Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Analysis and Transfer of AutoML Concepts for Clustering Algorithms

Dennis Tschechlov

Course of Study:	Softwaretechnik
Examiner:	PD Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Manuel Fritz, M.Sc.
Commenced:	June 20, 2019
Completed:	December 20, 2019

Abstract

Data analysts are confronted with the choice of selecting an appropriate algorithm with suitable hyperparameters for datasets that they want to analyze. For this, they typically execute and evaluate many configurations in a trial-and-error manner. However, for novice data analysts this is a time-consuming task. Recent advances in the research area of AutoML address this problem by automatically find a suitable algorithm with appropriate hyperparameters. Yet, these systems are only applicable for supervised learning tasks and not for unsupervised learning.

In the scope of this work, existing AutoML systems are analyzed in detail. Subsequently, a concept is developed that uses components from existing AutoML systems but modifies them in such a way that they are applicable for unsupervised learning. Although, various kinds of unsupervised learning methods exist, this work focuses on the popular unsupervised method clustering. This concept is also prototypical implemented as proof-of-concept and is used for the evaluation. The comprehensive evaluation discusses the results for different optimization methods for selecting a suitable clustering algorithm with appropriate hyperparameters. The evaluation unveils that the predicted number of clusters of the implemented prototype deviates only slightly from the actual number of clusters. Hence, this work showed that it is possible to successfully transfer the concepts of existing AutoML systems to the unsupervised learning method of clustering and at the same time achieve precise results in an acceptable amount of time.

Kurzfassung

Datenanalysten stehen häufig vor der Wahl, einen geeigneten Algorithmus mit geeigneten Hyperparametern für Datensätze auszuwählen, die sie analysieren wollen. Dazu probieren sie typischerweise viele Konfigurationen nach der Versuch-und-Irrtum Methode aus und werten diese anschließend aus. Insbesondere für unerfahrene Datenanalysten ist dies jedoch eine zeitaufwändige Aufgabe. Die jüngsten Fortschritte auf dem Forschungsgebiet von AutoML stellen sich diesem Problem, indem sie automatisch einen geeigneten Algorithmus mit geeigneten Hyperparametern finden. Diese Systeme sind jedoch nur für betreute Lernaufgaben anwendbar und nicht für unbeaufsichtigtes Lernen.

Im Rahmen dieser Arbeit werden bestehende AutoML-Systeme detailliert analysiert. Anschließend wird ein Konzept entwickelt, das Komponenten aus bestehenden AutoML-Systemen verwendet, diese aber so modifiziert, dass sie für das unübewachte Lernen geeignet sind. Obwohl es verschiedene Arten von unpberwachten Lernmethoden gibt, fokussiert sich diese Arbeit auf die häufig verwendete Methode des Clustering. Dieses Konzept ist zudem auch prototypisch als Proof-of-Concept implementiert und wird für die Evaluation verwendet. Die umfassende Evaluation diskutiert die Ergebnisse für verschiedene Optimierungsmethoden zur Auswahl eines geeigneten Clustering-Algorithmus mit geeigneten Hyperparametern. Die Auswertung zeigt, dass die vorhergesagte Anzahl der Cluster des implementierten Prototypen nur geringfügig von der tatsächlichen Anzahl der Cluster abweicht. Damit wurde gezeigt, dass es möglich ist die Konzepte bestehender AutoML-Systeme erfolgreich auf die unüberwachte Lernmethode des Clustering zu übertragen und gleichzeitig präzise Ergebnisse in akzeptabler Zeit zu erzielen.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Acronyms

AutoML Automated machine learning. 9

BO Bayesian Optimization. 24

BOHB Bayesian Optimization and Hyperband. 24

CASH Combined Algorithm Selection and Hyperparameter optimization. 9

EI Expected Improvement. 25

EM Expectation Maximization. 29

GMM Gaussian Mixture Models. 29

HPO hyperparameter optimization. 9

kNN k-Nearest-Neighbor. 49

ML Machine learning. 23

SH Successive Halving. 26

SMAC Sequential model-based algorithm configuration. 33

SMBO sequential model-based optimization. 25

SSE sum of squared errors. 36

TPE Tree-structured Parzen Estimator. 33

1 Introduction

Nowadays, a lot of data is generated in the Big Data Era. It is reported that 2.5 quintillion byte, which is around 2 million terabytes, of data is generated every day and 90% of all data stored in the world is produced in the last two years[1]. However, the term Big Data does not only refer to a large volume of data, but also to various data sources, increased velocity of creating data and more variability in the data flows [Lan01]. But the most important aspect is that it is getting harder to analyze the data and to extract meaningful insights out of it due to the sheer size of the data.

To that end, many different processes exist to tackle the problem of extracting knowledge of raw data. Each of these processes have in common that they execute several steps in a iterative manner until the desired result is obtained. An example for such a process is the Knowledge Discovery in Databases (KDD) [FPS96] process, which is shown in Figure 1.1. This process starts by selecting the appropriate target data from the raw data. Subsequently, the target data is preprocessed and transformed into a format that is appropriate for the actual analysis step. In the Data Mining step, the data is analyzed and patterns are extracted. Afterwards, the data analyst evaluates and interprets the resulting patterns and gains new knowledge if he is satisfied with the result. However, as the grey dotted arrows show in Figure 1.1, if the result does not satisfy the analyst, he can repeat any of the aforementioned steps. He then repeats several steps of the process with different algorithms and hyperparameters until he is satisfied. This is a highly exploratory task and the success as well as

