Dataset2Graph: A GNN-based Methodology for AutoML for Clustering

Emmanouil Dilmperis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
e.dilmperis@gmail.com

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

In this paper, we propose `Dataset2Graph`, a novel methodology that targets the problem of automated machine learning (AutoML) for clustering. `Dataset2Graph` offers an end-to-end solution to meta-learning, by encoding any input dataset as a similarity graph which undergoes graph reduction techniques (sparsification and coarsening) to obtain a smaller and more concise graph representation of the dataset. Besides making the processing more efficient, the graph reduction aims to keep the most important structural properties of the dataset. Subsequently, a graph neural network (GNN) architecture is applied on the reduced graph, producing a graph embedding, which is propagated to a meta-learner that processes this embedding to predict the best clustering algorithm for the dataset at hand. Our empirical evaluation using a set of 50 collected datasets shows the effectiveness of `Dataset2Graph` for different GNN models under various settings and in comparison with the state-of-the-art.

KEYWORDS

AutoML, clustering, meta-learning, algorithm selection, graph neural networks

ACM Reference Format:
Emmanouil Dilmperis, Yannis Poulakis, Dimitris Petratos, and Christos Doulkeridis. 2025. Dataset2Graph: A GNN-based Methodology for AutoML for Clustering. In *Workshop on Data Management for End-to-End Machine Learning (DEEM '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3735654.3735947

1 INTRODUCTION

In recent years, Machine Learning (ML) has become an essential tool for scientists and practitioners to solve complex problems from a wide variety of application domain fields. However, a variety of tasks in the ML life cycle, for instance preprocessing, algorithm selection, hyperparameter tuning and model evaluation, are quite cumbersome and time-consuming, requiring testing a very large number of configurations. In turn, this has sparked new interest in the creation of automation tools for ML that will assist a data scientist.

Automated Machine Learning (AutoML) [8] is the subfield of ML that simplifies ML tasks and automates the end-to-end ML pipeline, giving the opportunity to build ML models with minimum effort. AutoML frameworks focus on different aspects of ML, including automated feature extraction, automated algorithm selection and hyperparameter optimization.

One of the most important tasks targeted by AutoML is automated algorithm selection. Given a set of available ML algorithms A, automated algorithm selection aims to tackle the problem of selecting the best algorithm $\alpha^* \in A$ that optimizes a metric function $\mathcal{L}$ for a given dataset D:

$$\alpha^* = \arg\min_{\alpha \in A} \mathcal{L}(\alpha, D) \tag{1}$$

Although AutoML has attracted the attention of the scientific community for supervised tasks, such as classification and regression, little has been done towards AutoML for unsupervised tasks like clustering. In this paper, we provide an AutoML methodology, termed `Dataset2Graph`, for automated algorithm selection for clustering, which aims to represent a tabular dataset as a graph and use a GNN to learn a graph embedding. Then, a meta-learner can be trained on graph embeddings of known datasets with their best performing clustering algorithm, in order to predict the best clustering algorithm for a new dataset.

In technical terms, the gist of `Dataset2Graph` is to represent each dataset as a similarity graph. Then, graph reduction techniques (sparsification and coarsening) are applied to derive a compact and concise graph of much smaller size. Node features are computed on this reduced graph and provided as input to a GNN, which produces a graph embedding. Finally, the graph embedding is provided as input to a to downstream MLP layers that predict the best performing clustering algorithm. A notable advantage of `Dataset2Graph` is that it does not rely on hand-crafted meta-features that may be task-specific, but instead exploits the graph abstraction to automatically produce a graph embedding for any given input dataset.

In summary, in this paper, we make the following contributions:

DEEM '25, June 22–27, 2025, Berlin, Germany

ACM ISBN 979-8-4007-1924-0/2025/06
https://doi.org/10.1145/3735654.3735947

- We introduce `Dataset2Graph`, an AutoML methodology for clustering algorithm selection that relies on graph-based representations of datasets.
- We advocate the use of Graph Neural Networks (GNN) for obtaining a latent representation for each dataset, which can be used to train a meta-learner to select the best performing clustering algorithm.
- We highlight the importance of graph reduction techniques, namely sparsification and coarsening, for GNN-based approaches in the context of AutoML for clustering.
- We provide experimental results for different GNN architectures, various graph reduction methods, and for different settings, which demonstrate the advantages of our methodology.

The rest of this paper is structured as follows. Section 2 describes the meta-learning approach followed to build a meta-knowledge repository of datasets along with the best performing clustering algorithm. Then, Section 3 outlines the `Dataset2Graph` methodology, which includes the graph representation of the input dataset and learning a graph embedding, as well as the prediction of the best clustering algorithm. In Section 4, we report on the results of the empirical evaluation, while the related work is covered in Section 5. Finally, Section 6 concludes the paper and describes future plans for extensions.

2 META-LEARNING FOR CLUSTERING ALGORITHM SELECTION

Meta-learning [19] is the subfield of ML, where models are trained to quickly adapt to new tasks using prior experience. Instead of learning from scratch from a given dataset, a meta-learning model has already been trained to multiple other tasks in the past, in order to provide quick solutions to new ones from past experiments. Also, meta-learning is widely used for algorithm selection [15].

The basis of our meta-learning approach is to leverage insights from the evaluation of the performance of different clustering algorithms on a set of available datasets $\mathcal{D} = \{D_1, D_2, \ldots, D_{|D|}\}$. To this end, we create a meta-knowledge repository which consists of a collection of numerical datasets along with their corresponding best performing clustering algorithm. Finding this best performing algorithm is achieved in an offline phase, by evaluating a set of algorithm configurations exhaustively.

