SPIA 2019 Group 3

AutoML for Clustering

by

Wen Yan Wong, Yungi Jeong,
Chan Jung, Seungjun Lee

Advised by

Professor name name

Submitted in partial fulfillment
of the requirements for SPIA 2019

Date of submission: August 21, 2019

Contents

Abstract

This paper presents preliminary research into the topic of *automated machine learning* (AutoML); an overview of the methodology we used to create `autocluster`, an open source library for the automated clustering of datasets; a comprehensive set of tests demonstrating the functionality of `autocluster`; and finally, suggestions for further development and research.

Acknowledgements

This project would not be possible without the support of many organizations and individuals.

We would like give our thanks to the Math Department of Seoul National University for their warm hospitality; the Math Department of the Hong Kong University of Science and Technology for their efficacy in organizing this program; our academic mentors Sunghyun Hong, Seungro Joo, Jooyeon Chung from Samsung SDS, Gwanghyun Choi from SNU, for their guidance; Prof Otto van Koert for his insightful suggestions; and finally, all the other participants of the SPIA 2019 program for their support and camaraderie in undertaking this project.

1 Introduction

1.1 Overview

From the beginning, computer scientists have wondered whether computers can be "made to learn" - whether it is possible to create automated, self-improving systems without explicit human instructions [28]. The field of machine learning emerged from this goal.

While machine learning has earned a significant number of achievements over the years - the highest profile of which being AlphaGo's triumph over Lee Sedol [36] - they are still far from being fully automated. Each algorithm performs only a single function. For each new task, it is necessary for human experts to manually perform the difficult tasks of feature engineering, model selection, and hyperparameter tuning [30]. The goal of fully automated machine learning is still not within our grasp.

As the development and research of machine learning technologies has matured, the focus of research has shifted. Previous research involved pioneering new machine learning structures and algorithms to solve different problems [23][18]. One of the newer topics in machine learning to emerge is Automated Machine Learning (AutoML), which aims to help us automatically optimize existing solutions [22][38].

While there has already been significant research into AutoML for *supervised* learning problems - i.e. classification and regression - there is very little progress into *unsupervised* learning problems - like clustering. Clustering is a form of unsupervised machine learning - learning from unlabeled data. Clustering algorithms attempt cluster points of data into similar groups, so that a data scientist may glean some insight into the structure of the points in a dataset.

1.2 Objectives

As part of the SPIA research program, we were tasked with exploring the field of AutoML. The end goal of this project is to develop a library in Python which performs automated clustering of datasets. Specifically, our project focuses on adopting existing hyperparameter optimization methods (eg. Random Search, Bayesian Optimization), as well as techniques from the field of Meta-learning (eg. Warmstarting) to perform automated clustering.

This paper presents our research into the field of AutoML - in particular, that of the CASH problem - and the field of clustering. We define the problem statement of an automated clustering library; we discuss the flow structure, technologies, and challenges involved with developing said automated clustering library; and finally we present open source Python library for automating clustering. We present the empirical results of an experiment to evaluate the performance of our clustering library, followed by a discussion into the limitations, as well as further research questions and directions.

1.3 Literature Survey

1.3.1 Automated Machine Learning

In surveys of AutoML research produced by [42][30][10], they define AutoML as producing maximum performance from learning tools without human assistance [30].

The surveys breakdown the supervised machine learning pipeline into 2 sections: feature engineering - which includes data preprocessing, feature extraction, and feature selection; and model building - which involves algorithm selection and hyperparameter tuning. The goal of AutoML is to automate a portion of this pipeline.

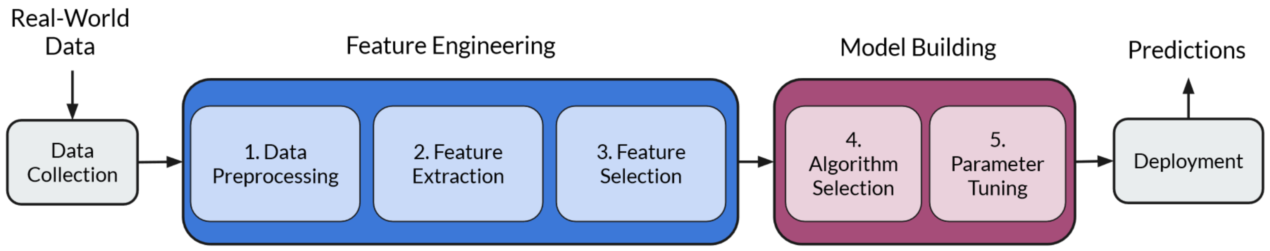

Figure 1: Supervised Machine Learning Pipeline [10]

There exists research in automating each step of the pipeline [22][24][38], however, to stay within the scope of our project this literature review will only focus on the problem of model building, i.e. Combined Algorithm Selection and Hyperparameter optimization (CASH).

1.3.2 CASH - Combined Algorithm Selection and Hyperparameter optimization

After performing feature engineering, an algorithm needs to be selected, and corresponding hyperparameters of the model needs to be tuned to yield best performance [30]. Each model has specific strengths and weaknesses on various datasets, and each hyperparameter affect how well these models can learn from the data. Due to the similarity of these two tasks, Feurer, Klein, Eggensperger, *et al.* [12] have formulated them into a unified optimization problem known as the CASH problem:

Definition 1 (CASH): *Let* $\mathcal{A} = \{A^{(1)}, A^{(2)}, \dots, A^{(R)}\}$ *be a set of algorithms, and let the hyperparameters of each algorithm* $A^{(j)}$ *have domain* $\Lambda^{(j)}$*. Further, let* $D_{train} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ *be a training set which is split into* K *cross-validation folds* $\{{D_{train}}^{(1)}, {D_{train}}^{(2)}, \dots, {D_{train}}^{(K)}\}$ *and* $\{{D_{valid}}^{(1)}, {D_{valid}}^{(2)}, \dots, {D_{valid}}^{(K)}\}$ *such that* ${D_{train}}^{(i)} = D_{train} \setminus {D_{valid}}^{(i)}$ *for* $i = 1, \dots K$*. Finally, let* $\mathcal{L}(A^{(j)}, \lambda, {D_{train}}^{(i)}, {D_{valid}}^{(i)})$ *denote the cost that algorithm* $A^{(j)}$ *achieves on* ${D_{valid}}^{(i)}$ *when trained on* ${D_{train}}^{(i)}$ *with hyperparameters* $\lambda \in \Lambda^{(j)}$*. Then the CASH problem is to find the joint algorithm and hyperparameter settings that minimize the loss:*

$$A^*, \lambda^* \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{K} \sum_{n=1}^{K} \mathcal{L}(A^{(j)}, \lambda, {D_{train}}^{(i)}, {D_{valid}}^{(i)})$$

Simple Search Approaches

Simple search approaches are CASH optimization methods which take a naive understanding of the search space. They make no assumptions about the nature of the search space, assuming each individual configuration is completely independent - i.e. the results of new configurations is independent of the structure of the training data and the results of previous configurations [42]. Simple search consists of the following:

Grid Search - One of the simplest forms of hyperparameter optimization, grid search splits the search space uniformly into a grid, and evaluates them one at a time, until all configurations are exhausted or a pre-defined cap is reached. The best performing configuration is returned.

