PyClust: Building Meta-learning Repositories for Clustering

Yannis Poulakis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
gpoul@unipi.gr

Dimitris Petratos
Department of Digital Systems
University of Piraeus
Piraeus, Greece
dpetr@unipi.gr

Christos Doulkeridis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
cdoulk@unipi.gr

***Abstract*—Meta-learning is the principled process of leveraging knowledge from previously solved tasks to improve performance on future, similar ones. This process relies on meta-learning repositories that store dataset descriptors (meta-features) together with algorithm performance evaluations. In the context of clustering, however, there is still a limited availability of modern tools that support systematic meta-feature extraction and efficient evaluation, hindering the construction of rich meta-learning repositories. In this paper we present *PyClust* a toolkit that is used for populating efficiently meta-learning repositories for clustering. *PyClust* comprises of a standalone python library (*PyClustKit*, pypi: pyclustkit) and *PyClustUI*, a graphical tool for meta-learning. Each component supports different use cases for the machine-learning practitioner. *PyClustKit* supports the extraction of a rich set of meta-features which can be used to capture similarities at dataset level and implements many internal cluster validity indices (CVIs), which are widely used for evaluation of clustering results due to the absence of ground truth in unsupervised learning. *PyClustUI* is a graphical tool that integrates the capabilities of *PyClustKit* to enable end-to-end meta-learning repository creation and also integrates the necessary methods to support meta-learning model training for algorithm selection, one of the most popular meta-learning tasks.**

***Index Terms*—Clustering, meta-learning, algorithm selection, cluster validity indices**

I. Introduction

Clustering is a fundamental task in unsupervised learning, that is widely applied in customer segmentation, document topic modeling and product recommendation systems. Meta-learning, the process of leveraging insights from previously solved tasks to improve future similar ones, has potentially many downstream applications in clustering such as training meta-models for algorithm selection. Yet, despite the advancements in unsupervised learning, clustering meta-learning remains underdeveloped. The absence of modern tools for building repositories that capture the knowledge required to design meta-learning applications is one of the key barriers. Furthermore, the meta-learning repository population is challenging due to the inherent lack of ground truth in unsupervised applications. Practitioners need to calculate a large number of internal cluster validity indices (CVIs) to evaluate models, which becomes a increasingly costly as data grow in scale. In this paper, we present PyClust, a tool designed to help machine-learning practitioners build meta-learning repositories in a simple and efficient manner, that can be used for various downstream meta-learning tasks under the scope of clustering.

PyClust consists of two components, *PyClustKit* and *PyClustUI*. *PyClustKit* integrates several innovative features. First, it supports a rich set of meta-features that have been proposed in the literature, which can be readily computed for any given dataset. Second, it includes the implementation of a comprehensive collection of internal cluster validity indices (CVIs) that are used to evaluate different aspects of clustering quality [2]. Internal CVIs are particularly useful in unsupervised scenarios as they do not require prior knowledge about the clusters. Notably, *PyClustKit* implements 45 CVIs; In contrast, Scikit-learn [3], one of the most popular Python libraries for machine learning currently only provides 3 CVIs. Third, processing in *PyClustKit* is optimized, by reusing already computed intermediate results across different CVIs which greatly reduces computation times when multiple CVIs are to be calculated, which is typically the case in most meta-learning and even AutoML applications. These contributions enhance the library's functional breadth and facilitate meta-learning in the context of repository population.

Additionally, PyClust consists of *PyClustUI*, a graphical interface for users to orchestrate the end-to-end meta-learning repository-building workflow with extra capabilities for algorithm selection. We consider algorithm selection to be the most popular of the meta-learning tasks and as such is the first to be totally supported end-to-end through user-interface. Specifically, the main functionalities covered in *PyClustUI* are: (i) dataset ingestion, (ii) meta-feature extraction, (iii) execution of configurable clustering algorithms, (iv) computation of CVIs, (v) algorithm evaluation and finally (vi) meta-learner training and evaluation. For these reasons, we developed a low-code interface built upon Gradio [6]. The interface offers defaults, various plots and serves as an entry point lowering the barrier for practitioners who prefer a graphical workflow while still leveraging the benefits of *PyClustKit*.