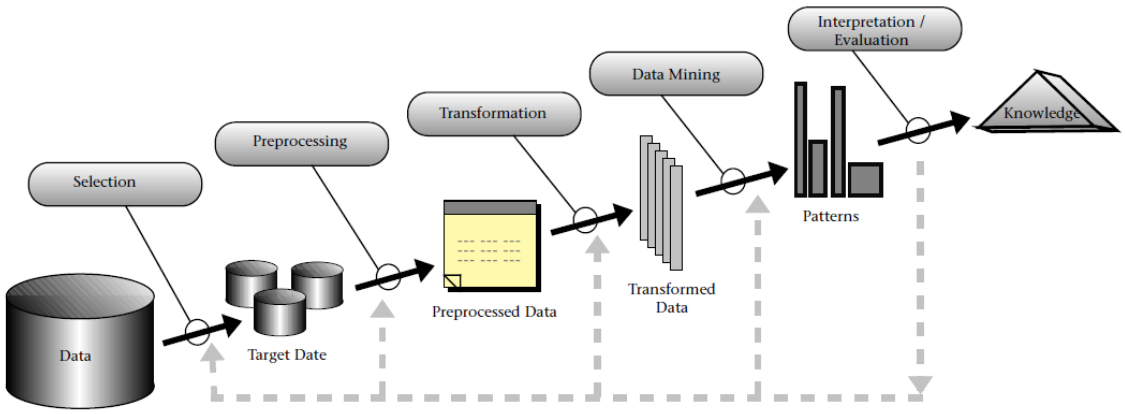


Figure 1.1: Overview of the KDD process [FPS96].

[1] https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/

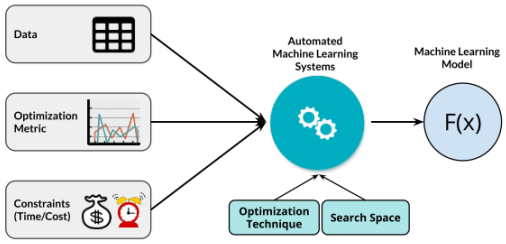


Figure 1.2: Basic overview of the functionality of AutoML systems [EMS19].

the duration of the process is based on the knowledge and experience of the analyst. The reason is that many different algorithms with various hyperparameters exist and it is not always clear how to choose an appropriate one. This results in a time-consuming exploration process.

The analysis in the data mining step is often performed using different machine learning techniques. These can generally be divided into *supervised* and *unsupervised* learning techniques. While supervised techniques have the knowledge of the expected outcome, such as class labels, unsupervised techniques do not have them.

For supervised learning techniques, recent advances in the research area of AutoML automate a large part of these steps [EMS19; FEF+18; FKE+15; THHL13]. As shown in Figure 1.2, these AutoML systems have three inputs. The first is the data, where the class labels must be predicted. The second is an optimization metric. The framework tries to optimize this metric and to find the best model according to this metric. The last input is a constraint or budget. This budget defines a limited resource that can be consumed at most to find the best model. This can be, e.g., a time constraint or the number of iterations that an algorithm can perform. Using these inputs, the AutoML system tries to find a machine learning model. To this end, the system uses an optimization technique, which searches for the best model with its hyperparameters from the search space. The search space contains all available algorithms with their possible hyperparameter values. In this context, the best model is the model that shows the best performance according to the optimization metric. The existing AutoML systems differ in the optimization technique they use [FEF+18; FKE+15; THHL13].

Even though Thornton et al. state that these concepts can be applied to unsupervised learning as well [THHL13], such systems do not exist yet. The lack of information in the training data, such as class labels, makes it more complex to transfer these concepts to clustering. Yet, the optimization techniques and hence the optimization metrics in the AutoML systems for supervised learning require this information.

The goal of this work is to analyse the concepts of the existing AutoML systems in detail and to transfer this knowledge for the unsupervised task of clustering. The contributions of this work include the following:

- Analysis of existing optimization techniques in the AutoML systems.
- Analysis of optimization metrics that can be used in the optimization techniques and to evaluate clustering results at the same time.
- Analysis of further methods to improve the optimization techniques.

- Preparation of a concept that transfers the aforementioned techniques for clustering.
- Prototypical implementation of the concept to select an algorithm and appropriate hyperparameter values for clustering.
- Evaluation of the prototypical implementation.
- Comparison to existing estimation methods that exist for clustering.

The remainder of this work is structured as follows:

Chapter 2 presents the required background for this work, such as the optimization techniques, the clustering algorithms and optimization metrics used in this work. An overview of the related work is given in Chapter 3. Chapter 4 analyzes the existing AutoML systems and describes the similarities between them. The transfer of the AutoML concepts to clustering is explained in Chapter 5. Chapter 6 shows details of the implementation of the concept. A comprehensive evaluation of the AutoML concept for clustering analyses is performed in Chapter 7. Chapter 8 concludes this work and an outlook for future work is given in Chapter 9.

2 Background

In this chapter the required background for this work is discussed. For this, the AutoML concept is first explained in Section 2.1. Then the optimization techniques that are typically used in existing AutoML systems are presented in Section 2.2. Since the goal of this work is to transfer these concepts to clustering, clustering in general, as well as the algorithms and metrics that are used in this work are explained in Section 2.3. Subsequently, Section 2.4 describes the concept of Meta-learning.

2.1 AutoML

The term AutoML refers to automating Machine learning (ML) pipelines. The goal is to help novice and inexperienced data analysts with choosing solid ML algorithms with appropriate hyperparameters. Though the term refers to automating every step of a ML pipeline, like automatic feature extraction and selection [KMP17; KSS17; KTSP17], this work focuses on the automatic algorithm and hyperparameter selection.

The goal of AutoML is, given a dataset and an optimization metric, to automatically, without human interaction, produce predictions for this dataset that optimize the metric value within a given budget. This means that an AutoML system has to automatically choose an appropriate ML algorithm as well as its hyperparamters. Thornton et al. describe this problem as CASH problem [THHL13].

An algorithm with fixed hyperparameter values is called a configuration. The values for the hyperparameters can be chosen from a *configuration space*. This contains all possible combinations of algorithms and hyperparameter values that are considered for the analysis. Since multiple classifiers are available with various hyperparameters, this results in a large configuration space. Also, the hyperparameters have to be chosen dependent on the algorithm since they can vary from algorithm to algorithm. The k-Means algorithm, for example, has the number of clusters k as hyperparameter while the DBSCAN [EKSX96] algorithm has two density-based hyperparameters (*minPts* and ϵ). Therefore, Thornton et al. formalize the CASH problem as hierarchical hyperparameter optimization problem [THHL13]. They extend the configuration space with a new root-level hyperparameter that represents the choice of an algorithm. This is a conditional hyperparameter, i.e., the actual hyperparameters are chosen depending on the algorithm [HHCB09].