Random Search - Instead of splitting the search space into a grid, random search instantiates each configuration randomly. Similar to grid search, random search will evaluate configurations until the configurations are exhausted or a pre-defined cap is reached. The best performing configuration is returned.

Both simple approaches have the benefit of being easy to implement, easy to parallelize, however random search is described as being the better algorithm, due to it taking advantage of hyperparameter hierarchy [4].

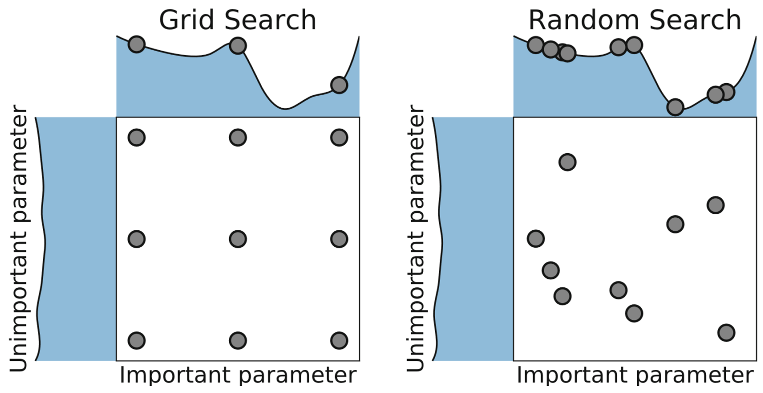

Figure 2: Visualization of how random search better explores important hyperparameters

The limitations of these simple search approaches should be obvious once we consider the sheer number of existing algorithms and the complexity of hyperparameters. Scikit-learn [29] implements more than 30 different classification algorithms, with a total of 250 hyperparameters between them. A complete search of all possible configurations is simply impossible. On large datasets, an evaluation of a single configuration may take hours to days to complete, making these approaches impractical for large scale systems.

Bayesian Optimization

While random search has the benefit of being parallelizable, it spends a significant amount of time evaluating poor configurations. The is because for most configuration spaces of CASH problems, high quality configurations are not distributed independent of position in the search space [35]. Within the search space there exists regions of high quality configurations, which is a property that grid search and random search fail to exploit.

Bayesian Optimization [35] is a blackbox optimization algorithm which constructs a probabilistic model that utilizes all of the information available from previous evaluations to make better decisions on which configurations to search next. This allows Bayesian Optimization to find the optimum value of non-convex functions with fewer evaluations [38].

There are three main components in the Bayesian Optimization framework, namely:

- Objective function: This is the *blackbox function* which is expensive to evaluate.
- Surrogate model: This model seeks to *learn* the objective function from past evaluations.
- Acquisition function: This function *proposes* new configurations in the search space to be evaluated.

Algorithm 1 provides an illustration of how Bayesian Optimization works conceptually:

Algorithm 1 Bayesian Optimization

```
$N \leftarrow$ number of iterations
cs $\leftarrow$ search space of CASH optimization
cfg_ls $\leftarrow$ list of evaluated configurations in cs
f(cfg) $\leftarrow$ objective function which evaluates a configuration cfg from cs
g(cfg) $\leftarrow$ surrogate model which gives an estimate of a configuration cfg from cs
$\alpha$(cfg,cfg_ls) $\leftarrow$ acquisition function cfg from cs

for $i = 1$ to $i = N$ do
  new_cfg $\leftarrow \underset{\texttt{cfg}\in\texttt{cs}}{\operatorname{argmax}}\ \alpha(\texttt{cfg}, \texttt{cfg\_ls})$
  cfg_list.append(new_cfg, f(new.cfg))
end for
```

A Gaussian Process (GP) [31] is typically used as the *surrogate model* to learn the objective function f:

$$p(f) = GP(f; \mu, K)$$

where μ and K are the mean vectors and covariance matrices, respectively. The GP becomes a better model of the objective function f with more observations. For instance, let $D = (X, f)$ be the set of observations thus far, then:

$$p(f|D) = GP(f; \mu_{f|D}, K_{f|D})$$

For Bayesian Optimization to be worth running, evaluating the acquisition function $\alpha(x)$ must be inexpensive compared to evaluating the expensive blackbox function f. One common choice of acquisition function is *expected improvement* (EI). Let $f^{'}$ be the minimal value of f observed so far. Expected improvement finds the point that f improves upon $f^{'}$ the most, which corresponds to the utility function:

$$u(x) = max(0, f^{'} - f(x))$$

The expected improvement (EI) acquisition function is as follows:

$$a_{EI}(x) = E[u(x)|x, D] = \int_{-\infty}^{f'} (f' - f)\mathcal{N}(f; \mu(x), K(x,x))df$$

$$= (f' - \mu(x))\Phi(f'; \mu(x), K(x,x)) + K(x,x)\mathcal{N}(f'; \mu(x), K(x,x))$$

In each iteration, the point in the configuration space which maximizes expected improvement (EI) is selected as the next candidate point to be evaluated. Expected improvement of a point is governed by two components. It can be increased by reducing the mean function $\mu(x)$, or by increasing the variance $K(x,x)$. These two terms can be seen as *exploitation* (evaluating at points at low mean) and *exploration* (evaluating at points with high uncertainty). The degree of trade-off between exploration and exploitation needs to be configured carefully, as excessive exploration can lead to inefficient optimization, while too much exploitation leaves the protocol open to strong initial biases, and a high chance of getting stuck in a local minimum.

1.3.3 Meta-Learning

When a data scientist tests algorithms and hyperparameters, they do not start from scratch and test all variants. Rather, they will test promising configurations they have seen work well in the past. This idea - applying learning gained from solving previous tasks to solve future tasks - is key to meta-learning.

Feurer, Springenberg, and Hutter [13] defined meta-learning as: a method to suggest good configurations for a novel dataset based on configurations that are known to perform well on similar, previously evaluated, datasets. To achieve this, they devised *warmstarting*.

Warmstarting begins with the definition a set of dataset *metafeatures*, based on the properties of the dataset. Using these metafeatures, they can evaluate how similar two datasets are. Thus, for a novel dataset, we can collect a list of configurations which have worked well for similar datasets. We will discuss implementaion details in the methodology section.

One of the key suggestions made by Feurer et al. is that warm starting may help us improve the performance of Bayesian Optimization, improving the rate of performance improvement, as well as the final performance quality.

2 Methodology

2.1 Problem definition

Our proposed `autocluster` framework aims to solve the *Combined Algorithm Selection and Hyperparameter optimization* (CASH) problem for *clustering tasks.* The CASH problem was formulated by Feurer, Klein, Eggensperger, *et al.* [12] as follows:

Recap of definition of CASH: *Let $\mathcal{A} = \{A^{(1)}, A^{(2)}, \dots, A^{(R)}\}$ be a set of algorithms, and let the hyperparameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$. Further, let $D_{train} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{{D_{train}}^{(1)}, {D_{train}}^{(2)}, \dots, {D_{train}}^{(K)}\}$ and $\{{D_{valid}}^{(1)}, {D_{valid}}^{(2)}, \dots,$ ${D_{valid}}^{(K)}\}$ such that ${D_{train}}^{(i)} = D_{train} \setminus {D_{valid}}^{(i)}$ for $i = 1, \dots K$. Finally, let $\mathcal{L}(A^{(j)}, \lambda, {D_{train}}^{(i)}, {D_{valid}}^{(i)})$ denote the cost that algorithm $A^{(j)}$ achieves on ${D_{valid}}^{(i)}$ when trained on ${D_{train}}^{(i)}$ with hyperparameters $\lambda \in \Lambda^{(j)}$. Then the CASH problem is to find the joint algorithm and hyperparameter settings that minimize the loss:*

$$A^*, \lambda^* \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{K} \sum_{n=1}^{K} \mathcal{L}(A^{(j)}, \lambda, {D_{train}}^{(i)}, {D_{valid}}^{(i)})$$

The `autocluster` framework aims to solve this CASH optimization problem with respect to clustering tasks. Therefore, the goal is to find the optimal choices of *dimension reduction algorithm*, *clustering algorithm*, and values of the *hyperparameters* of the chosen algorithms.