The rest of this paper is structured as follows. In Section II, we provide the overview of the framework and go in detail about its components. In Section III, we present some use-cases we expect users to apply our framework. Finally, in Section IV, we conclude our work with some final remarks and future steps.

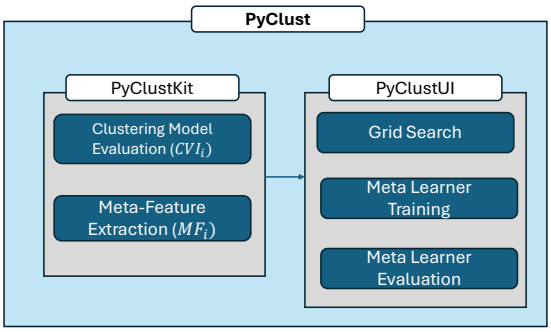

Fig. 1. The PyClust Components.

II. The PyClust Components

A. PyClustKit

PyClustKit is the low-level component that enables optimized meta-feature extraction CVI computation (Figure 1) and in a computationally efficient manner.

1) Meta-Features: For meta-features to be meaningful, they need to encompass information that is relevant with the task at hand. While various meta-features have been proposed in the literature [4], some already developed in python libraries [1], they mostly apply to supervised learning tasks, such as classification, and they are not necessarily suited for unsupervised learning tasks. In *PyClustKit*, we implement most of the meta-features listed in [1], which, to the best of our knowledge, represent a comprehensive list of meta-features originally suggested for clustering algorithm selection. This list includes statistical meta-features, such as the number of features and the ratio of real-valued to discrete features, as well as clustering-specific meta-features, such as statistics over the distribution of pairwise distances (mean, std, min/max, etc.). More precisely, we have implemented 9 descriptive, 20 statistical, 16 landmark, 3 similarity-based and 1 graph-based methods to extract meta-features. The complete list of meta-features implemented in *PyClustKit* can be found in project's GitHub repository.

2) Cluster Validity Indices: Evaluation for clustering is inherently difficult due to the unavailability of ground truth and is often based on internal indices that measure geometrical aspects of the clusters, such as inter- and intra-cluster distances and density [2]. In order to support a thorough evaluation of clustering results, we have developed the necessary methods to calculate 45 different internal CVIs. Although many of these validity indices have been proposed in the literature long before, we observed that many data science tools either provide only a limited number of CVIs (see Table I), or they are not optimized and as a result demonstrate suboptimal processing performance.

Towards this latter goal of computational efficiency, we have identified a set of 28 processes that are reused in the computation of different indices. To further elaborate, consider the Generalized Dunn indices [2] (18 in total) which are calculated by combining one of 3 different distance measurements for the numerator and one of 6 different distance measurements for the denominator called *deltas* and *Deltas*, respectively. To avoid repeating computations, we implement a process tracking mechanism that stores the output of these operations to be reused. We depict all the identified operations as a directed acyclic graph (DAG). Each time a CVI is to be computed, we identify the sub-graph that represents the required operations in a hierarchical order, identify if any of them have previously been executed and (if so) reuse the previously computed result. The complete list of CVIs implemented in *PyClustKit* can be found in project's GitHub repository.

[1]https://pymfe.readthedocs.io/

	Number of CVIs	Programming Language
PyClustKit	45	Python
Scikit-Learn [3]	3	Python
ClusterCrit [8]	42	R
ClustersFeatures [7]	40	Python

TABLE I
Number of CVIs implemented in different libraries

B. PyClustUI

PyClustUI builds on top of *PyClustKit* with a graphical interface enabling meta-feature extraction and CVI computation through a user-friendly interface. Additionally it encompasses extra processes that enable end-to-end meta-learning repository building and meta-learner training for algorithm selection (Figure 1). These are (a) configurable clustering algorithm training via Grid Search and (b) meta-learner training.