Formally, each algorithm a_i from the set $A = \{a_1, ..., a_{|A|}\}$ has a respective parametric space denoted as $\Lambda_{a_i} = \{\lambda_{a_i,p_j} : j = 1 \ldots, k\}$, where λ_{a_i,p_j} is the domain of parameter p_j and k denotes the number of parameters of algorithm a_i. Therefore, the search space Θ is the union of the Cartesian products of algorithm configurations, as in Equation 2. Continuous-valued domains of parameters are discretized accordingly.

$$\Theta = \bigcup_{i=1}^{|A|} (\{a_i\} \times \Lambda_{a_i}) \tag{2}$$

We refer to a specific algorithm a_i with parameterization λ_{a_i,p_j} as a *clustering algorithm configuration*. Hence, we train each clustering algorithm configuration (a_i, λ_{a_i,p_j}) and evaluate the results accordingly to retrieve the one that optimizes a specified metric $\mathcal{L}$,

$$a^* = \arg\max_{a \in A} \mathcal{L}(a) \tag{3}$$

It is important to note that we evaluate an algorithm using different values of hyperparameters in the offline phase, in order to find the best performing algorithm a^*. Then, in the online phase, we solve the problem of algorithm selection only.

In order to populate our meta-knowledge repository we have carefully selected a collection of numeric datasets that contain information on the ground truth clustering. We stress that such information is only used to create a robust meta-knowledge repository and it is *not required* for new datasets that become available in the online phase (during inference). Put differently, we do not use nor require knowledge about the ground truth for a new dataset, which aligns with the unsupervised nature of the clustering task. Additionally, the availability of ground truth during the training of the meta-learner allows the use of an external cluster validity index (CVI). The Adjusted Rand Index (ARI) is an external CVI which evaluates the performance of a clustering algorithm configuration by comparing its results (the discovered clusters) against the ground truth:

$$ARI(X,Y) = \frac{\sum_{ij} \binom{n_{x_i y_i}}{2} - \sum_i \binom{n_{x_i}}{2} \sum_j \binom{n_{y_j}}{2} / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{x_i}}{2} + \sum_j \binom{n_{y_j}}{2}\right] - \left[\sum_i \binom{n_{x_i}}{2} \sum_j \binom{n_{y_j}}{2}\right] / \binom{n}{2}} \tag{4}$$

where $X = \{x_1, \ldots, x_i\}$ is the set of the produced clusters, $Y = \{y_1, \ldots, y_j\}$ is the set of ground truth clusters, n is the total number of instances, $n_{x_i} = |x_i|$ and $n_{x_i y_j} = |x_i \cap x_j|$. ARI's values are in the interval $[-1, 1]$, where 1 indicates full identification, whereas -1 indicates no identification.

3 DATASET2GRAPH

In this section, we present our methodology (`Dataset2Graph`) for AutoML for clustering. We focus on the online phase. The methodology is illustrated in Figure 1 and is presented in the following subsections. Two main processes can be identified. First, the input dataset is represented by a similarity graph, which undergoes graph reduction to obtain a concise representation of the dataset (Section 3.1). Then, this is fed into a graph neural network (GNN) architecture that produces an embedding for the graph, which is given as input to a classifier that predicts the best clustering algorithm (Section 3.2).

3.1 Graph Construction

In this section, we describe our approach for transforming a tabular dataset, where rows correspond to records and columns to attributes, into a graph that will be used later for learning a representation in the form of an embedding.

3.1.1 Pre-processing. Initially, the input dataset is pre-processed to facilitate the subsequent data analysis steps. First, columns with unique values and columns with more than 30% of missing values were eliminated. For columns with fewer missing values, we replaced the missing values with the average value of the k-nearest neighbors (with k=10). Finally, we performed normalization of the values to $[0, 1]$.

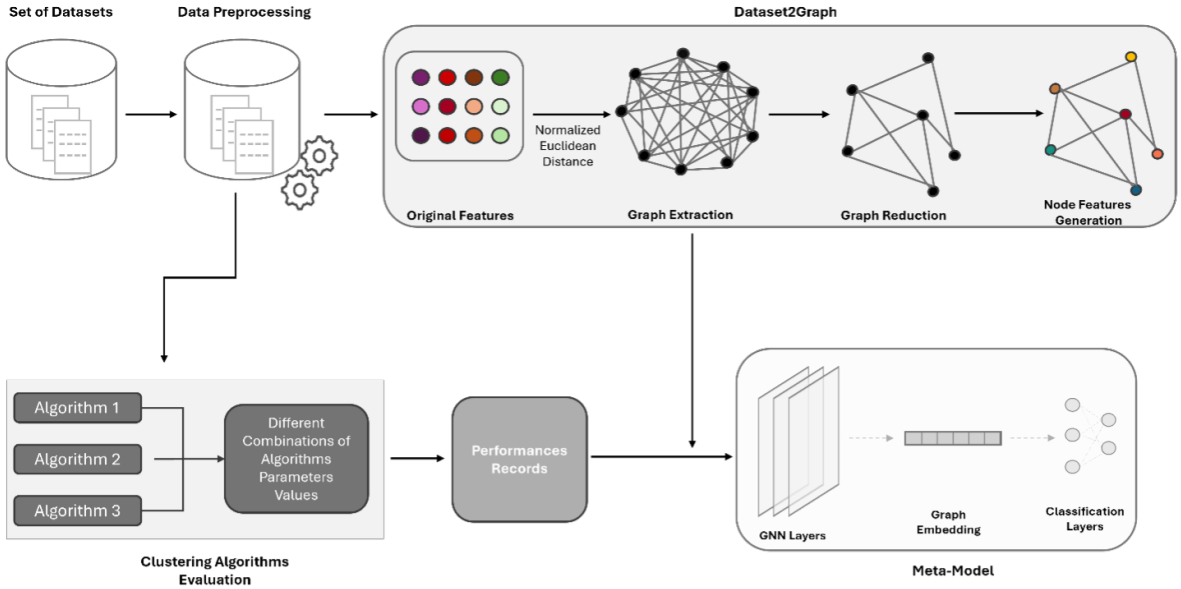

Figure 1: The `Dataset2Graph` methodology.