For the remaining sections of this report, we will refer to a single setting of $\mathrm{A}^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}$ as a single *configuration* from the search space. *ie. The goal of CASH optimization is to find the **optimal configuration** from the search space which minimizes a chosen objective function.*

2.1.1 Search space

Our `autocluster` clustering framework is built on top of the `sklearn` package [5]. Thus, the search space of CASH optimization used by `autocluster` includes most of the clustering and dimension reduction algorithms available in `sklearn`.

Tables 1 and 2 consist of a full list of algorithms and hyperparameters currently supported by `autocluster`.

Method	Hyperparameters
TSNE	`n_components` (integer) `perplexity` (float) `early_exaggeration` (float)
PCA	`n_components` (integer) `svd_solver` (categorical) `whiten` (categorical)
Incremental PCA	`n_components` (integer) `whiten` (categorical) `batch_size` (integer)
Kernel PCA	`n_components` (integer) `kernel` (categorical)
Fast ICA	`n_components` (integer) `algorithm` (categorical) `fun` (categorical) `whiten` (categorical)
Truncated SVD	`n_components` (integer) `algorithm` (categorical)
Latent Dirichlet Allocation(LDA)	`n_components` (integer) `learning_method` (categorical)

Table 1: Dimension reduction algorithms included in the search space of CASH optimization in `autocluster`.

Method	Scalabillity	Hyperparameters
K-Means [2]	Not scalable	`n_clusters` (integer)
Mini Batch K-Means [34]	Very large `n_samples`, medium `n_clusters`	`n_clusters` (integer) `batch_size` (integer)
Affinity Propagation [17]	Not scalable with `n_samples`	`damping` (float) `affinity` (categorical)
Mean Shift [8]	Not scalable with `n_samples`	`bin_seeding` (categorical) `bandwidth` (float)
Spectral Clustering [39]	Medium `n_samples`, small `n_clusters`	`n_clusters` (integer) `eigen_solver` (categorical) `affinity` (categorical) `assign_labels` (categorical)
Agglomerative Clustering	Large `n_samples` and `n_clusters`	`n_clusters` (integer) `eigen_solver` (categorical) `affinity` (categorical)
DBSCAN [11]	Very large `n_samples`, medium `n_clusters`	`eps` (float) `min_samples` (integer)
OPTICS [1]	Very large `n_samples`, large `n_clusters`	`min_samples` (integer) `metric` (categorical) `cluster_method` (categorical)
Birch [41]	Large `n_samples` and `n_clusters`	`n_clusters` (integer) `branching_factor` (integer)
Gaussian Mixture	Not scalable	`n_components` (integer) `covariance_type` (categorical) `init_params` (categorical) `warm_start` (categorical)

Table 2: Clustering algorithms included in the search space of CASH optimization in `autocluster`.

2.1.2 Objective functions

In *supervised learning* tasks such as regression and classification, the goal is to minimize an objective function, such as the root mean squared error (RMSE) or softmax function [26]. The objective function's value is computed using predictions from the supervised learning model and labels of the dataset.

In an *unsupervised learning* task such as clustering, the input dataset is typically unlabeled. This implies that the objective function's value can only be computed using the outputs of a clustering model (ie. the labels assigned by clustering model onto each datapoint). This raises the question: "How do we evaluate the *goodness* or *quality* of a clustering result?".

In the `autocluster` framework, the following evaluation metrics, available on from the `sklearn` [29] package, have been adopted as objective functions for the CASH optimization problem:

Metric	Range	Normalized Form
Silhouette Coefficient [33]	$-1 \leq x \leq 1$	$(1-x)/2$
Davies-Bouldin Index [9]	$x \geq 0$	$\tanh(x)$
Calinski-Harabasz Index [6]	$x \geq 0$	$1-\tanh(x)$

Table 3: Evaluation metrics adopted as objective functions.

Based on the `sklearn`'s official documentation [29], these metrics have a common weakness, which is that they tend to favor convex clusters than other concepts of clusters, such as density based clusters like those obtained from DBSCAN. This implies that a CASH optimization using these metrics as objective function will likely result in convex-based clustering algorithms (eg. KMeans, Gaussian Mixture) being selected as the optimal configuration.

2.2 Blackbox optimization algorithms

In the `autocluster` package, we included 3 choices of blackbox optimization strategies to optimize the hyperparameters of clustering algorithms: *Random Optimization (RO)*, *Bayesian Optimization (SMAC)* and *Bayesian Optimization (SMAC) + Warmstarting (Meta-learning)*.

2.2.1 Random Optimization (RO)

Random Optimization (RO) was included in the `autocluster` package as a lightweight optimizer because it is easy to implement, and converges faster than grid search [4].

Algorithm 2 Random Optimization (RO)

```
$N \leftarrow$ number of iterations
cs $\leftarrow$ search space of CASH optimization
f() $\leftarrow$ objective function which evaluates a configuration cfg from cs
opt_cfg $\leftarrow$ best configuration so far

for $i = 1$ to $i = N$ do
  new_cfg $\leftarrow$ cs.sample_new_configuration()
  if f(new_cfg) $<$ f(opt_cfg) then
    opt_cfg $\leftarrow$ new_cfg
  end if
end for
```

In our actual implementation, an additional parameter `cutoff_time` was included to control the overall running time of the RO optimizer. If a configuration which takes more than `cutoff_time` seconds to be evaluated, the configuration will be ignored and the optimizer will proceed to the next iteration.

In our experiments, we used RO (Algorithm 2) as a baseline optimizer for comparison (see section 3.1).

2.2.2 Bayesian Optimization (SMAC)

We utilised the Sequential Model-based Algorithm Configuration (SMAC) package developed by Hutter, Hoos, and Leyton-Brown [19] to perform Bayesian Optimization in the `autocluster` package. SMAC uses a Random

Forest [25] as a surrogate model in Bayesian Optimization. In the optimization settings, we used Expected Improvement (EI) as the acquisition function (see section 1.3.2 for details).

2.2.3 Bayesian Optimization (SMAC) + Warmstarting (Meta-learning)

As shown by Feurer, Springenberg, and Hutter [13], the performance of Bayesian Optimization can be improved significantly if the initial configurations evaluated by the optimizer are promising and 'close' to the optimal configuration. In order to speed up the convergence rate of `autocluster.fit()`, we adopted a similar approach by using a warmstarter (pretrained via a meta-learning procedure) to recommend promising initial configurations to the SMAC optimizer.