The basic workflow of AutoML systems is to execute and evaluate as much configurations as possible until a predefined budget is exhausted. Subsequently, the best configuration according to a certain metric is chosen. However, the problem of a large configuration space still remains while having a limited budget. Because of that, today's AutoML systems implement elaborated methods that can search the configuration space in an effective way, such that the most promising configurations are evaluated first. These methods are actually hyperparameter optimization methods but with the hierarchical approach, as shown above, they can also be used to select an algorithm as well. These

methods are described in Section 2.2. There are also techniques to improve these methods. In order to do so, they learn from past experiences and transfer them to previously unseen datasets. One of them is meta-learning, which is discussed in Section 2.4.

2.2 Hyperparameter Optimization Techniques

This section describes hyperparameter optimization techniques that are used in existing AutoML systems. The primary task of these systems is to optimize a black-box function f, which input is a configuration from the configuration space. The common steps of the opimizers are:

(1) Select a configuration to execute.

(2) Execute the current configuration on a dataset.

(3) Evaluate the results of the configuration with a metric.

There, the steps (2) and (3) are typically included in f. In general, the optimizers differ in how they perform step (1), i.e., how they select a configuration. Also, they usually have a fixed number of iterations how often they can perform the three above mentioned steps.

The optimizer methods can be divided into three categories [BB12; BBBK11; FKE+15; FKH18; LJD+17]:

- **Base**: Methods that are basically used like the Random optimizer.
- **Sequential model-based**: Methods that create a model to predict the best hyperparameters. One representative of this class is the Bayesian Optimization (BO) method.
- **Multi-armed Bandit**: These methods select a lot of configurations in parallel with a very low budget for each configuration. Examples for this class are Hyperband and Bayesian Optimization and Hyperband (BOHB), which is a combination of BO and Hyperband.

These methods are analyzed in the following in more detail.

2.2.1 Random

One of the simplest hyperparameter optimization methods is Random [BB12]. Nevertheless, it can be very effective, especially if the configuration space is not that large. In general, it performs even better than Gridsearch [BB12], which defines a subset of the configuration space (typically a grid) and evaluates all points in this subset. As the name suggests, the selection of the next configurations is done randomly. This is also a major drawback of Random because it does not take the knowledge of already evaluated configurations into account to select the next configurations.

Algorithm 2.1 General Bayes Optimization Procedure.

```
1: procedure BayesOptimization(i, f, C )
2:     M ← Initialize probabilistic model
3:     D ← Initialize configurations
4:     for j ← 1,...,i do
5:         c_j ← Select next configuration
6:         y_j ← f(c_j)
7:         D ← D ∪ (c_j, y_j)
8:         M ← Update statistical model
9:     end for
10: end procedure
```

2.2.2 Bayes Optimization

The Bayes optimization is a sequential model-based optimization (SMBO) method relying on a probabilistic model and an acquisition function [BCD10; Fra18; SLA12; SSW+16]. The goal is to approximate a black-box function $f(x)$ using a probabilistic model, which is also called the model posterior. The function $f(x)$ is generally expensive to evaluate and can be approximated in a cheaper way.

The general procedure of Bayesian optimization is shown in Algorithm 2.1. Line 1 describes the inputs. The number of iterations that the optimization procedure can perform is denoted by i. This corresponds to the budget of an AutoML system and could also be a time-constraint instead. The other inputs are the black-box function f and the configuration space C. First, the probabilistic model is initialized, which means a statistical model with its configuration is chosen (line 2). The set D which holds the configurations with their evaluations has to be initialized as well (line 3). This is done either with several random configurations or empty. Then, in each iteration, a configuration c_i is selected for evaluation (line 5). For this, an acquisition function is used to select the next configuration. Hereby the acquisition function must take a trade-off between exploration and exploitation. On the one side configurations that seem to perform good according to M and on the other side configurations where great uncertainty about the performance prevails should be selected. An often-used acquisition function is the Expected Improvement (EI) [BCD10]. Line 6 is the most expensive step of the procedure. There, the black-box function is evaluated for the configuration c_i. The result y_i together with the configuration c_i is added to the set D (line 7) and based on D, the probabilistic model is updated (line 8). The *best* configuration is the one that achieves the best value with respect to f.

An often-used probabilistic model is the Gaussian Processes [Ras04] model due to the facts that the objective is continuously and every iteration in the optimization procedure only depends on the last iteration [BCD10]. Also, tree-based models have shown to perform well [FKE+15; HHL11; THHL13] because of their simplicity and lower runtime complexities.

An advantage of Bayesian optimization over Random is that it uses the knowledge of the already evaluated configurations in order to select the next configuration. Though it produces an overhead by updating the probabilistic model, this overhead balances out by a good choice of configurations to evaluate next. But this only holds if the configuration space is large and the evaluation of f is

Algorithm 2.2 General procedure of Hyperband.

1: **procedure** HYPERBAND(B, η)
2: $s_{max} \leftarrow \lfloor log_\eta(B) \rfloor$
3: $B_{max} \leftarrow (s_{max} + 1) * R$
4: $L \leftarrow \emptyset$
5: **for** $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
6: $n \leftarrow \lceil \frac{B_{max} * \eta^s}{B*(s+1)} \rceil$
7: $r \leftarrow \frac{B}{\eta^s}$
8: $L \leftarrow L \cup$ SUCCESSIVEHALVING(n, r, s)
9: **end for**
10: **return** Configuration with smallest intermediate loss
11: **end procedure**
12:
13: **procedure** SUCCESSIVEHALVING(n, r, s)
14: $T \leftarrow$ Select n hyperparameter configurations uniformly random
15: $AllEvaluations \leftarrow \emptyset$
16: **for** $i \in \{0, \ldots, s\}$ **do**
17: $n_i \leftarrow \lfloor \frac{n}{\eta^i} \rfloor$
18: $r_i \leftarrow \frac{r}{\eta^i}$
19: $L \leftarrow \emptyset$
20: **for** $t \in T$ **do**
21: $y_i \leftarrow f(t, r_i)$
22: $L \leftarrow L \cup (y_i, t, r_i)$
23: $AllEvaluations \leftarrow AllEvaluations \cup (y_i, t, r_i)$
24: **end for**
25: $T \leftarrow$ Get top $\lceil \frac{n_i}{\eta} \rfloor$ performing configurations from T
26: **end for**
27: **return** $AllEvaluations$
28: **end procedure**

expensive or there is no closed form solution. The latter one means that it is not possible to derive the gradients of f. Moreover, Bayes optimization is a sequential procedure, which means it is hard to parallelize it.