3.1.2 Similarity Graph. Given the pre-processed input dataset D that consists of n rows and m columns, we construct the similarity graph as follows. First, each row is represented as a node, while edges capture the similarity between pairs of rows. Thus, we construct a fully-connected weighted graph $G(V, E, w)$, $|V| = n$, $|E| = \frac{n(n-1)}{2}$, where an edge's weight $w(i, j)$ between node $v_i \in V$ and $v_j \in V$ represents the similarity between rows i and j and is formally defined as: $w(i, j) = 1 - \frac{d(i,j)}{\sqrt{m}}$. , where $d(i, j)$ stands for the Euclidean distance between rows i and j and the weight is normalized to take values in $[0, 1]$.

3.1.3 Graph Reduction. The similarity graph G of a dataset D is dense and not appropriate for use as input for a GNN. The reasons are manifold. First, the computational cost will be high as the number of edges is $|E| = O(n^2)$. Second, dense graphs make it hard for a GNN to identify local patterns. In addition, oversmoothing [24] may occur, a situation in which node embeddings learnt from the GNN tend to converge to similar values, rendering them indistinguishable. For these reasons, we apply *graph reduction* techniques [7] on G to obtain a more concise representation. We apply two categories of graph reduction, namely *sparsification* and *coarsening*.

Graph sparsification [13] is a graph simplification method that reduces the number of edges. Given an input graph $G(V, E)$, the output of graph sparsification is a new graph $G'(V, E')$, which contains the same set of vertices but a subset of the edges $E' \subset E$. Intuitively, the aim is to eliminate the noise present in the dense original graph, by retaining only the most important edges. This will also help a GNN to focus on these connections and avoid oversmoothing. Also, graph sparsification maintains the spectral properties of the graph.

On the other hand, graph coarsening creates a new representation $G_c = (V_c, E_c)$ of the original graph $G(V, E)$, where $|V_c| < |V|$ and $|E_c| < |E|$ that maintains its properties. In contrast to sparsification, the result of coarsening is not a subgraph, but instead a new graph where vertices are the result of node merging. By means of an iterative process, a pair of vertices is selected to be merged in each iteration, until a desired number of vertices is reached. When two vertices are merged, a new node is created that keeps the edges of the individual vertices with weights equal to an aggregation of the weights of the original edges.

`Dataset2Graph` adopts a three-step graph reduction procedure. In the first step, we eliminate edges with weights below a given threshold w_{thr}. As such, only edges with weight $w(i, j) \geq w_{thr}$ are retained. However, this may result in a disconnected graph, which is not desirable because it will prevent message passing between vertices in the subsequent graph learning phase. To avoid this, we compute the maximum spanning tree using Kruskal's algorithm over negated edge weights, which returns a connected tree with maximum aggregate weights. Thus, before eliminating an edge, we check if it belongs to the maximum spanning tree, and if this is the case we keep the edge. Using this technique, we can guarantee that the result will be a connected graph.

In the second step, we apply graph sparsification. Different techniques are applicable towards this goal:

- k-neighbor: for each node, k edges are selected to be retained with probability proportional to their weights. In case a node has fewer than k edges, they are all retained.
- Rank degree [22]: selects a random number of nodes (seed nodes), sorts their neighbors in decreasing order of node degree, and retains a percentage of edges that connect the seed nodes to top neighbors. The retained neighbors become new seed nodes, and the process is repeated until a desired graph density is obtained.

- Local degree [6]: keeps the edges of weights that corresponds to nodes with high degree in a deterministic manner. For each node $u \in V$, the edges connecting it to its top $deg(u)^a$ neighbors are embedded in the subgraph, which are ranked based on their degree in descending order. The parameter $a \in [0, 1]$ controls the sparsification of the graph. The algorithm used also ensures that the produced graph will remain fully connected after sparsification.
- t-Spanner [1]: creates a sparse graph by deleting edges between nodes v and u, if exists a path that connects them and satisfies the condition:

$$\text{Shortest\_Path\_Weight}(v, u) \leq t \cdot \text{Edge\_Weight}(v, u) \tag{5}$$

  where t defines a stretching coefficient.
- L-Spar [16]: a variation of local degree. Their main difference is that neighborhoods' classification is based on Jaccard similarity

$$\text{Jaccard}(v, u) = \frac{|N(v) \cap N(u)|}{|N(v) \cup N(u)|} \tag{6}$$

  where $N(v)$ is the neighborhood of v.

In the third step, a graph coarsening technique is applied to result in a user-specified targeted number of nodes. We use the following iterative algorithms:

- Heavy Edge Matching [10]: a heuristic algorithm for finding a maximal matching of a graph. A matching of a graph, is a set of edges, no two of which are incident on the same node. A matching of maximal size is called maximal. The nodes are visited in random order, but a node is matched with its (unmatched) neighbor node with highest weight (the heavier edge).
- Algebraic Distance [2]: the algebraic distances $s(u, v)$ are calculated for every edge $(u, v) \in E$. These distances depend on two parameters: a relaxation parameter (ω) and the number of iterations (k). As $k \longrightarrow +\infty$, the algebraic distances converge to zero. The speed with which an algebraic distance $s(u, v)$ converges to zero is an indicator of how strong the connection is between u and v.
- Local Variation (Edges) [13]: merges vertices whose edge weights show minimal change compared to their local neighborhood. This method evaluates the difference in edge weights between a node and its neighbors, merging those with relatively similar weights. The goal is to preserve local structural properties while reducing the size of the graph.