KDTree Warmstarter

The warmstarter is a nearest neighbors model which stores a list of benchmark datasets in memory. For each benchmark dataset, the corresponding top n best configurations (obtained during pre-training) are also stored in memory. When given a new dataset, characterised by a metafeatures vector (see section 1.3.3), the warmstarter identifies the top k most similar benchmark datasets in memory (ie. closest in terms of euclidean distance in the metafeatures vector space, see Figure 3). Then, the top n configurations from each of the top k 'closest' benchmark datasets are retrieved, yielding a total of $n \times k$ initial configurations to warmstart the SMAC optimizer.

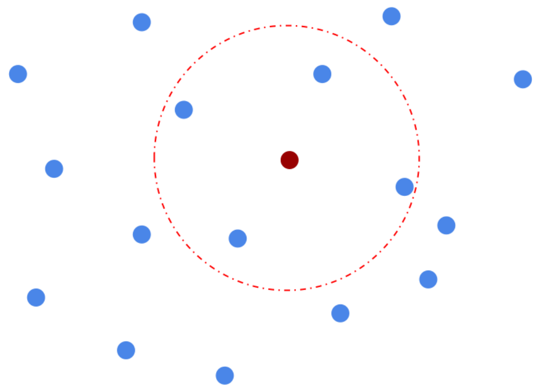

Figure 3: Graphical example of the metafeatures vector space. Red point represents the new dataset, while blue points represent benchmark datasets in memory. In this example, the warmstarter retrieves initial configurations from the top $k = 4$ most similar benchmark datasets.

Internally, the warmstarter implemented in `autocluster` is a KDTree nearest neighbors model (available in `sklearn` package [37]). We selected the KDTree data structure [3] instead of brute force approaches (eg. K-nearest-neighbors) due to its more efficient asymptotic runtime complexity ($\mathcal{O}(n \log n)$ as opposed to $\mathcal{O}(n^2)$) during query.

Pre-training the warmstarter

To pretrain the warmstarter, ideally, a variety of benchmark datasets will be chosen so that they are almost uniformly distributed throughout the metafeatures space. Bayesian Optimization (SMAC) is performed on each dataset for a large number of iterations (to ensure convergence), and the top n configurations on each dataset are saved for later use (warmstarting).

Metafeatures

As mentioned in section 1.3.3, metafeatures are a set features that can be used to characterize a dataset so that similarities between different datasets can be quantified. In our `autocluster` framework, we categorized metafeatures into 3 distinct classes:

- General: These features mainly capture the dimensionality, scale, complexity and noisiness of the data. See Table 4 for details.
- Numeric: These features focus on the columns with numeric values. See Table 5 for details.
- Categorical: These features focus on the columns with categorical values. See Table 6 for details.

Features	Description	Variants
Number of Instances		log
Number of Features		log, ratio to instances
Number of Missing Values		% missing
Sparsity	$\frac{\#\{\text{missing values \& zeros}\}}{\#\{\text{all values}\}}$	

Table 4: General metafeatures for the warmstarter

Features	Description	Variants
Sparsity	$\frac{\#\{\text{zeros}\}}{\#\{\text{numerical values}\}}$	
Skewness	Asymmetry of probability distribution	min, max, median, mean, 1st-quartile, 3rd-quartile
Kurtosis	Tailedness of probability distribution	
Correlation	Dependency between two variables	
Covariance	Joint variability of two variables	
PCA	Principal component analysis	fraction 95%, kurtosis, skewness

Table 5: Numerical metafeatures for the warmstarter

Features	Description	Variants
Entropy	Unpredictability of the state of average information	min, max, median, mean, 1st-quartile, 3rd-quartile

Table 6: Categorical metafeatures for the warmstarter

2.3 Overview of whole system

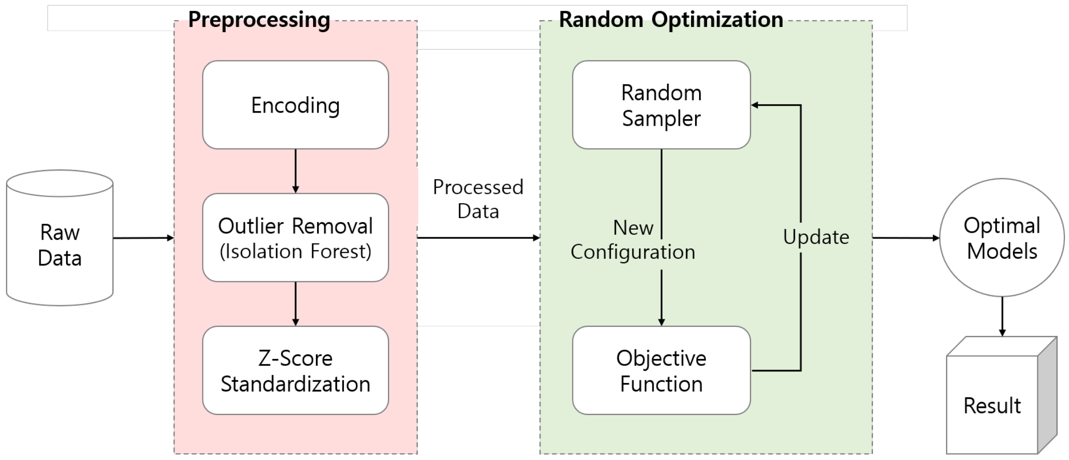

Figure 4: Flow chart of how `autocluster.fit()` with Random Optimization (RO) works.

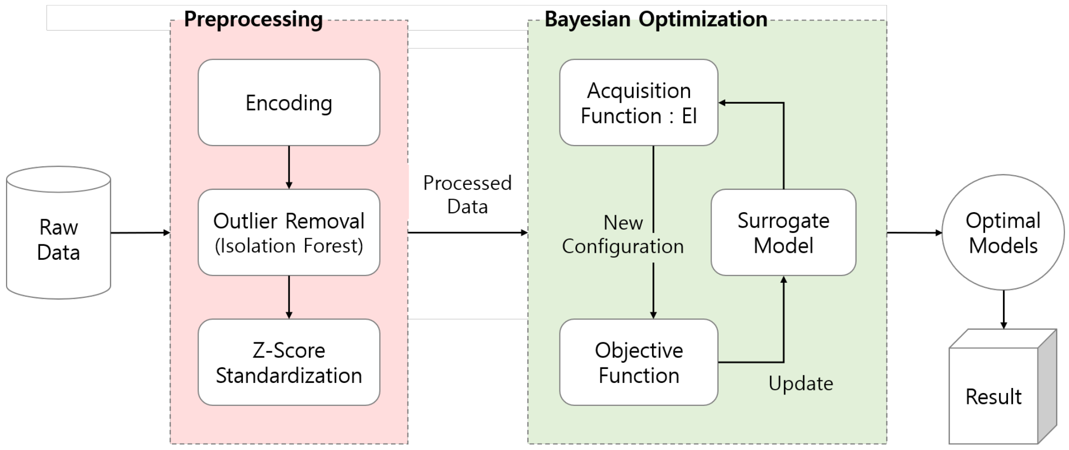

Figure 5: Flow chart of how `autocluster.fit()` with Bayesian Optimization (SMAC) works.

