the *ComplexityCalculator* class. Within the measures, six categories are distinguished:

- *Feature-based* containing *F1*, *F1v*, *F2*, *F3*, and *F4* measures,
- *Linearity* containing *L1*, *L2*, and *L3* measures,
- *Neighborhood* with *N1*, *N2*, *N3*, *N4*, *T1*, and *LSC* measures,
- *Network* containing *density*, *ClsCoef*, and *Hubs* measures,
- *Dimensionality* containing *T2*, *T3*, and *T4* measures,
- *Class imbalance* containing *C1* and *C2* measures.

The *ComplexityCalculator* module enables the computation and analysis of data sets in the context of the difficulty of the classification task. It offers a set of methods that allows calculating the measure values and presenting the result as a single score, report, or illustrative graph.

### 2.2. *Measures*

The package divides measures into six categories, introduced in the publication by Lorena et al. [7]. The measures return 0 or a value close to 0 for simple problems and values close to 1 for complex problems. The only measures not limited to 1 are *T2* and *T3* from the *dimensionality* category. Four implemented measures (*L1*, *L2*, *L3*, *N4*) are non-deterministic; therefore, subsequent calculations on the same data can yield varying results. In the case of measures from the Linear category, this behavior results from Linear SVM classifier optimization, whose weights are initialized randomly. In the case of *L3* and *N4* measures, the generation of synthetic instances in a randomized manner makes the calculations non-deterministic.

#### 2.2.1. *Feature-based measures*

The measures describe the ability of features to separate classes in the classification problem. They analyze features separately or evaluate how attributes work together.

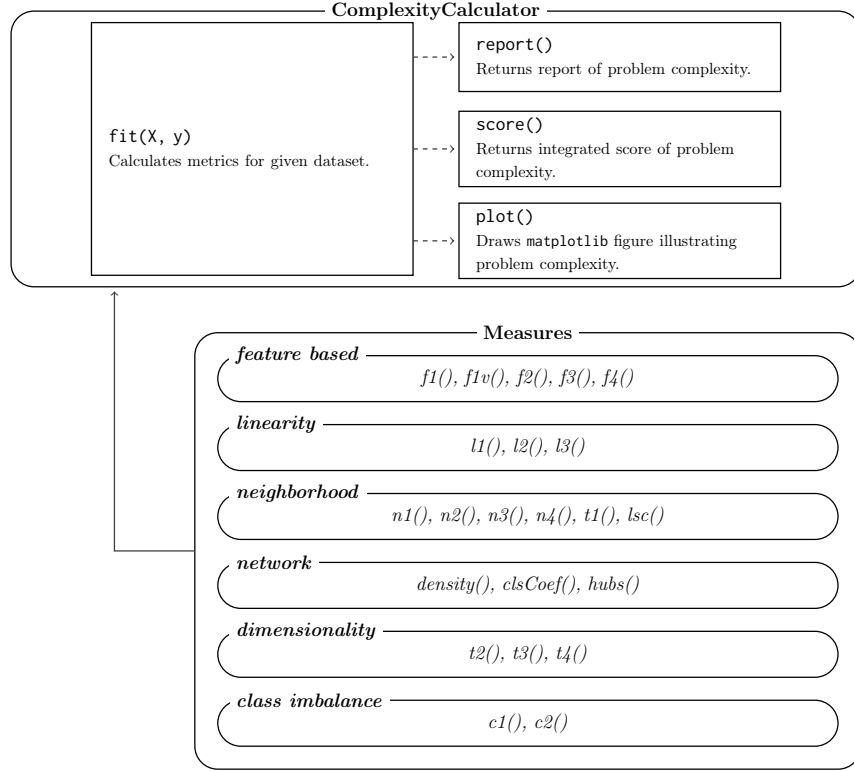


Figure 1: Overall schema of the software architecture

**F1 Maximum Fisher’s discriminant ratio.**

The measure describes the overlap of feature values in each class. The inverse of the original formulation, taking into account the most significant discriminant ratio, is taken into account, the same as in *ECOL* package.

**F1v Directional vector maximum Fisher’s discriminant ratio.**

The measure computes projection that maximizes class separation by directional Fisher’s criterion.