3.1.4 Node Features. After having obtained the reduced graph, we perform feature extraction for its nodes, so that each node is represented by a feature vector. The extracted features reflect the structural properties of the vertices of the reduced graph. We extract the following features:

- Node Degree: The number of u neighbors

$$x_u = |N(u)| \tag{7}$$

  where $N(u)$ is the neighborhood of node u, i.e., the nodes connected to u with an edge.
- Eigenvector centrality: A metric indicating the influence that a node has within the graph, taking into consideration the multitude of its neighbors and their quality

$$x_u = \frac{1}{\lambda} \sum_{v \in N(u)} e_{u,v} x_v \tag{8}$$

  where $e_{u,v}$ is the weight of the edge (u, v) and $\lambda > 0$ is a constant, mostly taken as the largest eigenvalue of the adjacency matrix.
- Betweenness Centrality: A metric that quantifies the importance of node, regarding its participation in the minimal paths of a graph

$$x_u = \sum_{v \neq u \neq t} \frac{\#\text{minimal paths between } v \text{ and } t \text{ that contains } u}{\#\text{minimal paths between } v \text{ and } t} \tag{9}$$

- Closeness Centrality: A metric that quantifies the effectiveness of a node to have access to all the nodes of a graph

$$x_u = \frac{1}{\sum_{u \neq v} \#\text{length of minimal path between } v \text{ and } u} \tag{10}$$

- Clustering Coefficient: A metric that quantifies the trend of nodes to form clusters, examining the connectivity of a node's neighbors

$$x_u = \frac{\#\text{edges between the neighbors of } u}{\binom{|N(u)|}{2}} \tag{11}$$

Thus, we build a feature vector with these features for each node of the reduced graph. The aim is to provide these feature vectors as input to a GNN, so that the GNN can learn the graph topology and encode it in a single vector that we call graph embedding.

3.2 The Meta-model

The meta-model is responsible for finding the best clustering algorithm for a given dataset. This problem is a classification problem where the predicted class corresponds to the clustering algorithm that is considered best for the given dataset. The meta-model takes as input a reduced graph G_i (of dataset D_i) represented as follows:

- a matrix X_i ($n' \times d'$), which contains the node features, where n' is the number of vertices in the reduced graph G_i and d' is the number of extracted features.
- a list E_i that contains the edges present in G_i
- a list E_i^w of edge weights that correspond to the edges in E_i
- a label l_i which corresponds to the best performing clustering algorithm which has been identified during the clustering evaluation step.

The meta-model is a GNN architecture that consists of: (a) in input layer which takes as input X_i, E_i and E_i^w, (b) multiple GNN layers, (c) a readout layer that extracts the graph embedding, and (d) classification layers that are responsible for the prediction of the clustering algorithm.

In more detail, let H_i^k denote the matrix of node characteristics of the i-th dataset extracted in the k-th layer of the GNN. Let us use H^0 to denote the input matrix, i.e., $H_i^0 = X_i$. At the $k+1$ layer we obtain:

$$H_i^{k+1} = \sigma(agg(H_i^k, E_i, W^k)) \tag{12}$$

where $agg()$ is the message passing function which differs from one GNN to another, W^k is the learnable weight matrix, E_i is the edge representation matrix and σ is the activation function.

Then, a pooling layer is applied on the k-th layer (the final layer) to obtain the graph embedding:

$$\text{GraphEmbedding}_i = \text{Average\_Pooling}(H_i^k) \tag{13}$$

Finally, the vector $GraphEmbedding_i$ is given as input to the classification layers to obtain the predicted label $\hat{l_i}$ (clustering algorithm):

$$\hat{l_i} = \max(\text{softmax}(f(\text{GraphEmbedding}_i))) \tag{14}$$

where $f()$ denotes a function that performs linear transformation, and $softmax()$ that converts the vector into a probability distribution. The label is determined by the maximum (max()) probability.

4 EXPERIMENTAL EVALUATION

We implemented the Dataset2Graph methodology in Python, using various libraries: NumPy, NetworkKit, NetworkX, whereas for the GNNs PyTorch Geometric was employed. We also implemented MARCO-GE [3] in Python for comparison.

4.1 Experimental Setup

4.1.1 Datasets and Clustering Algorithms. We evaluated the performance of `Dataset2Graph` on 50 real-world datasets from UCI Machine Learning Repository, Kaggle and OpenML. The selected datasets showcase diversity in terms of domain field, number of features and number of instances. We evaluate the performance of `Dataset2Graph` for clustering algorithm selection, taking into consideration three clustering algorithms that are considered representatives of different categories. More precisely, we select *K-Means* from the centroid-based algorithms, *DBSCAN* from the density-based algorithms and *Agglomerative Clustering* from the hierarchical algorithms. Table 1 lists the algorithms, their parameters that were used in this study, as well as the number of datasets for which they were selected as best performing algorithms.

4.1.2 Evaluation Methodology and Metrics. As an evaluation methodology, we chose leave-one-out cross validation (LOOCV), due to the limited number of training data. In LOOCV, each time, a single dataset (graph) is used for validation, while the other 49 serve as a training set. The procedure is finalized when all available graphs are used as a validation set, extracting the mean of the corresponding metric.