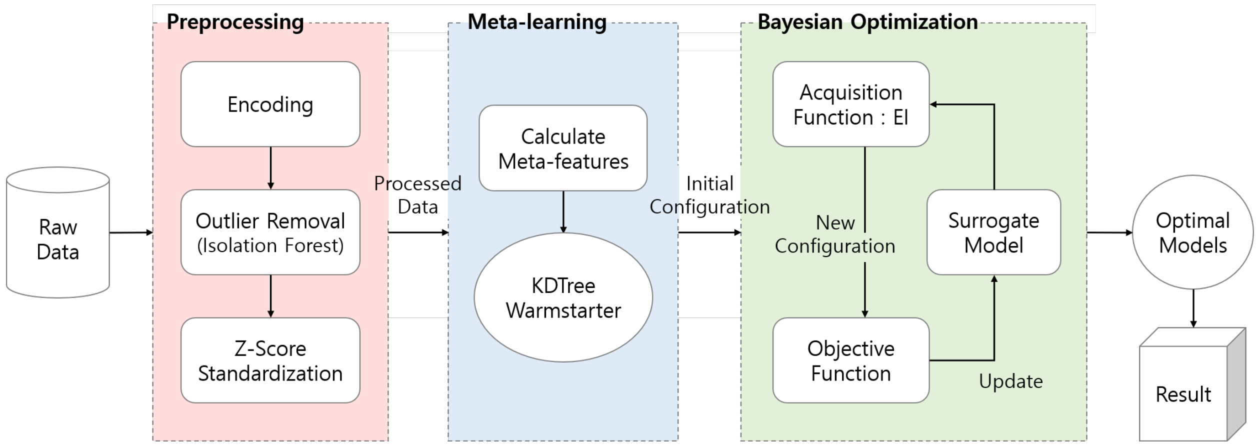

Figure 6: Flow chart of how `autocluster.fit()` with Bayesian Optimization (SMAC) + Warmstarting works.

3 Evaluation

3.1 Experimental setup

To evaluate the performance of `autocluster.fit()`, we conducted experiments to compare the convergence rate of the 3 blackbox optimization strategies presented in section 2.2.

1. **Datasets**: We used a total of 120 synthetic clustering datasets available from Fränti and Sieranoja [16]'s webpage. More specifically, these datasets are comprised of
   - S-sets [14]: Synthetic 2-dimensional data with $N = 5000$ vectors and $k = 15$ Gaussian clusters with varying degrees of cluster overlap.
   - A-sets [20]: Synthetic 2-dimensional data with increasing number of clusters. There are 150 vectors per cluster.
   - Birch-sets [40]: Synthetic 2-dimensional data with $N = 100,000$ vectors and $k = 100$ clusters.
   - G2 sets [27]: Gaussian clusters datasets with varying cluster overlap and dimensions.
   - DIM-sets (high) [15]: High-dimensional datasets $N = 1024$ and $k = 16$ Gaussian clusters.
   - DIM-sets (low) [21]: Low-dimensional synthetic data with Gaussian clusters.
   - Unbalance [32]: Synthetic 2-dimensional data with $N = 6500$ vectors and $k = 8$ Gaussian clusters.
2. **Train-test split**: Since the Bayesian Optimization (SMAC) + Warmstarting approach requires a pre-trained warmstarter, we performed a train-test split on the 120 synthetic clustering datasets:
   - Test: 20 datasets (approx. 16.7%) were randomly selected from the pool of 120 clustering datasets.
   - Train: The remaining 100 datasets were used to pretrain the warmstarter.
3. **Metafeatures for warmstarter**:
   - Since the benchmark datasets chosen for this experiment only consist of numerical values, we only used general and numeric metafeatures (see Table 4 and Table 5) to construct the metafeatures space.
4. **Blackbox optimization algorithms**:
   - Random Optimization (RO): Details in Figure 4
   - Bayesian Optimization (SMAC): Details in Figure 5
   - Bayesian Optimization (SMAC) + Warmstarting : Details in Figure 6
5. **Search space of optimization algorithms**:
   - All dimension reduction models (Table 1), clustering models (Table 2), and their corresponding hyperparameters were included in the configuration space.
   - In addition, dimension reduction was implemented as an *optional* procedure because some benchmark datasets are already low dimensional (eg. S-sets and A-sets).
6. **Optimization parameters (Testing)**:
   - Number of iterations: 100. This is also equivalent to the number of evaluations.

- Cutoff time: 100 seconds. This is the maximum time allowed for evaluating a single configuration.
- Number of folds (k-fold cross validation loss): 3.
- Number of initial configurations from warmstarter: 25. This is only applicable for the SMAC + Warmstarting approach.

7. **Optimization parameters (Pre-training of warmstarter)**:
   - Number of iterations: 200
   - Cutoff time: 1000 seconds.
   - Number of folds (k-fold cross validation loss): 3
8. **Objective function**: We used the normalized form of Silhouette Coefficient (see Table 2.1.2) as the loss function to be minimized. In addition, we included several additional conditions which, if fulfilled, results in a $+\infty$ loss being returned:
   - `number_of_clusters_identified == 1`:
     - To encourage the optimizer to segregate the data points and uncover more patterns.
   - $\dfrac{\#\{\text{points in smallest cluster}\}}{\#\{\text{points in total}\}} < 0.01$:
     - To discourage the optimizer from favoring clustering results with extremely small clusters.
   - $\dfrac{\#\{\text{points in smallest cluster}\}}{\#\{\text{points in largest cluster}\}} < 0.05$:
     - To discourage the optimizer from favoring clustering results with extremely small clusters.

3.2 Empirical results

3.2.1 Convergence rate

In this section, we present the results of the experiment described in 3.1.

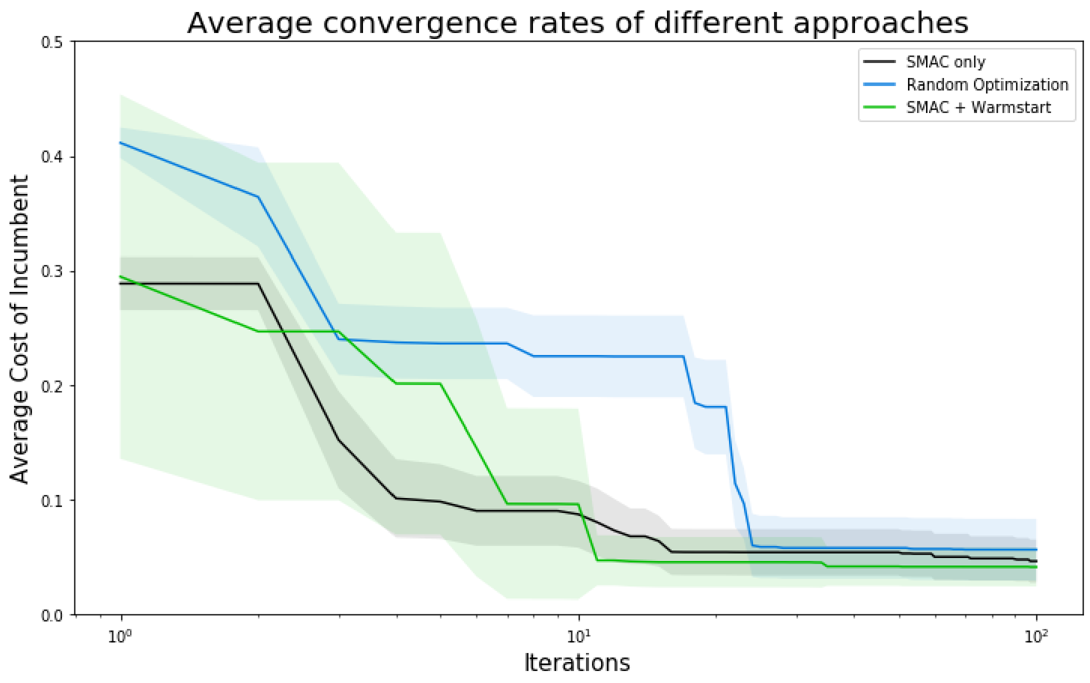

Figure 7: The average cost of incumbent at i-th iteration is computed by averaging the cost of incumbent at i-th iteration on each of the 20 testing datasets. The colored region is a representation of the standard deviation of the convergence curve at i-th iteration.