**F2 Volume of overlapping region**

The measure describes the overlap of the feature values within the classes. It is determined by the minimum and maximum values of

features in each class. The overlap is then calculated and normalized by the range of values in each class.

**F3 Maximum individual feature efficiency**

The measure describes the efficiency of each feature in the separation of classes. It considers the maximum value among all features. The equation proposed by Lorena et al. [7] has been slightly modified to obtain a maximum complexity value of 1 in case all instances of separate classes overlap.

**F4 Collective feature efficiency**

The measure describes the features synergy. The instances separated by the most discriminant attribute that was not used already are excluded from 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 and the total number of samples.

*2.2.2. Linearity measures*

The measures evaluate the level of problem class linear separation. The measures use Linear Support Vector Machines (SVM) classifier.

**L1 Sum of the error distance by linear programming**

The measure calculates the distance of incorrectly classified samples from the SVM hyperplane.

**L2 Error rate of linear classifier**

The measure is described by the error rate of the Linear SVM classifier within the dataset.

**L3 Non-linearity of linear classifier** The measure is described by the classifier's error rate on synthesized points of the dataset. The synthetic points are obtained by linearly interpolating instances of each class. The class of original examples determines the label of an augmented point, and the number of artificial points is equal to the original dataset size.

### 2.2.3. **Neighborhood measures**

The measures analyze the neighborhood of instances in a feature space. Neighbors of each sample are established based on the distance between problem instances.

#### **N1 Fraction of borderline points**

The *Minimum Spanning Tree* is generated over input instances in order to obtain this measure. The value is computed by calculating the number of edges in the MST between examples of different classes over a total number of samples.

#### **N2 Ratio of intra/extra class NN distance**

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 value is calculated.

#### **N3 Error rate of NN classifier**

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

#### **N4 Calculates the Non-linearity of NN classifier (N4) metric**

The measure is determined by the error rate of the 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.

#### **T1 Fraction of hyperspheres covering data**

The measure is defined 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 one already covers the center instance. The elimination starts from the hyperspheres with the largest radius and continues to the ones with a smaller radius. The hyperspheres that were not eliminated are taken into account during the calculation of complexity.

*LSC* **Local set average cardinality (LSC)**

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 considered during the calculation.

2.2.4. **Network measures**

The measures consider the instances as the vertices of the graph. All measures of this category generate an epsilon-Nearest Neighbours graph. The epsilon value is set to 0.15, same as in the *ECOL* package. The edges are selected based on the Gower distance between samples, normalized to the range between 0 and 1. The edge is placed between the points if a normalized Gower distance is smaller than 0.15. Edges between instances of distinct classes are removed.

*density* **Density metric**

The measure calculates the number of edges in the final graph divided by the total possible number of edges.

*clsCoef* **Clustering Coefficient metric**

For the purpose of obtaining this measure, the neighborhood of each vertex is calculated, i.e., 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 in the dataset.

*hubs* **Hubs metric**

For the purpose of obtaining this measure, the neighborhood of each vertex is obtained. The measure scores each sample by the number of connections to neighbors, weighted by the number of connections the neighbors have.

2.2.5. **Dimensionality measures**

The measures analyze the relation between the number of features and the number of instances in the dataset.



**T2 Average number of features per dimension**

For the purpose of obtaining this measure, the number of dimensions describing the dataset is divided by the number of instances.

**T3 Average number of PCA dimensions per points**

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.

**T4 Ration of the PCA dimension to the original dimension**

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.

**2.2.6. Class imbalance measures**

The Class Imbalance measures evaluate the dataset based on the degree of data imbalance.

**C1 Entropy of Class Proportions** The measure is obtained based on the proportion of each class's samples divided by the total number of samples.

**C2 Imbalance Ratio**

The measure is obtained based on the proportion of each class's samples divided by a number of opposite class samples.

**2.3. ComplexityCalculator**

The library introduces the *ComplexityCalculator* class to facilitate the use of measure implementation. Its objects are initialized with a list of measures and an optional list of (a) category colors for visualization purposes, and (b) a dictionary indicating the number of measures in a given category, which are necessary only in case of non-default collection of measures.

By default, the module will analyze all 22 metrics of the *measures* module. Executing the `fit()` method, which takes a set of features  $X$  and a set of labels  $y$  as an argument, will calculate the values of the metrics.