2.2.3 Hyperband

Hyperband [LJD+17] is a multi-armed bandit method that relies on Successive Halving (SH) [JT16]. In contrast to Bayesian optimization, the basic idea is to select as many configurations as possible in parallel but with a smaller budget for each parallel execution. Subsequently, the best configurations that were evaluated are evaluated with a higher budget. This is repeated until only one configuration is left which is executed with the maximum budget.

The general algorithm is shown in Algorithm 2.2. The inputs in line 1 are the maximum budget B for a single configuration and η which is a ratio of how many configurations are discarded from one SH round to another. The algorithm can generally be divided into three phases. The first part ranges

from line 2 - 4. Here, the maximum number of SH runs that are performed (line 2), the maximum budget B_{max} that can be allocated through an entire Hyperband run (line 3) and L which saves the best results of each SH run (line 4) are initialized. In the second part (line 5 - 9), the number of configurations n that are selected at the beginning of each SH run and the resource budget that can be allocated for one iteration of SH are determined. Finally, in the third part (lines 13 - 29) the SH runs are performed. In line 14, n configurations are selected uniformly random from the configuration space. In the lines 17 - 23 an SH iteration is performed. There, n_i (line 17) describes the number of configurations that are evaluated in one SH iteration and r_i the budget that is used to evaluate this configuration. In lines 21 - 23 each selected configuration is evaluated using the function f and the budget r_i. Here it is important to notice that it must be possible to define a budget for f which approximates $f(t)$. This can be, e.g., the number of iterations for an algorithm or a time constraint. In line 25 the top $k = \lfloor \frac{n_i}{\eta} \rfloor$ are chosen and taken to the next SH iteration. Line 27 returns the set $AllEvaluations$, because for finding the best configuration not only the SH iterations with maximum budget are considered but all evaluations of a certain configuration. This is done in line 10 by finding the configuration with the smallest *intermediate* loss of all evaluations.

An advantage of Hyperband is that it is easy to parallelize, especially running the different SH runs and its iterations. It also does not evaluate configurations with maximum budget but rather starts with a small budget and only evaluates the best η configurations with a higher budget. This also means that it has to be possible to approximate $f(\cdot)$ for a given budget b with $\hat{f}(\cdot, b)$ while $\hat{f}(\cdot, b) = f(\cdot)$ if b is the maximum budget. Additionally, it does not use the knowledge of already evaluated configurations to select the next configurations. Another drawback is that Hyperband takes all evaluations of configurations into account. This can lead to biased solutions. The performance of configurations that perform good with maximum budget could be worsened by runs with smaller budget although the runs with maximum budget are more important.

2.2.4 Bayesian Optimization and Hyperband

BOHB [FKH18] is a combination of the Bayesian optimization and the Hyperband method. In Hyperband, the configurations that are evaluated in the SH iterations are chosen uniformly random. In comparison to that BOHB, like Bayes optimization, uses a statistical model to select the next configurations for an SH run. Instead of random selecting configurations in the different SH iterations of Hyperband, BOHB uses Bayesian optimization to fit a model and to select the next configurations. This means that in Algorithm 2.2 only line 14 would change between Hyperband and BOHB. Though BOHB keeps track of all evaluations of configurations, only the ones with maximum budget are used to determine the best configuration. In contrast to that, Hyperband determines the best configuration according to the intermediate loss of all evaluations for a configuration. Also, BOHB shares all configurations with their evaluations and with the used budget across all SH runs.

2.3 Clustering

Clustering is an unsupervised learning method that is often used in ML [Bis06] or Data Mining [HKP12] tasks. In contrast to classification or regression, the input data does not contain any expected values for each entity, such as class labels. Clustering is used in many different applications such

as business intelligence, image pattern recognition, Web search, biology and security [HKP12]. Clustering aims at finding groups (or clusters) in a dataset such that the data points in the partitions are very similar (maximizing the *intra-cluster* similarity) and the data points from different clusters are dissimilar (minimizing *inter-cluster* similarity) [KC88]. Therefore, a similarity measure is needed that describes the similarity between the data points. An often-used metric is the euclidean distance metric [KC88]. If the data is partitioned into k disjunct partitions the clustering is called *hard* clustering and otherwise *soft* clustering. Although, several classes of clustering algorithms exist [FAT+14; XT15; XW05], the focus of this work will be on *partitional* clustering algorithms.

2.3.1 Partitional Clustering Algorithms

Partitional clustering algorithms are often used due to their simplicity and effectiveness at the same time [Jai10]. They initially choose k clusters (often randomly) and then iteratively improve these clusters. To this end, the clusters are represented by a centroid. In each iteration the centroids are updated, and the data points are assigned to the cluster which centroid has the nearest distance. This is repeated until either a predefined number of iterations is reached, or no more changes are performed. For this, the number of clusters must be chosen by the user. However, choosing k is not a trivial choice and many methods exist to estimate k [DB79; HHE03; PM00; Rou87; SJ03]. Moreover, partitional clustering algorithms can find local optima instead of global ones due to the random initialization of the centroids. It was shown that the initialization has a great impact on the performance and the result [FS19]. In addition, partitional clustering algorithms tend to detect only clusters with non-convex shapes.

K-means

The k-Means [Mac67] algorithm is a very popular partitional clustering algorithm [WKR+08] with many variations [AV07; BMV+12; KMN+02; KRTT15; Llo82; SKK00]. The centroids of k-Means are the centers of each clusters, which are synthetically generated as geometric mean of each cluster. The general procedure is as follows:

1. Initialize k centers randomly.
2. Assign each data point x to the cluster with the center which has the smallest distance to x.
3. Move the centroid to the center of its cluster.
4. Repeat the steps 2 and 3 until no point changes its cluster affiliation anymore, or a threshold of the number of iterations is met.