From Figure 7, it is clear that random optimization has the slowest rate of convergence among the three algorithms. On average, SMAC + Warmstart outperforms SMAC alone, but its convergence rate at the beginning suffers from high variance. This can be reasoned from the fact that the initial 25 configurations evaluated by SMAC + Warmstart are recommendations retrieved from the KDTree Warmstarter, which are promising configurations but may be sub-optimal in some cases. However, after this initial slump, the SMAC + Warmstart converges to a more optimal configuration.

3.2.2 Visualization of clustering results

In this section, we present some visualizations of the clustering results produced by the optimal models found by the SMAC + Warmstarting algorithm.

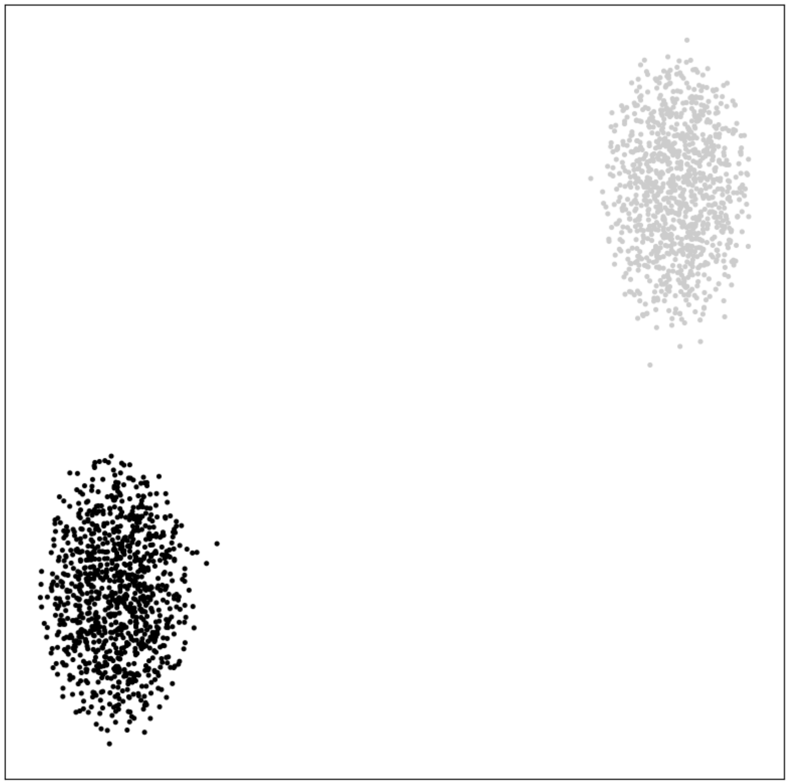
Figure 8: Clustering result on the G2 sets (see section 3.1). The dataset comprises of 2 Gaussian clusters in 16-dimensional space with $N = 2048$ points. The optimal configuration obtained by SMAC + Warmstarting consists of a TSNE dimension reduction model + KMeans clustering model with `n_clusters = 2`.

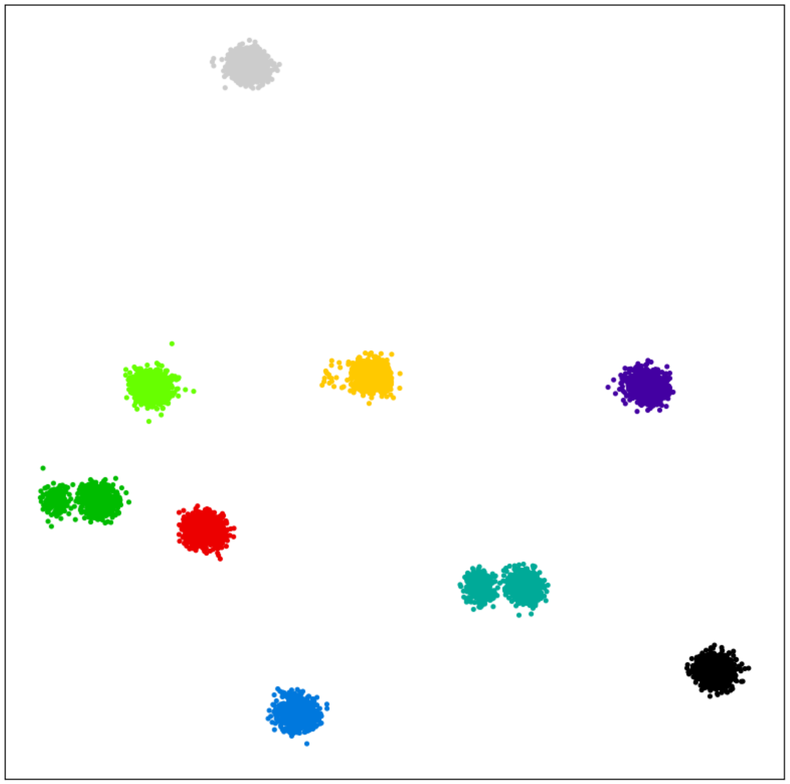
Figure 9: Clustering result on the DIM-sets (see section 3.1). The dataset comprises of 9 Gaussian clusters in 14-dimensional space with $N = 2048$ points. The optimal configuration obtained by SMAC + Warmstarting consists of a PCA dimension reduction model + Affinity Propagation clustering model.