The two metrics used to evaluate `Dataset2Graph` performance are F1-Macro and F1-Weighted. F1-score is the harmonic mean of Precision and Recall: $F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$. Macro F1-score is the average of F1-score of every cluster

$$F1_{macro} = \frac{F_1^1 + \ldots + F_1^n}{n} \tag{15}$$

and weighted F1 score is the sum of clusters' F1-scores, taking into consideration clusters' weights

$$F1_{weighted} = weight_1 \cdot F_1^1 + \ldots + weight_n \cdot F_1^n \tag{16}$$

4.1.3 Model Design. We experimented with different GNN layers, including Graph Convolution Networks (GCN) [11], Graph Attention Networks (GAT) [20] and Graph Isomorphism Networks (GIN) [23]. In addition, we tried different combinations of graph reduction techniques, as described in Section 3.1.3, and different node features defined in Section 3.1.4. The results of the most promising model designs are mentioned in Section 4.2.

Algorithm	Category	Parameters	# Datasets
K-Means	Centroid	n_clusters	25
DBSCAN	Density	eps, min_samples	15
Agglomerative	Hierarchical	n_clusters, linkage	10

Table 1: Clustering Algorithms used in `Dataset2Graph`.

Table 2: Importance of node features and graph reduction.

Metric	Approach	GCN	GAT	GIN
F1-Macro	**With (all)**	**0.630**	**0.494**	**0.404**
	Without	0.343	0.343	0.266
	With (degree)	0.467	0.426	0.401
F1-Weighted		0.513	0.455	0.492
	With (all)	**0.647**	**0.486**	**0.499**
	Without	0.445	0.388	0.367

4.2 Results

In this Section we describe the results obtained from an exhaustive experimentation procedure in multiple levels. In Section 4.2.1 we emphasize the importance of node features diversity. In Section 4.2.2 we highlight the importance of a join sparsification and coarsening approach for graph reduction and in Section 4.2.3 we compare `Dataset2Graph` with its basic GNN-based baseline, MARCO-GE.

Table 2 contains the best results obtained from our experimentation comparing different GNN architectures and combinations of sparsification and coarsening. A more detailed study of the experimental results is presented in Appendix A.

4.2.1 Node Features. In Table 2 the rows **With (degree)** indicate that in the model design we only used node degree (Equation 7) as a node features. The rows **With (all)** indicate that in the model design we used all node features defined in Section 3.1.4. The corresponding GCN, GAT and GIN columns, refer to the results obtained from the best sparsification and coarsening combinations, as can be found in Appendix A, Tables 4, 5, 6 and 7.

It is obvious from Table 2 and the corresponding rows, that the selection of graph topological features defined in Section 3.1.4, enhances the performance of `Dataset2Graph`.

4.2.2 Graph Reduction. We also investigated the importance of sparsification and coarsening for `Dataset2Graph`. In Table 2, rows **Without** indicates that no sparsification or coarsening technique was used for the graph reduction. For these experiments, the graph reduction was performed using a threshold over the weights, keeping only 60% of them, and the node features used were all defined in Section 3.1.4, as it was empirically proven in 4.2.1 that they are important. A more detailed evaluation of **Without** rows can be found in Appendix A, Table 8. The results show that performance is improved when we use sparsification and coarsening, keeping all features in the model design.

	Dataset2Graph	MARCO-GE [3]
F1-Macro	**0.630**	0.373
F1-Weighted	**0.647**	0.438

Table 3: Comparison of Dataset2Graph with MARCO-GE [3].

4.2.3 Comparison with MARCO-GE. In Table 3 we compare the best performing Dataset2Graph model designs (Appendix A, Table 9) with its basic baseline MARCO-GE.

Dataset2Graph best performing model design includes GCN layers, as can be found in Appendix A. The results of the comparison are shown in Table 3 and indicate that Dataset2Graph outperforms MARCO-GE on both F1-Macro and F1-Weighted.

5 RELATED WORK

AutoML for clustering is a challenging research topic, which has attracted wide attention recently. For a detailed survey, we refer to [15]. Several research efforts follow a meta-learning approach [18, 21], where a set of meta-features is extracted from datasets, which are used to train a meta-learner upon a meta-knowledge repository to identify the best performing algorithm. This repository typically contains previously processed datasets along with information about their corresponding best performing algorithm. Then, prediction of the best clustering algorithm can be performed by finding similar datasets based on the meta-features.

Following this approach, preliminary research works have proposed the use of statistical meta-features (e.g., mean, variance, etc.) computed over each column of the dataset and the aggregated across columns. Later, Ferrari and Castro [5] included the use of distance-based meta-features for automated clustering. AutoClust [14] introduces some landmark meta-features, internal cluster validity indices values after training the non-parametric MeanShift algorithm. TPE-AutoClust [17] also introduces some additional landmark meta-features, 3 different internal CVI calculated after training 3 clustering algorithms. Furthermore, AutoCluster [12] examines landmark meta-features that capture characteristics of trained clustering algorithms such as the number of leaves of the Agglomerative clustering algorithm and additionally proposes the Hopkin's statistical test to measure spatial randomness as a meta-feature.

Recent approaches for AutoML for clustering apply dataset-agnostic meta-feature extraction, without relying on explicitly predefined meta-features. TRIO [4] learns graphical representations from datasets and then extracts a latent representation by means of a Graph Convolutional Neural Network. Dataset2Vec [9] combines engineered meta-features with meta-features learnt by a deep neural network. The basic idea is to encode a dataset as a vector and use such vectors as proxies for computing similarities between datasets.