A major advantage of k-Means is the runtime complexity of $O(n * m * k * i)$ where n is the number of data points, m the dimensions of each data point, k the number of clusters and i the number of iterations that were performed. This makes it feasible to handle large datasets because it scales linearly with the number of data points. Though, it has poor performance for handling datasets with high dimensionality [HKP12]. In addition, the algorithm is sensitive to noise since they affect the position of the centroids [GKR91].

K-medoids

The k-Medoids [KR87] algorithm is similar to the k-Means algorithm. The difference between both algorithms is the calculation of the cluster centers. While k-Means calculates the cluster centers by the geometric means of each cluster, k-Medoids uses mean data points in a cluster, called *medoids*. The algorithm runs the following steps:

1. Initialize k medoids.
2. Assign each data point x to the medoid with the smallest distance.
3. For each cluster choose a data point as new medoid such that the distances between all points in the cluster and the medoid is minimal.
4. Repeat steps 2 and 3 until the medoids do not change anymore, or a predefined number of iterations is reached.

The k-Medoids algorithm is more robust to outliers since the center of a cluster is represented by a real data point of the dataset. Additionally, because the medoid is a data point, any distance metric can be used. For k-Means this does not necessarily hold since finding the (synthetically) mean in a metric space is not an easy task for every distance metric. However, the algorithms for k-Means and k-Medoids seem to be similar, the runtime complexity for k-Medoids is quadratic, in fact $O(n^2 * k * i)$. That is, because the third step of the k-Medoids procedure, choosing a new medoid, is a very cost intensive step. For every data point of a cluster, it must be checked if it is worth to swap the current medoid with this point. Checking if a swap is worth means that the distance of each data point in the cluster to the medoid must be calculated, resulting in pairwise comparisons, i.e., the quadratic runtime of the algorithm. Similar to k-Means, k-Medoids does not guarantee to find a global optimum.

Gaussian Mixture Models

The Gaussian Mixture Models (GMM) [Ras04] is an implementation of the Expectation Maximization (EM) [DLR77] algorithm. Here, each of the k clusters is assumed to follow a Gaussian distribution $\mathcal{N}(x|\mu,\sigma)$. There, μ represents the center of each Gaussian distribution, Σ is the covariance matrix and $x = \{x_1, \ldots, x_n\}$ a set of observations. Additionally, a probability distribution π that describes for each cluster the probability that a point inside this cluster is used. The algorithm performs the following steps [Bis06]:

1. Initialize the parameters for each gaussian mixture $\Theta = (\Theta_1, \ldots, \Theta_k)$ with $\Theta_i = (\pi_i, \mu_i, \Sigma_i)$.
2. Evaluate the current cluster affiliations for these parameters:

$$\gamma(z_{ni}) = \frac{\pi_i \mathcal{N}(x_n|\mu_i, \Sigma_i)}{\sum_{j=1}^{k} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)} \tag{2.1}$$

3. Update the current parameters of each cluster i:

$$\mu_i = \frac{1}{N_i} \sum_{n=1}^{N} \gamma(z_{ni}) x_n \tag{2.2}$$

$$\Sigma_i = \frac{1}{N_i} \sum_{n=1}^{N} \gamma(z_{ni})(x_n - \mu_i)(x_n - \mu_i)^T \tag{2.3}$$

$$\pi_i = \frac{N_i}{N} \tag{2.4}$$

Where $N_i = \sum_{n=1}^{N} \gamma(z_{ni})$.

4. Repeat steps 2 and 3 until the parameters or the log-likelihood do not change anymore or a predefined number of iterations is met.

In contrast to k-Means and k-Medoids, the result of GMM is a probability distribution instead of a hard-clustering result. That means that every data point has a probability, which indicates it belongs to a certain cluster. The probability that a point x_n belongs to a cluster i is described by $\gamma(z_{ni})$ (see Equation (2.1)). Like for k-Means and k-Medoids, GMM does not find a global optimum. This is also dependent on the initialization of the hyperparameters. GMM has the same runtime complexity like k-Means, which is $O(n * m * k * i)$ [BBB19].

2.3.2 Clustering Metrics

The results of a clustering can be evaluated with a suitable metric. For the evaluation of clustering results, three classes of metrics exist [HBV01; KC88]:

Internal Metrics These metrics evaluate the internal structure of clusters. To this end, they do not require external information. They measure the compactness and the separation of the resulting clusters. Most often the internal measures are combinations of both.

External Metrics In contrast to internal metrics, external metrics require external information. They use a gold-standard to compare the result of a clustering algorithm to the expected clustering result.

Relative Metrics This class of metrics uses other metrics, like the internal ones, to compare different clustering results. Typically, they run an algorithm with different configurations and then select the best configuration according to the metric. In this regard they are similar to the optimization techniques in Section 2.2. However, in contrast to the optimization techniques, the relative metrics do not define a strategy how to select the next configuration but rather do an exhaustive search over a predefined range.

The relative metrics are not considered explicitly for this work as the optimizer methods in Section 2.2 are a kind of relative metrics but with different strategies to search for the best configuration. The internal metrics are used in this work since they can be directly applied on clustering results without external knowledge. Though the external metrics cannot be applied without an external knowledge they could be used in a training or offline phase, assuming that the labels for the datasets are known in this part.

The external metrics compare the clustering labels that are predicted by an algorithm against the expected ground truth labels. Typically, their score is either in $[-1; 1]$ or $[0; 1]$, while a score of 1.0 indicates a perfect match of the predicted and the ground truth labels. An important aspect for external metrics is if they can ignore permutations. This means that if the names of class A and B are swapped the metric should calculate the same value for both labelings. For example two

cluster assignments $A = (1, 1, 2, 2, 2)$ and $B = (2, 2, 1, 1, 1)$ should compute the same score. This property holds for all external metrics. In the following descriptions the labels that are predicted by a clustering algorithm are denoted by P and the expected ground truth labels by E.