1) Meta-Learning: Meta-learning refers to the process of learning from previously solved machine learning tasks to improve the solution of similar future tasks. *PyClustUI* harnesses the capabilities in meta-feature extraction of *PyClustKit* and builds on top of it to provide other utilities, that should otherwise require various code snippets, to build meta-learning repositories and also train meta-learners for algorithm selection. Each dataset D_i gets associated with a vector of values $MF_i = \{m_1, \ldots, m_{|M|}\}$, called meta-features, that represent various properties of the dataset, which are likely to be important for the task at hand.

For each dataset D_i from a collection of datasets $D = \{D_1, \ldots, D_{|D|}\}$ that the user provides, this meta-feature vector is extracted and mapped with the best algoritmh $a^*_{l,CVI}$ for dataset D_i, found during grid search (cf. Section II-B2 - Equation 2). Afterwards, a meta-learner F can be trained over the meta-knowledge repository to learn to predict the best algorithm a^* for a dataset at hand $F(m_1, \ldots, m_{|M|}) \rightarrow a^*$. In this context, where the target label consists of discrete values, any classification algorithm can be trained as a meta-learner. In the current version, *PyClustUI* supports K-NN, for its simplicity and popularity, and decision trees for their capability of providing explainable results [5]. The meta-learner is then evaluated through accuracy and confusion matrix and can be added to the repository for later usage. Each trained meta-learner can be reused through a single interface where the user

(i) uploads the dataset, (ii) extracts meta-features (iii) selects the trained meta-learner to use and finally (iv) retrieves the prediction.

2) Parameter Search: To train a meta-learner, a repository of meta-information is required to be available relevant to the task. For algorithm selection, this repository should contain the meta-features of previously seen datasets and their corresponding best performing algorithm. Finding this algorithm can be achieved by evaluating (offline) a set of algorithm configurations. To this end, *PyClustUI* supports grid search for parameter search of different clustering algorithms.

Each implemented algorithm $a_l \in A = \{a_1, ..., a_{|A|}\}$, has a respective parametric space denoted as $\Lambda_{a_l} = \{\lambda_{a_l,p_j} : j = 1 \ldots, k\}$, where λ_{a_l,p_j} is the domain of parameter p_j and k denotes the number of parameters of algorithm a_l. Therefore, the whole search space Θ is the union of the Cartesian products of algorithm configurations, as in Equation 1. Continuous-valued domains of parameters are discretized accordingly.

$$\Theta = \bigcup_{l=1}^{|A|} (\{a_l\} \times \Lambda_{a_l}) \tag{1}$$

Hence, for each algorithm configuration, i.e., algorithm a_l and paramaterization λ_{a_l,p_j}, that produces a clustering result, we compute a set of CVIs, denoted as $C_{\{a_l\}\times\lambda_{a_l,p_j}} = \{c_1^{\lambda_{a_l,p_j}}, \ldots, c_{|C|}^{\lambda_{a_l,p_j}}\}$. Given a CVI $cvi \in \{1, \ldots, |C|\}$, upon which we want to rank algorithms $a_l \in A$, we define the best algorithm to be

$$a^*_{l,cvi} = \arg\max_{a_l \in A} \bigcup_{\lambda_{a_l,p_j} \in \Lambda_{a_l}} \{c_{cvis}^{\lambda_{a_l,p_j}}\} \tag{2}$$

Notice that *PyClustUI* visualizes as plots indicative results of this process, such as the frequency of best performing algorithm across CVIs as well as the number of clusters obtained.

III. Demonstration

The main functionalities of PyClust are bundled as an open-source docker image[2]. The user interface of *PyClustUI* is built upon Gradio [6] and the repository can be found here[3]. *PyClustKit* can be installed independently using PyPI[4] or downloaded as a github repository[5]. Jupyter notebooks that demonstrate the capabilities of *PyClustKit* can be found in the project's GitHub repository. Additionally, a short video demonstrating PyClust's functionalities, both for *PyClustKit* and *PyClustUI*, can be found here.

In this section we present three scenarios that demonstrate the advantages of PyClust. The first describes the usage of the developed library *PyClustKit* programmatically, while the other two describe two scenarios that rely on the *PyClustUI* graphical user interface.