MARCO-GE [3] follows an approach where each dataset is processed by PCA and the projected dataset is transformed into a similarity graph using the cosine similarity. The graph is simplified using a crisp similarity threshold, i.e., edges with weights lower than 0.9 are removed. Then, DeepWalk is applied to extract the node features and a Graph Convolutional Neural Network is used to learn the graph embedding. Finally, an XGBoost meta-model is used for the clustering algorithm prediction.

Although Dataset2Graph shares similarities with MARCO-GE, there exist important differences at technical level that need to be highlighted. First, as described in Section 3.1.2, Dataset2Graph constructs a fully-connected weighted graph based on normalized Euclidean distances. On the other hand, MARCO-GE constructs the similarity graph based on cosine similarity. Second, as described in Section 3.1.3, Dataset2Graph reduces the number of graph edges based on coarsening and sparsification techniques, empowering the topological conservation of the graph. In contrast to that, MARCO-GE reduces the number of graph edges based on a weight threshold, keeping only 90% of them. Third, as described in Section 3.1.4, Dataset2Graph creates node features based on graph topological features, while on the other hand, MARCO-GE extracts node features based on random walks. Last, but not least, after extraction of the embedding, Dataset2Graph uses jointly a Multilayer Perceptron (MLP) to classify its graph. On the other hand, MARCO-GE does not use the model per se to classify the graph. It uses the extracted embedding and fits it to an XGBoost meta-learner to predict the class of the graph.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we presented Dataset2Graph, an end-to-end AutoML methodology for clustering algorithm prediction. Dataset2Graph represents a tabular dataset in a concise graph representation, which is then used as input to a graph neural network to produce a graph embedding, so as to eventually enable a meta-learner to predict the best clustering algorithm based on this graph embedding. After having identified the main steps of the methodology, we have tested and compared different technical solutions for each step. Also, we performed a comparison with a state-of-the-art approach. The results indicate that Dataset2Graph can be effectively applied on the problem of AutoML for clustering, demonstrating promising preliminary results and leaving room for future optimizations at different steps of the proposed methodology.

ACKNOWLEDGMENTS

The research leading to the results presented in this paper has received funding from the European Union's funded Projects: MobiSpaces under grant agreement no 101070279 and Green.DAT.AI under grant agreement no 101070416.

REFERENCES

[1] Surender Baswana and Sandeep Sen. 2007. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms* 30, 4 (2007), 532–563. https://doi.org/10.1002/RSA.20130

[2] Jie Chen and Ilya Safro. 2011. Algebraic Distance on Graphs. *SIAM J. Sci. Comput.* 33, 6 (2011), 3468–3490. https://doi.org/10.1137/090775087

[3] Noy Cohen-Shapira and Lior Rokach. 2021. Automatic selection of clustering algorithms using supervised graph embedding. *Inf. Sci.* 577 (2021), 824–851. https://doi.org/10.1016/J.INS.2021.08.028

[4] Noy Cohen-Shapira and Lior Rokach. 2021. TRIO: Task-agnostic dataset representation optimized for automatic algorithm selection. In *IEEE International Conference on Data Mining, ICDM 2021, Auckland, New Zealand, December 7-10, 2021*, James Bailey, Pauli Miettinen, Yun Sing Koh, Dacheng Tao, and Xindong Wu (Eds.). IEEE, 81–90. https://doi.org/10.1109/ICDM51629.2021.00018

[5] Daniel Gomes Ferrari and Leandro Nunes de Castro. 2015. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Inf. Sci.* 301 (2015), 181–194. https://doi.org/10.1016/J.INS.2014.12.044

[6] Michael Hamann, Gerd Lindner, Henning Meyerhenke, Christian L. Staudt, and Dorothea Wagner. 2016. Structure-preserving sparsification methods for social networks. *Soc. Netw. Anal. Min.* 6, 1 (2016), 22:1–22:22. https://doi.org/10.1007/S13278-016-0332-2

[7] Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B. Aditya Prakash, and Wei Jin. 2024. A Comprehensive Survey on Graph Reduction: Sparsification, Coarsening, and Condensation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*. ijcai.org, 8058–8066. https://www.ijcai.org/proceedings/2024/891

[8] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automated Machine Learning - Methods, Systems, Challenges*. Springer. https://doi.org/10.1007/978-3-030-05318-5

[9] Hadi S. Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. 2021. Dataset2Vec: learning dataset meta-features. *Data Min. Knowl. Discov.* 35, 3 (2021), 964–985. https://doi.org/10.1007/S10618-021-00737-9

[10] George Karypis and Vipin Kumar. 1995. Multilevel Graph Partitioning Schemes. In *Proceedings of the 1995 International Conference on Parallel Processing, Urbana-Champain, Illinois, USA, August 14-18, 1995. Volume III: Algorithms & Applications*, Kyle A. Gallivan (Ed.). CRC Press, 113–122.

[11] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016). arXiv:1609.02907 http://arxiv.org/abs/1609.02907

[12] Yue Liu, Shuang Li, and Wenjie Tian. 2021. AutoCluster: Meta-learning Based Ensemble Method for Automated Unsupervised Clustering. In *Advances in Knowledge Discovery and Data Mining - 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11-14, 2021, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 12714)*. Springer, 246–258.