The Adjusted Rand Index [HA85] measures the probability that every pair of points are clustered similarly. This means that either both points are in the same class in P and E or both are different classes. The values of this metric are in $[-1; 1]$. The V-measure [RH07] is based on conditional entropy analysis and consists of two other measures, which are Homogeneity and Completeness. The objective of Homogeneity is to measure that each cluster contains only members of a single class. The Completeness score on the other side measures if all members of a given class are in the same cluster. The relation between Homogeneity and Completeness can be described as $Homogeneity(P, E) = Completeness(E, P)$. The V-measure describes the geometric mean of the Homogeneity and the Completeness score. The values of all three are bounded by 0.0 and 1.0. The Fowlkes-Mallows [FM83] score measures the geometric mean of precision and recall. It is also bounded by 0.0 and 1.0. The Adjusted Mutual Information [VEB10] is based on entropy and measure the information that P and E are sharing. The values of this metric are in $[0; 1]$.

In contrast to external metrics, the internal metrics do not need external information. They are based on compactness and separation measures. The values of internal metrics are typically not normalized. However, they all have an objective which is either the minimum or maximum. This means, when evaluating different clustering results, the best result can be obtained by either the lowest or the highest score.

The Silhouette Coefficient [Rou87] is defined for each data point. Let a denote the average distance from a point to all other points in the same cluster and b the mean distance between each centroid of the clusters. The Silhouette coefficient for a point is defined by the subtraction of a from b and dividing through the maximum value of a and b. Subsequently, the average of all silhouette values for each point is obtained. The Silhouette coefficient is the only examined internal metric in this work that is normalized. The values lie in the interval $[-1; 1]$. The higher the score the better is the clustering result. The Davies-Bouldin Index [DB79] is calculated by a quotient of a compactness and a separation measure. The compactness is measured by the average distance of each point of a cluster to its centroid. The separation is measured by the distance between the centroids of the clusters. It is not normalized but the best and lowest score is 0. The Calinski-Harabasz Index [CH74] is the ratio of the between cluster dispersion (a separation measure) and the within-cluster dispersion (a compactness measure). Values of this metric are not bound, but higher values indicate better results.

2.4 Meta-learning

Meta-learning [BGSV08] describes the task of learning based on already acquired experiences from previous tasks. In the area of machine learning, it can be used to learn configurations and their performance from previous tasks. To this end, the experience is based on characterizations of already processed datasets and the evaluations of configurations on these. The dataset characterizations, also called *meta-features*, are used to identify which of the previously processed datasets are most similar to new unseen datasets. The best configurations with respect to the evaluations of the already processed datasets are recommended for the new unseen dataset.

For the characterization of the datasets, various meta-features exist. In general, they can be divided into the following categories [RGS+18]:

- **Simple**: Can be directly extracted from the data and include measures like the number of instances or the number of attributes.
- **Statistical**: Measure statistical information about the data. They include measures like mean, median, standard deviation, etc.
- **Information-theoretic**: Based on entropy and capture the information and the complexity in the data. For example, the attribute concentration measures how strong the association between each attribute and each class label is.
- **Model-based**: This group of meta-features use structural properties of machine learning models. For example, they build a decision tree from the data and the number of leaves of the tree is one meta-feature.
- **Landmarking**: The landmarking meta-features use the performance of simple classifiers as meta-features. An example is the accuracy of a Naive Bayes classifier. That is, a dataset is similar to another dataset if the accuracy produced by the classifier is similar.

Landmarking and model-based meta-features cannot be used for clustering since we are assuming that no class labels are available. Therefore, only meta-features that do not require class labels are used in this work.

3 Related Work

In this section the related work, which is structured in three parts, is presented. Section 3.1 presents existing AutoML frameworks. An overview of existing approaches to use meta-learning for clustering is given in Section 3.2. Section 3.3 describes hyperparameter estimation methods that are applied for clustering. A summary of the related work is given in Section 3.4.

3.1 AutoML Frameworks for classification

In this section commonly used AutoML systems are described. For this, fundamental functionalities of each system are explained.

One of the first proposed AutoML systems is Auto-WEKA [KTH+17; THHL13]. It is based on the machine learning software WEKA [HFH+09] that supports 39 classification algorithms, whereas two of them are ensemble methods. To this end, each algorithm can have its own set of required parameters. In the case of WEKA, this results in overall 786 hyperparameters. The authors of Auto-WEKA formulate the problem of selecting an appropriate algorithm with suitable hyperparameters as CASH problem. They extend the hyperparameter search space by a root-level hyperparameter, which is the name of the algorithm, and use tree-based Bayesian optimization methods to solve the problem. The root-level hyperparameter is a conditional hyperparameter; depending on its value the other hyperparameters are chosen. This results in a smaller search space and hyperparameters that are not needed for certain algorithms are not taken into account. Regarding the Bayes optimization, two implementations were compared. These were Sequential model-based algorithm configuration (SMAC) [HHL11] and Tree-structured Parzen Estimator (TPE) [BBBK11]. They compared them with each other and with Random and Gridsearch. Thereby, SMAC outperformed the others.

The Auto-sklearn [FKE+15] system extends the concept of Auto-WEKA and has won the first AutoML challenge [GBC+15]. In contrast to Auto-WEKA, Auto-sklearn uses scikit-learn [PVG+11] as machine learning library, which implements 15 classification algorithms. Similar to Auto-WEKA, Auto-sklearn solves the CASH problem using a new conditional root-level hyperparameter for the algorithm name. An overview of the Auto-sklearn workflow is given in Figure 3.1. The concept of Auto-WEKA is shown by the ML framework and the Bayes optimizer, which is also implemented

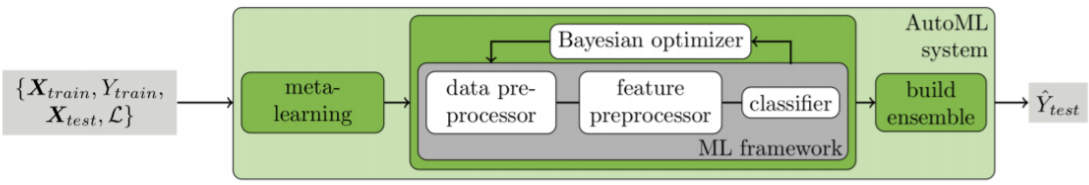


Figure 3.1: Basic workflow of the Auto-sklearn system [FKE+15].

using the SMAC library. However, Auto-sklearn adds two components to the concept of Auto-WEKA, which are *meta-learning* and *build ensemble*. The meta-learning component is used to warmstart the Bayesian optimizer with configurations that performed well on similar datasets in the past. Instead of including ensemble classifiers in the optimization procedure, Auto-sklearn takes the models that performed best and builds an ensemble out of them. This reduces the hyperparameter space to 110 hyperparameters compared to 760 hyperparameters in Auto-WEKA. The authors showed that these two components lead to better performance and accuracy.