The obtained measures' values can be accessed as a single value using the `score()` method, optionally taking a vector of weights as a parameter. The

vector length has to correspond to the number of analyzed measures. By default, each measure has an equal weight, which means that executing the `score()` method will return the arithmetic mean of measured complexities.

The `report()` method provides a more detailed description of the classification problem. The method returns a dictionary containing a summary of each metric value, their arithmetic mean, and other data set characteristics, such as the number of samples, dimensionality, labels, and the number of classes with a prior probability.

The values can also be presented in the form of a graph. Executing the `plot` method returns a chart that illustrates the value of each measure from respective categories as well as the default score of the problem.

#### 2.4. Minimal processing example

The *proplexity* module is open Python software released under the *GPL-3.0* license and versioned in the public *Python Package Index* (PyPI) repository. Therefore, it can be easily obtained with the *pip* package installer with the command:

```
> pip install 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
```

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:

```
1 # Importing proplexity
2 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:

```

1 # Loading benchmark dataset from scikit-learn
2 from sklearn.datasets import load_breast_cancer
3 X, y = load_breast_cancer(return_X_y=True)
4
5 # Initialize CoplexityCalculator with default parametrization
6 cc = px.ComplexityCalculator()
7
8 # Fit model with data
9 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.*" might 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:

```
1 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()`:

```
1 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:

```
1 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:

```
1 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:

```
1 # Import matplotlib
2 import matplotlib.pyplot as plt
3
4 # Prepare figure
5 fig = plt.figure(figsize=(7,7))
6
7 # Generate plot describing the dataset
8 cc.plot(fig, (1,1,1))
```

An example of a complexity graph is shown in the Figure 2.

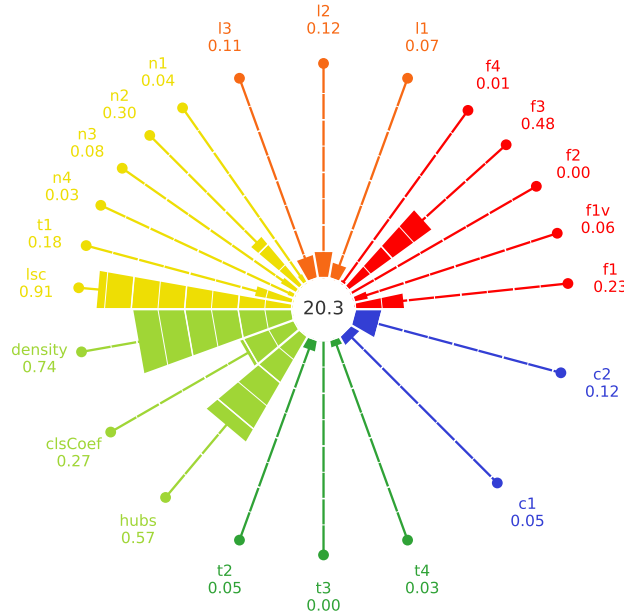


Figure 2: Exemplary complexity graph generated by *proplexity* module

### 3. Comparison with available modules

The juxtaposition in Table 1 presents a comparison of the available libraries analyzing the difficulty of classification problems. The columns successively present the *ECoL*, *DCoL*, and the *problexity* libraries. The rows show the categories of the complexity measures. The values in the cells indicate the number of available metrics compared to the number described in a publication by Lorena et al. [7].

Since *DCoL* was the earliest implemented library of the analyzed ones, it contains the fewest measures. The *ECoL* and *problexity* libraries are based on the same publication, so the number of measures in each category matches. The Table also shows the availability of utilities offered by the libraries and basic information describing them. All of the compared packages offer the possibility of generating a report containing a summary of the values of selected measures. In addition, the *problexity* package includes a method allowing to represent selected measures as a single value and as well contains a tool for graphically presenting the measures.