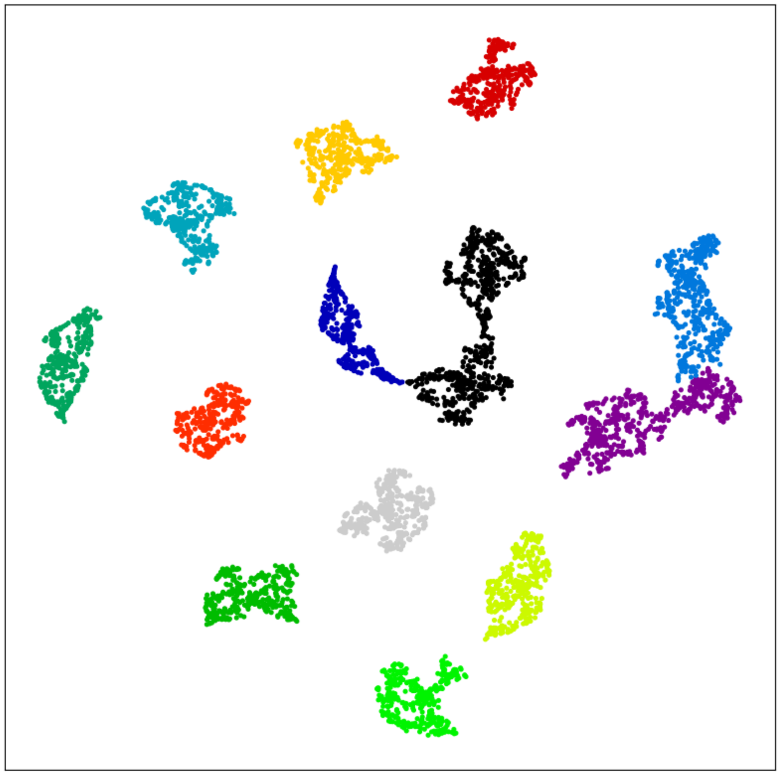

Figure 10: Clustering result on the S-sets (see section 3.1). The dataset comprises of 15 Gaussian clusters in 2-dimensional space with $N = 5000$ points. The optimal configuration obtained by SMAC + Warmstarting consists of a TSNE dimension reduction model + Agglomerative clustering model with `n_clusters = 13`.

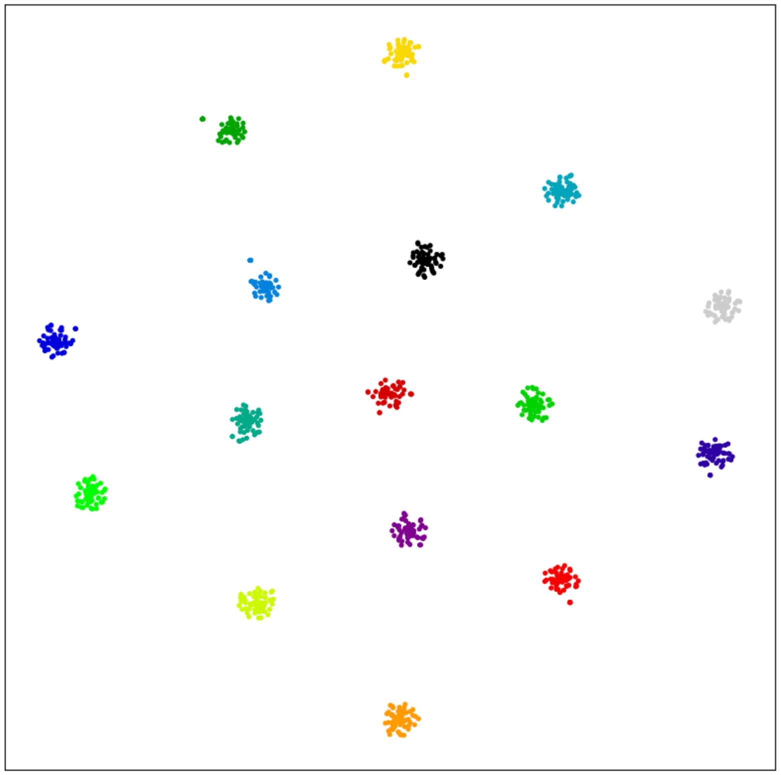

Figure 11: Clustering result on the DIM-sets (see section 3.1). The dataset comprises of 16 Gaussian clusters in 128-dimensional space with $N = 1024$ points. The optimal configuration obtained by SMAC + Warmstarting consists of a Truncated SVD dimension reduction model + Birch clustering model.

4 Discussion

4.1 Limitations and Further Research

Dimension Reduction

One of the more contestable decisions we have made is to include dimension reduction, as well as the parameters for it, as part of the optimization process.

We decided to include dimension reduction for the following reasons:

- Dimension reduction is a generally well accepted method to make training and feature identification more efficient on high dimensional datasets.
- The ability to transform non-convex datasets into a form that is easier for clustering evaluation metrics to cluster well.
- It allows for easier visualization of the dataset for the end user.

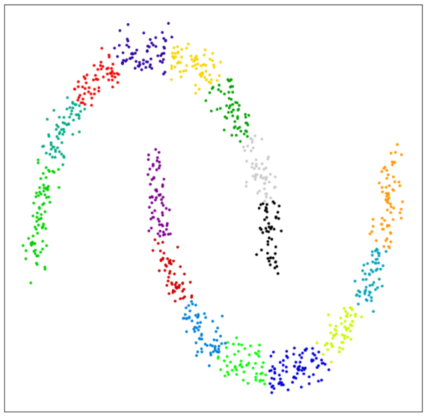

Figure 12: `autocluster` without dimensionality reduction on non-convex clusters

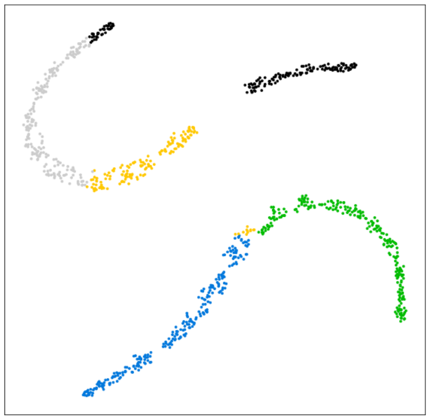

Figure 13: `autocluster` with TSNE for dimenmsion reduction on non-convex clusters for 50 iterations.

While choosing to do dimension reduction had many measurable benefits (see Figure 12, Figure 13 and Figure 14), it has also created some unintentional behaviors, when the goals of the clustering metric do not align with what we want the algorithm to do. For example, the optimization algorithm may decide to perform excessive dimension reduction even when none is necessary to ”clump” groups of data points closer together, in order to achieve higher score on our metrics.

Clustering metrics

As mentioned earlier, all of the metrics we use favor convex clustering - i.e. blobby clusters - as can be seen in the figure below. However, human intuition often clusters objects very differently, grouping points according to continuity of the cluster. This leads to perceived poor performance for non-convex clusters like noisy moons, as seen below.

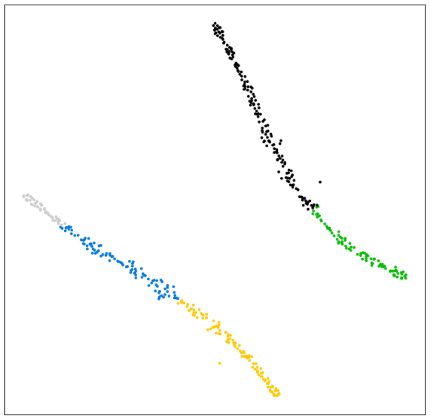

Figure 14: `autocluster` with further TSNE for dimension reduction used on non-convex clusters for 100 iterations.