[2]https://hub.docker.com/repository/docker/giannispoy/pyclust/general
[3]https://github.com/automl-uprc/PyClustUI
[4]https://pypi.org/project/pyclustkit/
[5]https://github.com/yannispoulakis/pyclustkit

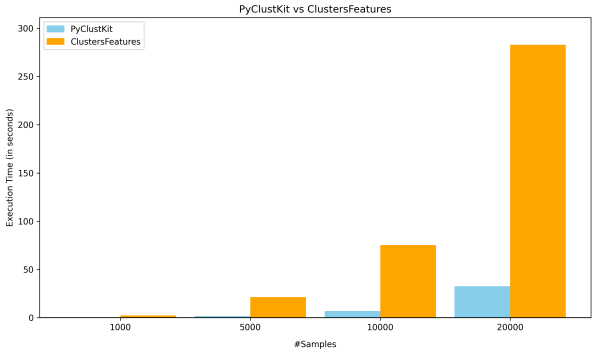

Fig. 2. Comparison of PyClustKit against ClusterFeatures in terms of execution time for a fixed number of 50 features.

a) Scenario 1: Clustering Evaluation and Meta-feature Extraction in Python: As previously mentioned, we provide access to CVI computation and meta-feature extraction through a Python library. This option is significantly more lightweight than running a docker container on any setup and provides complete control over the implemented functionalities. The library has been designed to be as user-friendly as possible and both operations, CVIs computation and meta-feature extraction, can be performed with just a few lines of code. Listing 1 shows how a data scientist can extract the meta-features of a given dataset D and compute the CVIs of interest.

```
from pyclustkit.metalearning import
    MFExtractor
from pyclustkit.eval import CVIToolbox

mfe = MFExtractor(D)
cvit = CVIToolbox(D, labels)

mfe.calculate_mf(category="descriptive")
print(mfe.search_mf(category="descriptive",
    search_type="values")

>>> {"no_attributes": 10,
        "no_instances": 1000, ... }

cvit.calculate_cvi(cvi=["dunn","sdbw"])
print(cvit.cvi_results)

>>> {"dunn": 0.79, "sdbw": 0.12}
```

Listing 1. Usage example of PyClustKit library

For the computation of CVIs, we additionally provide the results of benchmark tests against other popular libraries used for clustering evaluation, namely ClustersFeatures [7] and ClusterCrit [8]. This experiment aims to demonstrate the advantages of reusing previously computed results that *PyClustKit* applies through its sub-process sharing methodology. We have opted to exclude Scikit-Learn [3] due to the very low number of CVIs supported (3). As *PyClustKit* includes a superset of the CVIs implemented in the other libraries and to ensure a fair comparison, we instruct *PyClustKit* to compute only the

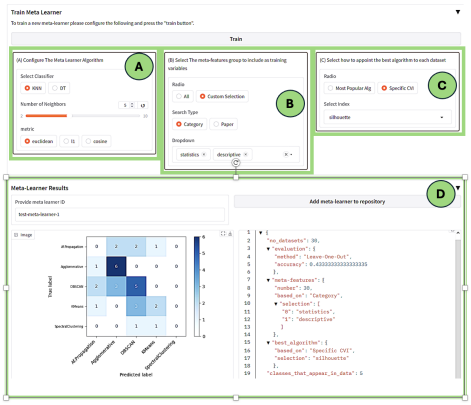

Fig. 3. Meta-learner training in PyClust.

CVIs supported by each of the other libraries, respectively. To that end, we generated synthetic datasets, via the scikit-learn python library, varying in sample size (1,000, 5,000, 1,0000, 20,000) and number of features (10, 50, 100).