[13] Andreas Loukas. 2019. Graph Reduction with Spectral and Cut Guarantees. *J. Mach. Learn. Res.* 20 (2019), 116:1–116:42. https://jmlr.org/papers/v20/18-680.html

[14] Yannis Poulakis, Christos Doulkeridis, and Dimosthenis Kyriazis. 2020. AutoClust: A Framework for Automated Clustering based on Cluster Validity Indices. In *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, Claudia Plant, Haixun Wang, Alfredo Cuzzocrea, Carlo Zaniolo, and Xindong Wu (Eds.). IEEE, 1220–1225. https://doi.org/10.1109/ICDM50108.2020.00153

[15] Yannis Poulakis, Christos Doulkeridis, and Dimosthenis Kyriazis. 2024. A Survey on AutoML Methods and Systems for Clustering. *ACM Trans. Knowl. Discov. Data* 18, 5 (2024), 120:1–120:30. https://doi.org/10.1145/3643564

[16] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. 2011. Local graph sparsification for scalable clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis (Eds.). ACM, 721–732. https://doi.org/10.1145/1989323.1989399

[17] Radwa El Shawi and Sherif Sakr. 2022. TPE-AutoClust: A Tree-based Pipline Ensemble Framework for Automated Clustering. In *IEEE International Conference on Data Mining Workshops, ICDM 2022 - Workshops, Orlando, FL, USA, November 28 - Dec. 1, 2022*, K. Selçuk Candan, Thang N. Dinh, My T. Thai, and Takashi Washio (Eds.). IEEE, 1144–1153.

[18] Joaquin Vanschoren. 2018. Meta-Learning: A Survey. *CoRR* abs/1810.03548 (2018). arXiv:1810.03548 http://arxiv.org/abs/1810.03548

[19] Joaquin Vanschoren. 2019. Meta-Learning. In *Automated Machine Learning - Methods, Systems, Challenges*, Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). Springer, 35–61. https://doi.org/10.1007/978-3-030-05318-5_2

[20] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJXMpikCZ

[21] Anna Vettoruzzo, Mohamed-Rafik Bouguelia, Joaquin Vanschoren, Thorsteinn S. Rögnvaldsson, and KC Santosh. 2024. Advances and Challenges in Meta-Learning: A Technical Review. *IEEE Trans. Pattern Anal. Mach. Intell.* 46, 7 (2024), 4763–4779. https://doi.org/10.1109/TPAMI.2024.3357847

[22] Elli Voudigari, Nikos Salamanos, Theodore Papageorgiou, and Emmanuel J. Yannakoudakis. 2016. Rank degree: An efficient algorithm for graph sampling. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2016, San Francisco, CA, USA, August 18-21, 2016*, Ravi Kumar, James Caverlee, and Hanghang Tong (Eds.). IEEE Computer Society, 120–129. https://doi.org/10.1109/ASONAM.2016.7752223

[23] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ryGs6iA5Km

[24] Xu Zhang, Yonghui Xu, Wei He, Wei Guo, and Lizhen Cui. 2023. A Comprehensive Review of the Oversmoothing in Graph Neural Networks. In *ChineseCSCW (1)*. 451–465. https://doi.org/10.1007/978-981-99-9637-7_33

A MORE EXPERIMENTAL RESULTS

The aim of these experiments is to evaluate the different possible combinations of graph sparsification and coarsening techniques to find the best performing one. We also test different dimensionality values for the produced embeddings. Moreover, we evaluate the number of different layers that comprise the graph neural network. Last, but not least, we provide detailed results for different GNNs. The number of nodes in the reduced graph that were set as target were: 50, 100, 200, and 300.

Table 4 shows the results obtained using a Graph Convolutional Network (GCN), while Table 5 shows results from a Graph Attention Network (GAT), and finally Table 6 reports the results for a Graph Isomorphism Network (GIN). In these experiments, we use a single node feature, namely the node degree. We use labels for sparsification and coarsening, for example, LS10_AG100 refers to L-Spar sparsification with density parameter equal to 10 and Algebraic Distance coarsening that results in 100 nodes.

Table 4 shows the top-5 performing combinations of sparsification/coarsening for GCN (based F1-weighted score on the test set), while varying the number of layers in the GNN from 1 to 3. The best results are obtained when using 3 layers and with the combination LS15_AG200. Another general remark is that L-Spar seems to work well, as it appears quite often in the top-5. Also, it seems using fewer than 200 nodes in the reduced graph does not perform well (these configurations are never in the top-5).

Table 5 shows the top-5 results for GAT, where the 2-layer models performed better. There is no clear winner in terms of graph reduction algorithms. Finally, Table 6 depicts the results for GIN. Also here, there is no graph reduction method that works best consistently. Overall, it seems that the GCN with 3 layers provided the best results (LS15_AG200: 0.513), when used with L-Spar and Algebraic Distance. The second best approach (LS15_AG300: 0.492) was GIN with 2 layers with L-Spar and Algebraic Distance, but using 300 nodes instead of 200.

Then, we repeat the experiment but using more node features this time. Practically, we wish to explore how the addition of node features (described in Section 3.1.4) will affect the performance of `Dataset2Graph`. Table 7 shows the results obtained on the test set, comparing the use of one feature (with degree) vs. all 5 features (with all). It is clear that the addition of more features improves the performance of GCN. In contrast, in the case of GAT, this addition worsened the results. In the case of GIN, the results are mixed, the addition sometimes improved and others worsened the results.

Table 8 presents the performance of `Dataset2Graph` when we disable the use of graph reduction techniques. This experiment aims to answer the question whether graph reduction offers any advantages. In this experiment, the only step that was used to reduce the size of the graph was the crisp threshold w_{thr} which was set to 0.6, i.e., edges with weight smaller than 0.6 are discarded. Interestingly, we observe that all methods show worse performance compared to the previous experiments where graph reduction was used. This justifies the use of graph reduction in `Dataset2Graph`.

Table 9 shows the overall performance of `Dataset2Graph` when using both the graph reduction techniques as well as all the node features. These are the best results that we obtain, namely the F1-weighted score for the test set is equal to 0.647.