Table 1: Comparison of measures and utilities available in *ECoL*, *DCoL* and *problexity*

AREA	FUNCTIONALITY	<i>ECoL</i>	<i>DCoL</i>	<i>problexity</i>
<i>Measures</i>	Feature-based	5/5	5/5	5/5
	Linearity	3/3	3/3	3/3
	Neighborhood	5/5	5/5	5/5
	Network	3/3	0/3	3/3
	Dimensionality	3/3	1/3	3/3
	Class imbalance	2/2	0/2	2/2
<i>Utility Module</i>	Score			✓
	Report	✓	✓	✓
	Plot			✓
<i>Basic information</i>	Language	R	C++	Python
	Current version	0.3.0	—	0.3.2

### 4. Impact

In recent years, the complexity measures for classification problems have gained particular interest in the scientific community. Their most common use is the construction of meta-attributes (features describing sets [16]) to automate the selection of processing flows typical of the meta-learning topic [8].

An interesting trend here is, in particular, the construction of abstract representations of recognition tasks [9], which allow for an initial generalization of the problem under consideration [17], allowing for a significant reduction of the time necessary to select the optimal classification model for a given task [10].

An alternative field of use is the classification of difficult data, with particular emphasis on multidimensional discrete signals – for example – in the form of multispectral and geospatial problems [12]. On the one hand, the measures of complexity allow for agnostic estimation of the correct structure of the objects in the predictor’s action space [18]. On the other hand, they are tools useful in filtering its noise [11]. The same agnostic characteristics show a particular potential in the processing of imbalanced data [19], allowing the quality of the proposed resampling to be assessed without having to rebuild the recognition model [20].

As with imbalanced data, complexity measures also find their application in processing data streams [13]. By demonstrating the relationship between the quality of the recognition model and some currently available measures, it is possible to use them as a proxy-classifier, which allows for a significant reduction in the time of reviewing available solutions when using any optimization methods [21].

The field of classification problem complexity is still very active, not only in applications described above but also in the proposals of new measures that appear each year. The newly proposed measures allow the assessment of other processing contexts such as category learning [22], rule-based dissociations [23], or lost points identification [24], to enumerate a few.

## 5. Conclusions

This paper presents the *problexity* library for *Python* programming language. The library contains measures for assessing the complexity of binary classification problems. Twenty-two evaluation measures of the problem complexity have been implemented in the following categories: feature-based, linearity, neighborhood, network, dimensionality, and class imbalance. Additionally, the library incorporates the *ComplexityCalculator* module, which provides additional tools for analyzing classification data sets.

The library was created to fill the gap related to the lack of accessible measures for assessing the complexity of the classification problem in Python. Increasing the availability of complexity assessment methods and creating

a tool for exploring them will allow for a more detailed analysis of data sets' characteristics, potentially impacting the development of new machine learning methods.

The current version of the package offers a set of measures adapted to binary classification datasets, which are the most frequent objective of machine learning applications. The intent of future versions is for the library to include measures adapted to multiclass problems. As we pointed out in the impact section, in the field of data complexity evaluation, new measures are being proposed. The package will be maintained to contain other measures of classification complexity and possibly extended with regression complexity evaluation measures. Further works will focus on adapting the library to analyze data streams in the context of data difficulties. Finally, the *proplexity* library will continue to be used in research studies of the classification problem complexity employment.

## Acknowledgements

This work was supported by the Polish National Science Centre under the grant No. 2019/35/B/ST6/0442 as well as by the statutory funds of the Department of Systems and Computer Networks, Faculty of Information and Communication Technology, Wrocław University of Science and Technology.

## References

- [1] K. Stapor, P. Ksieniewicz, S. García, M. Woźniak, How to design the fair experimental classifier evaluation, *Applied Soft Computing* 104 (2021) 107219.
- [2] F. Hoffmann, T. Bertram, R. Mikut, M. Reischl, O. Nelles, Benchmarking in classification and regression, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9 (5) (2019) e1318.
- [3] J. M. Sotoca, J. Sánchez, R. A. Mollineda, A review of data complexity measures and their applicability to pattern classification problems, *Actas del III Taller Nacional de Minería de Datos y Aprendizaje. TAMIDA* (2005) 77–83.
- [4] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, F. Herrera, *Learning from imbalanced data sets*, Vol. 10, Springer, 2018.