Computing CVIs with ClusterCrit for datasets with more than 5,000 samples was manually terminated at the 4 hours threshold and no results were retrieved. This indicates that ClusterCrit does not scale well as the sample size increases. *PyClustKit* had an execution time of no more than 10 minute to compute the CVIs for sample sizes varying from 5,000 to 20,000. Results against the ClusterFeatures library are presented for a fixed feature size of 50 in Figure 2. For other values of feature size the improvement in execution time remains substantially large ranging from 56.69% to 96.89% for the combinations of number of samples and features, 20,000-100 and 1000-10, respectively. The results validate that *PyClustKit* is significantly faster in execution time than all of its competitors. All the benchmark tests and data are publicly available in the project's GitHub repository.

b) Scenario 2: Creating a Meta-Knowledge Repository: Meta-learning continually improves as more experience on the task at hand is accumulated. Specifically, for the task of algorithm selection that *PyClustUI* supports (Section III-0c), we learn from the association of dataset meta-features to model performance. To extract meta-features in *PyClustUI* is as simple as possible and can be performed with a single click of a button. For parameter search, *PyClustUI* facilitates a grid-search approach. A set of algorithms along with their respective parameters is evaluated on all possible CVIs. The search space can be configured by the user to better suit potential computational limitations. When this process concludes, two key figures are displayed in a designated user-interface section. The first one displays the frequency of each algorithm when selecting the set of trials that are retrieved by optimizing each index. The second plot displays the number of clusters found, again for the same set of configurations.

Moreover, the user interface includes a section to further explore grid search results. In this section, the user may search for the configuration that optimizes a selected index and view a color-coded scatter plot based on the partitioning of this selected configuration. If the active dataset is multidimensional, it can be reduced to two dimensions according to one of the three available dimensionality reduction methods (PCA, MDS, T-SNE) for visualization.

c) Scenario 3: Meta-learning for Algorithm Selection: After they have built a meta-knowledge repository, *PyClustUI* allows users to train their own meta-learner through its easy-to-use graphical interface and support the task of automated algorithm selection. Users need to head to the respective section in the user interface to provide the configuration, as shown in Figure 3. The main items to configure are the choice of classification meta-learner and its parameters (Figure 3-A), the subset of meta-features to use (Figure 3-B), and which CVI will be used to indicate the best performing algorithm (Figure 3-C). After the training has been concluded, various metadata and evaluation metrics are depicted, such as the number of datasets used and the confusion matrix of the results (Figure 3-D). Finally, users can head to the main section of *PyClustUI* and use the trained meta-learner to infer the algorithm selection for a new dataset. There, they need to (i) upload a dataset, (ii) extract its corresponding meta-features and (iii) apply the meta-learner.

IV. Conclusion and Future Work

In this paper we presented our framework for state-of-the-art meta-feature generation and evaluation of clustering results. While the complete framework can be accessed as a docker image, certain functionalities are also distributed as a Python Library. Potential future enhancements include the usage of AutoML techniques for parameter optimization, further visualization techniques for explainability, and integrating other novel meta-features and CVIs.

Acknowledgment

This work has been partly supported by the University of Piraeus Research Center. Also, it has received funding from the Horizon Europe research and innovation programme under the GA 101070416 (project Green.DAT.AI).

References

[1] Y. Poulakis, C. Doulkeridis, D. Kyriazis, "A Survey on AutoML Methods and Systems for Clustering", ACM Transactions on Knowledge Discovery Data, 2024
[2] M. Halkidi, Y. Batistakis, M. Vazirgiannis, "On Clustering Validation Techniques", Journal of Intelligent Information Systems, 2001
[3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python", Journal of Machine Learning Research, 2011
[4] J. Vanschoren, "Meta-Learning: A Survey", https://arxiv.org/abs/1810.03548, 2018
[5] Souza et al. 2022. Decision trees with short explainable rules. Advances in neural information processing systems 35 (2022), 12365–12379.
[6] A. Abubakar, A. Abdalla, A. Abid, D. Dawood, A. Alfozan, J. Zou, "Gradio": Hassle-Free Sharing and Testing of ML Models in the Wild", arXiv preprint arXiv:1906.02569, 2019
[7] S. Bertrand, "Clusters-features : a python module to evaluate the quality of clustering." https://github.com/Simon-Bertrand/Clusters-Features, 2021
[8] I. Giné-Vázquez, B. Desgraupes, "ClusterCrit": Clustering Indices (2023), https://cran.r-project.org/web/packages/clusterCrit/index.html