Sparsification/Coarsening	Embeddings dimensionality	Train F1-Macro	Test F1-Macro	Train F1-Weighted	Test F1-Weighted
1 GCN layer					
LS15_AG300	128	0.372	0.389	0.466	0.481
LS5_VA300	32	0.372	0.391	0.460	0.480
LS15_AG200	128	0.367	0.366	0.458	0.456
LS10_VA300	128	0.362	0.367	0.443	0.447
KN10_AG200	128	0.330	0.342	0.427	0.438
2 GCN layers					
TS7_AG200	32	0.404	0.420	0.483	0.498
TS7_HE200	32	0.402	0.420	0.480	0.498
LS5_VA300	128	0.383	0.389	0.471	0.480
LS15_AG200	256	0.413	0.376	0.485	0.459
LS10_VA300	128	0.364	0.367	0.444	0.447
3 GCN layers					
LS15_AG200	64	0.396	**0.467**	0.475	**0.513**
LS5_VA300	64	0.379	0.389	0.467	0.480
LS10_VA300	128	0.359	0.367	0.446	0.447
LS15_AG300	32	0.361	0.360	0.448	0.446
TS7_HE200	32	0.355	0.363	0.433	0.441

Table 4: Graph Convolutional Network (GCN): The top-5 performing variations of sparsification/coarsening for different number of layers. Use of one single feature (node degree).

Sparsification/Coarsening	Embeddings dimensionality	Train F1-Macro	Test F1-Macro	Train F1-Weighted	Test F1-Weighted
1 GAT layer					
LS5_VA200	256	0.243	0.345	0.335	0.411
KN15_AG300	128	0.253	0.348	0.353	0.407
TS7_AG200	32	0.396	0.342	0.470	0.403
LOC15_HE200	256	0.242	0.360	0.338	0.402
KN10_AG300	256	0.221	0.374	0.323	0.397
2 GAT layers					
TS7_HE200	32	0.391	0.381	0.466	**0.455**
LS10_HE200	32	0.383	**0.426**	0.439	0.436
TS7_AG200	128	0.376	0.371	0.449	0.435
LS10_VA300	64	0.444	0.421	0.496	0.429
KN5_VA200	32	0.328	0.364	0.425	0.427

Table 5: Graph Attention Network (GAT): The top-5 performing variations of sparsification/coarsening for different number of layers. Use of one single feature (node degree).

Sparsification/Coarsening	Embeddings dimensionality	Train F1-Macro	Test F1-Macro	Train F1-Weighted	Test F1-Weighted
1 GIN layer					
LOC10_AG300	32	0.213	0.213	0.299	0.298
LS10_HE200	256	0.213	0.213	0.299	0.298
LS10_VA100	128	0.217	0.213	0.302	0.298
LS10_VA300	128	0.213	0.213	0.299	0.298
TS7_AG200	128	0.208	0.208	0.284	0.284
2 GIN layers					
LS15_AG300	128	0.349	0.396	0.441	**0.492**
TS10_HE200	64	0.329	**0.401**	0.400	0.476
TS9_VA300	32	0.293	0.285	0.390	0.381
LS15_AG200	64	0.297	0.257	0.392	0.355
LS10_HE200	128	0.258	0.249	0.339	0.333

Table 6: Graph Isomorphism Network (GIN): The top-5 performing variations of sparsification/coarsening for different number of layers. Use of one single feature (node degree).

Sparsification/Coarsening	F1-Macro (with degree)	F1-Macro (with all)	F1-Weighted (with degree)	F1-Weighted (with all)
GCN models				
LS15_AG200	0.467	**0.527**	0.512	**0.547**
TS7_AG200	0.420	**0.457**	0.498	**0.519**
TS7_HE200	0.420	**0.492**	0.498	**0.545**
GAT models				
TS7_HE200	**0.381**	0.302	**0.455**	0.282
LS10_HE200	**0.426**	0.212	**0.436**	0.298
TS7_AG200	**0.371**	0.366	**0.435**	0.352
GIN models				
LS15_AG300	**0.396**	0.218	**0.492**	0.321
TS7_HE200	**0.401**	0.256	**0.476**	0.331
TS9_VA300	0.285	**0.403**	0.381	**0.499**

Table 7: The top-3 performing variations of sparsification/coarsening for GCN, GAT and GIN. Comparison of the use of one single feature (with degree) vs. all 5 node features (with all).

GNN Model	# layers	Train F1-Macro	Test F1-Macro	Train F1-Weighted	Test F1-Weighted
With degree					
GCN	2	0.330	0.362	0.428	0.461
GAT	2	0.348	0.390	0.446	0.427
GIN	2	0.290	0.267	0.387	0.369
With all					
GCN	2	0.332	0.343	0.431	0.445
GAT	1	0.341	0.343	0.439	0.388
GIN	2	0.255	0.266	0.355	0.367

Table 8: Performance of top GNN models without applying graph reduction.

Sparsification/Coarsening	Embeddings dimensionality	Train F1-Macro	Test F1-Macro	Train F1-Weighted	Test F1-Weighted
GCN models					
KN10_VA100	128	0.676	**0.630**	0.696	**0.647**
TS7_HE200	32	0.482	0.549	0.537	0.596
TS9_VA300	128	0.389	0.493	0.421	0.552
GAT models					
TS9_VA300	32	0.399	0.403	0.471	0.486
LOC15_VA300	64	0.405	0.419	0.446	0.486
LS10_VA100	32	0.359	0.494	0.402	0.486
GIN models					
TS9_VA300	32	0.334	0.404	0.424	0.499
LS5_VA200	64	0.319	0.359	0.414	0.454
TS7_AG200	32	0.455	0.378	0.497	0.453

Table 9: The top-3 performing configurations of `Dataset2Graph` when using all node features (with all).