The PoSH Auto-sklearn won the second AutoML challenge [GSB+19]. The system is based on Auto-sklearn but they combined the Bayesian optimizer with Hyperband [LJD+17]. This has the benefit that more configurations are evaluated with a smaller budget and bad performing configurations can be discarded earlier. This optimizer is combination of Bayes optimization and Hyperband, also known as BOHB [FKH18]. The other components are the same as for Auto-sklearn, i.e., meta-learning and ensemble building are also used. They showed that the BOHB optimizer outperformed the Bayesian optimizer and lead to better results.

Note, that more AutoML systems exist, examples can be found in the surveys [EMS19; QMH+18; ZH19] but all of them use the concepts described above (or at least some of them). However, they all cover supervised learning methods and none of them applied these concepts on unsupervised learning algorithms. This means there is no approach that covers the algorithm selection and the hyperparameter optimization together for clustering. To this end, the following two sections are discussing related work that covers on the one side the algorithm selection problem and on the other hand, the estimation of hyperparameters for clustering on their own.

3.2 Algorithm Selection for Clustering using Meta-Learning

In the literature, several approaches that attempt to select a suitable clustering algorithm exist [DPS+08; FC12; GN15; NPDC09; PC19; SC19; SLD09]. All these approaches have in common that they are using meta-learning to address this problem. They differ in the used (a) meta-features, (b) clustering algorithms, (c) metrics for evaluating the clustering results, (d) meta-learning model, (e) ranking algorithm to recommend the best configurations.

However, all the approaches have at least the following three components: 1. A *meta-feature extractor*, which extracts the meta-features from a given dataset. 2. A *meta-learner*, which applies a machine learning algorithm on the extracted meta-features of the training data, which are already processed data, and applies it to unseen data. 3. A *database* that stores the training data with the extracted meta-features and also a ranking for the best configurations. Nevertheless, none of the works use all meta-features that are used by existing AutoML systems. They also do not consider the CASH problem, but only the algorithm selection problem. To this end, they use various algorithms with the "optimal" hyperparameters for each dataset. This means they only consider datasets where, e.g., the number of clusters is known in advance. This also leads to a very limited configuration space since the hyperparameters are fixed and hence only the different algorithms are included.

De Souto et al. use 8 meta-features, which are mainly based on statistical measures [DPS+08]. Two of them are only applicable to the micro-array domain. They consider 7 clustering algorithms and run them on 32 datasets, on each with the "optimal" hyperparameters. Thus, the configuration space only contains 7 algorithms. The authors use the regression Support Vector Machines (rSVM)

algorithm as meta-model. To this end, they used the meta-model to predict a ranking for the algorithms. They use the adjusted Rand index, an external metric, to evaluate the result of the algorithms. The optimal algorithm for a dataset is determined by the best metric score. However, they only use datasets from a very specific domain, the cancer gene expression micro-array domain. They also use a limited set of meta-features and the datasets require a the ground truth clustering labels to evaluate the best result.

Nascimento et al. [NPDC09] use meta-learning to derive knowledge from the training datasets. The knowledge is derived in form of rules, which automatically decide which clustering algorithm should be used. They built on the approach from [DPS+08] by using the same algorithms and extending the meta-feature set with 5 additional meta-features. For the meta-learning model, they evaluate six different algorithms, which are: J48, PART, MLRules, Random Forest, k-NN and Support Vector Machine. Their evaluation showed that Random Forest and MLRules had the best performance in average. Though, their evaluation is not generally meaningful, since only datasets from the domain of gene expression for cancer are used.

Soares et al. compare two sets of meta-features, which both contain 9 meta-features based on statistics [SLD09]. They also evaluate two meta-models, the Multi-layer Perceptron (MLP) and the Support Vector Regressor (SVR), for predicting the best algorithm on the meta-features of an unseen dataset. They take 9 clustering algorithms into account and use the global error rate as metric to obtain the optimal result. Though they use more algorithms, the configuration space here still contains only 9 configurations. Additionally, the ground truth clustering labels are required to evaluate the global error rate. However, they use a very large set of datasets, namely 160 artificially generated datasets, which are either based on a gaussian or an ellipsoid cluster generator. Though, the maximum number of instances that are generated by the cluster generator is limited to 800 for both cluster generators. They show in their evaluation that both meta-models in combination with both meta-feature sets are at least better than the default ranking (average rank from training phase).

Gomes Ferrari and Nunes de Castro provide a novelty by evaluating three meta-feature sets [GN15]. These are the distance-based, attribute-based and the hybrid-based, which is the union of the other two, meta-feature sets. The distance-based meta-features are all meta-feature that can be calculated from the distance among each pair of instances in one dataset, e.g., the mean or the variance of the data. The attribute-based meta-features are based on the attributes of a dataset, e.g., the number of instances, attributes, etc. Overall, 19 distance-based and 9 attribute-based meta-features are taken into account, i.e., the hybrid-based set of meta-features contains 26 meta-features. They take 7 clustering algorithms into account. Since they also only take the "optimal" hyperparameters, this results in a configuration space with 7 elements. They use the k-NN algorithm as meta-model while evaluating different values for k. Also, a novelty in their approach is that they also use internal metrics to evaluate the results of the clustering algorithms. They create one ranking for each internal metric and for each algorithm. At the end to get a ranking per algorithm they combine all rankings from the different internal measures. They use 84 datasets from the UCI repository and evaluate all three sets of meta-features. The results showed that the best results were obtained for using the distance-based meta-features. Overall, the novelty in this paper is to use the distance-based meta-features, using multiple internal metrics and combine the different rankings from each metric. However, in the evaluation the hyperparameters for each clustering algorithm are fixed. They do not consider how to choose the best hyperparameters. Also, for an unseen dataset, calculating these 10 internal metrics and combining them adds additional computation overhead.