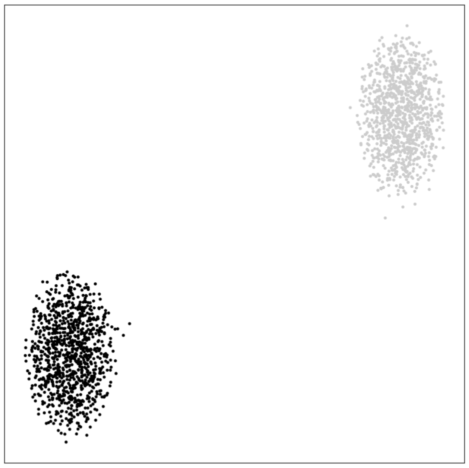

Figure 15: `autocluster` used on convex clusters

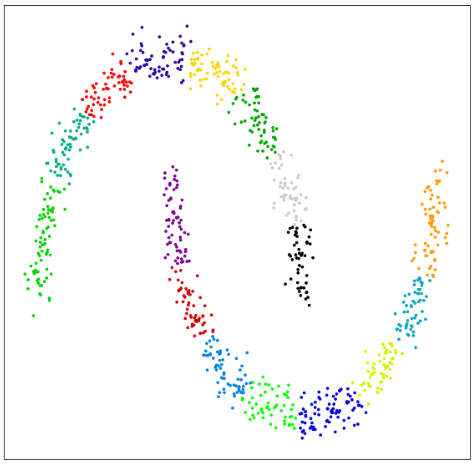

Figure 16: `autocluster` used on non-convex clusters

To fix these issues, we would need to use clustering metrics that favor continuity. While there does exist topological data analysis tools (TDA) [7] that are able to evaluate the quality of non-convex clusters, due to the technical difficulty of implementation, as well as the balooning complexity of the installation prerequisites, we decided against implementing TDA tools in this library.

Scalability

Due to the use of TSNE and the complexity of clustering algorithms like DBSCAN, for large dimensions and large datasets each time step can be painfully slow. This can affect the quality of performance, as the evaluation may exceed the given time cap (see `cutoff_time` in Section 3.1).

To mitigate this, we have considered performing sub-sampling on large datasets, then using the sub-sampled data to find best configuration. However, due to time limitations, we have not implemented this.

A more complex solution would be to incorporate multi-fidelity optimization techniques [42]. For instance, save the best performing configurations on subsamples, then fully evaluate them to find the best example.

5 Conclusion

We have successfully completed the intended objectives (section 1.2) of our project. We have developed a automated machine learning toolkit (links available in Appendix) for clustering. In our experiments, we have demonstrated that the proposed clustering algorithm (Bayesian Optimization + Meta-learning) performs well for benchmark clustering datasets (section 3.1), outperforming random search in terms of convergence rate. The algorithm works well for convex-based clusters (section 3.2.2), but further research and development may be required to deal with non-convex clusters.

6 Hardware and Software

6.1 Hardware

- CPU for SMAC optimization.
- Laptops with Ubuntu OS for development.

6.2 Software

- Ubuntu OS.
- Python.
- Jupyter Lab.
- Sublime text.
- Git.

References

[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “Optics: Ordering points to identify the clustering structure,” in *ACM Sigmod record*, ACM, vol. 28, 1999, pp. 49–60.

[2] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[3] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Jan. 1975. DOI: `10.1145/361002.361007`.

[4] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization.,” *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012. [Online]. Available: `http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.html#BergstraB12`.

[5] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: Experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[6] T. Caliński and J. Harabasz, “A dendrite method for cluster analysis,” *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[7] F. Chazal, L. J. Guibas, S. Y. Oudot, and P. Skraba, “Persistence-based clustering in riemannian manifolds,” *Journal of the ACM (JACM)*, vol. 60, no. 6, p. 41, 2013.

[8] D. Comaniciu and P. Meer, “Mean shift: A robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 5, pp. 603–619, 2002.

[9] D. L. Davies and D. W. Bouldin, “A cluster separation measure,” *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.

[10] R. Elshawi, M. Maher, and S. Sakr, “Automated machine learning: State-of-the-art and open challenges,” *arXiv preprint arXiv:1906.02287*, 2019.

[11] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *Kdd*, vol. 96, 1996, pp. 226–231.

[12] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, “Auto-sklearn: Efficient and robust automated machine learning,” *Automated Machine Learning The Springer Series on Challenges in Machine Learning*, pp. 113–134, 2019. DOI: `10.1007/978-3-030-05318-5_6`.

[13] M. Feurer, J. T. Springenberg, and F. Hutter, “Initializing bayesian hyperparameter optimization via meta-learning,” in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[14] P. Fränti and O. Virmajoki, “Iterative shrinking method for clustering problems,” *Pattern Recognition*, vol. 39, no. 5, pp. 761–765, 2006. DOI: `10.1016/j.patcog.2005.09.012`. [Online]. Available: `http://dx.doi.org/10.1016/j.patcog.2005.09.012`.

[15] P. Fränti, O. Virmajoki, and V. Hautamäki, “Fast agglomerative clustering using a k-nearest neighbor graph,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1875–1881, 2006.

[16] P. Fränti and S. Sieranoja, *K-means properties on six clustering benchmark datasets*, 2018. [Online]. Available: `http://cs.uef.fi/sipu/datasets/`.

[17] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[18] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[19] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," *Lecture Notes in Computer Science Learning and Intelligent Optimization*, pp. 507–523, 2011. DOI: `10.1007/978-3-642-25566-3_40`. [Online]. Available: `https://www.cs.ubc.ca/~hutter/papers/10-TR-SMAC.pdf`.

[20] I. Kärkkäinen and P. Fränti, "Dynamic local search algorithm for the clustering problem," Department of Computer Science, University of Joensuu, Joensuu, Finland, Tech. Rep. A-2002-6, 2002.

[21] ——, "Gradual model generator for single-pass clustering," *Pattern Recognition*, vol. 40, no. 3, pp. 784–795, 2007.

[22] A. Kaul, S. Maheshwary, and V. Pudi, "Autolearn[pleaseinsert"prerenderunicode–″intopreamble]automated feature generation and selection," in *2017 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2017, pp. 217–226.

[23] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural automl for deep learning," *arXiv preprint arXiv:1902.06827*, 2019.

[25] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: `https://CRAN.R-project.org/doc/Rnews/`.

[26] *Machine learning.* [Online]. Available: `http://cs229.stanford.edu/syllabus.html`.

[27] P. F. R. Mariescu-Istodor and C. Zhong, "Xnn graph," vol. LNCS 10029, pp. 207–217, 2016.

[28] T. M. Mitchell *et al.*, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, no. 37, pp. 870–877, 1997.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] Y. Quanming, W. Mengshuo, J. E. Hugo, G. Isabelle, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang, "Taking human out of learning applications: A survey on automated machine learning," *arXiv preprint arXiv:1810.13306*, 2018.

[31] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning.* MIT Press, 2008.

[32] M. Rezaei and P. Fränti, "Set-matching methods for external cluster validity," *IEEE Trans. on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 2173–2186, 2016.

[33] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[34] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*, ACM, 2010, pp. 1177–1178.

[35] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[36] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[37] *Sklearn.neighbors.kdtree¶*. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html#sklearn.neighbors.KDTree`.

[38] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[39] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[40] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: A new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

[41] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," in *ACM Sigmod Record*, ACM, vol. 25, 1996, pp. 103–114.

[42] M.-A. Zöller and M. F. Huber, "Survey on automated machine learning," *arXiv preprint arXiv:1904.12054*, 2019.

7 Appendix

7.1 Github repository

`https://github.com/wywongbd/autocluster`

7.2 Python package index (PyPi)

`https://pypi.org/project/autocluster/`