- [5] T. K. Ho, M. Basu, Complexity measures of supervised classification problems, *IEEE transactions on pattern analysis and machine intelligence* 24 (3) (2002) 289–300.
- [6] A. A. Soofi, A. Awan, Classification techniques in machine learning: applications and issues, *Journal of Basic & Applied Sciences* 13 (2017) 459–465.
- [7] A. C. Lorena, L. P. Garcia, J. Lehmann, M. C. Souto, T. K. Ho, How complex is your classification problem? a survey on measuring classification complexity, *ACM Computing Surveys (CSUR)* 52 (5) (2019) 1–34.
- [8] J. Vanschoren, Meta-learning: A survey, *arXiv preprint arXiv:1810.03548*.
- [9] M. M. Meskhi, A. Rivolli, R. G. Mantovani, R. Vilalta, Learning abstract task representations, in: I. Guyon, J. N. van Rijn, S. Treguer, J. Vanschoren (Eds.), *AAAI Workshop on Meta-Learning and MetaDL Challenge*, Vol. 140 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 127–137.  
URL <https://proceedings.mlr.press/v140/meskhi21a.html>
- [10] E. Konuk, K. Smith, An empirical study of the relation between network architecture and complexity, in: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2019.
- [11] H. Guillon, C. F. Byrne, B. A. Lane, S. Sandoval Solis, G. B. Pasternack, Machine learning predicts reach-scale channel types from coarse-scale geospatial data in a large river basin, *Water Resources Research* 56 (3) (2020) e2019WR026691.
- [12] F. Branchaud-Charron, A. Achkar, P.-M. Jodoin, Spectral metric for dataset complexity assessment, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [13] M. Ellis, A. S. Bosman, A. P. Engelbrecht, Characterisation of environment type and difficulty for streamed data classification problems, *Information Sciences* 569 (2021) 615–649.

- [14] A. Orriols-Puig, N. Macia, T. K. Ho, Documentation for the data complexity library in c++, Universitat Ramon Llull, La Salle 196 (1-40) (2010) 12.
- [15] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. López García, I. Heredia, P. Malík, L. Hluchý, Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey, *Artificial Intelligence Review* 52 (1) (2019) 77–124.
- [16] A. Rivolli, L. P. Garcia, C. Soares, J. Vanschoren, A. C. de Carvalho, Characterizing classification datasets: a study of meta-features for meta-learning, *arXiv preprint arXiv:1808.10406*.
- [17] A. Rivolli, L. P. Garcia, C. Soares, J. Vanschoren, A. C. de Carvalho, Meta-features for meta-learning, *Knowledge-Based Systems* 240 (2022) 108101.
- [18] L. P. Garcia, A. C. de Carvalho, A. C. Lorena, Effect of label noise in the complexity of classification problems, *Neurocomputing* 160 (2015) 108–119.
- [19] D. Lee, K. Kim, An efficient method to determine sample size in over-sampling based on classification complexity for imbalanced data, *Expert Systems with Applications* 184 (2021) 115442.
- [20] V. H. Barella, L. P. Garcia, M. P. de Souto, A. C. Lorena, A. de Carvalho, Data complexity measures for imbalanced classification tasks, in: *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2018, pp. 1–8.
- [21] Z. Cai, Y. Long, L. Shao, Classification complexity assessment for hyperparameter optimization, *Pattern Recognition Letters* 125 (2019) 396–403.
- [22] L. A. Rosedahl, F. G. Ashby, A difficulty predictor for perceptual category learning, *Journal of Vision* 19 (6) (2019) 20–20.
- [23] F. G. Ashby, J. D. Smith, L. A. Rosedahl, Dissociations between rule-based and information-integration categorization are not caused by differences in task difficulty, *Memory & cognition* 48 (4) (2020) 541–552.

- [24] C. Lancho, I. Martín de Diego, M. Cuesta, V. Aceña, J. M Moguerza, A complexity measure for binary classification problems based on lost points, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2021, pp. 137–146.

## Required Metadata

### Current executable software version

Table 2: Code metadata (mandatory)

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.3.2
C2	Permanent link to code/repository used for this code version	<i>https : //github.com/w4k2/problexity</i>
C3	Legal Code License	GPL-3.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	python
C6	Compilation requirements, operating environments & dependencies	
C7	If available Link to developer documentation/manual	<i>https : //problexity.readthedocs.io</i>
C8	Support email for questions	<i>joanna.komorniczak@pwr.edu.pl</i>