Pimentel and de Carvalho evaluate a new set of meta-features that combines correlation and dissimilarity, where the correlation measure is the main novelty of their approach [PC19]. They consider 10 clustering algorithms, i.e., from all presented approaches they use the greatest number of algorithms, though the configuration space contains only the 10 algorithms. The quality of clustering results is evaluated with 10 internal metrics. They also evaluate two meta-learners, namely the k-NN and random forests. In the evaluation they compare their set of meta-features to other existing set of meta-features. They perform their evaluation on 219 datasets which are taken from OpenML and from different domains, such as engineering, biology or physics. Moreover, they compare their algorithm selection system with existing ones. The results showed that their approach performed better than the ones that are already existing in the literature when using k-NN as meta-learner. Nevertheless, they also do not include the hyperparameter selection but only address the algorithm selection problem.

3.3 Methods for estimating the Number of Clusters

In the literature different methods exist to estimate the hyperparameters of a clustering algorithm. This work focuses on partitional clustering algorithms. For that reason, only the methods for estimating the number of clusters (denoted by k) are examined in the following.

In general, the methods for estimating the number of clusters can be divided into semi-automated methods and automated methods. Both require the execution of the clustering algorithm for parameters within a specified search space. Subsequently, the results for each value of k are evaluated according to a defined metric. This metric is the main difference between all methods. To this end, the Davies-Bouldin Index [DB79], the Silhouette index [Rou87], Calinski-Harabasz and many more [HBV01] can be used. The automated methods just select the k as best k that has the best metric value. Semi-automated methods require additional human assistance. For example, the elbow method [Tho53] uses the sum of squared errors (SSE) which is a monotonically decreasing function with an increasing number of clusters. The method displays a graph, containing all SSE values on the y-axis and k values on the x-axis, to the user and the analyst has to find an "elbow". The jump method [SJ03] automates this method by calculating the differences between every pair of k and $k-1$ values in the specified range. Then a transformation, in particular a negative exponent is added to each value, is applied such that the best value is chosen as optimal k value.

There are also methods that do not execute all k values in the specified range but only a certain number until some criterion is met. The G-Means [HHE03] and the X-Means [PM00] methods are examples for this. They start with a minimum number of clusters, e.g., 2 and then split each cluster by running k-Means with $k = 2$ only with the data from that cluster. For each split it is checked if some criterion is full-filled and rejected if not. This procedure is repeated until either no more splits are performed or a predefined upper bound is reached. The procedure has two advantages: First, k-Means does not have to be performed on the whole dataset and not for every k during the splitting procedure. The second one is that the evaluation if the criterion is met is also not evaluated on the whole dataset but only on the split clusters.

Sáez and Corchado estimate the number of clusters using meta-learning [SC19]. He uses 20 well-known internal clustering metrics as meta-features. The approach is divided into three stages. The first is the extraction of meta-features. Therefore, k-Means is executed with 9 different k values, ranging from $k = 2$ to $k = 10$, resulting in 180 meta-features that are extracted for each dataset. In

the second stage, they determine if the data is uniform or not and train a C4.5 model. If the data contains at least two clusters, the last stage, the application stage, is performed. To this end, all 180 metrics are computed for the new dataset and are given as input to a C4.5 model, which then determines the cluster cardinality. The evaluation is done using 30 real-world datasets from the UCI repository, but for each of them 25 new datasets are generated using stratified sampling. They also use 2250 synthetic datasets, which are generated according to a cluster generator method [QHJQ06]. Although the authors can predict the number of clusters with unsupervised data characterization using meta-learning, the effort for predicting the meta-feature for an unseen dataset is unacceptable high. Because, 180 metric calculations have to be performed.

3.4 Summary of the Related Work

In this section a summary of the presented related work is given. The related work shows approaches that are using AutoML to select appropriate supervised learning algorithms as well as their hyperparameters. To this end, approaches were shown that address the same problem for clustering. However, the existing literature lacks the following:

- The present AutoML systems take only supervised learning algorithms and their hyperparameters into account. None of them applies them to unsupervised learning like clustering.
- The methods for selecting clustering algorithms all rely on meta-learning, but they neglect the estimation of the hyperparemters.
- Each work that use meta-learning for clustering uses a different set of meta-features. Thus, it is not clear which meta-features lead to the most promising results. There is also no work that uses the same set of meta-features that are used in the current AutoML systems.
- The methods for estimating the hyperparameters for clustering do not use the methods that are used in AutoML systems, e.g. Bayes optimization. They typically execute the clustering algorithm and subsequently evaluate the result for all values in a predefined range. The optimization methods can remedy this by only running and evaluating k values in the range until a predefined budget is exceeded. Hence, runtime savings should be able to achieve. However, the accuracy of these methods is unclear.

4 Analysis of existing AutoML Systems

In this chapter, a detailed analysis of the concepts of the existing AutoML systems is conducted. The goal of these systems is to find a suitable configuration for a given training dataset within a given budget. In this context, a configuration is an algorithm with corresponding hyperparameter values. The space that contains a set of possible configurations is called the *Configuration Space*. To find a suitable configuration, the systems select different configurations while the budget is not exhausted and then select the best performing one according to an optimization metric. This metric is usually an external measure that compares the resulting labels of executing a configuration to the expected labels. The budget specifies the resources the AutoML system can use. This can be any kind of resource, e.g., a time constraint is a common choice.

The general architecture is shown in Figure 4.1. This architecture provides an abstract overview of existing AutoML systems, such as Auto-WEKA [THHL13] and Auto-sklearn [FKE+15]. The systems perform in general three steps: (1) The *Initialization* initializes configurations for the optimizers and is described in Section 4.1. (2) The *Optimizer Loop* runs the optimizer iterations until the budget is exhausted. Each step of the iteration is explained in Section 4.2. (3) The *Return Best Configuration* step chooses the best configuration from all executed configurations. The corresponding procedure is discussed in Section 4.3.

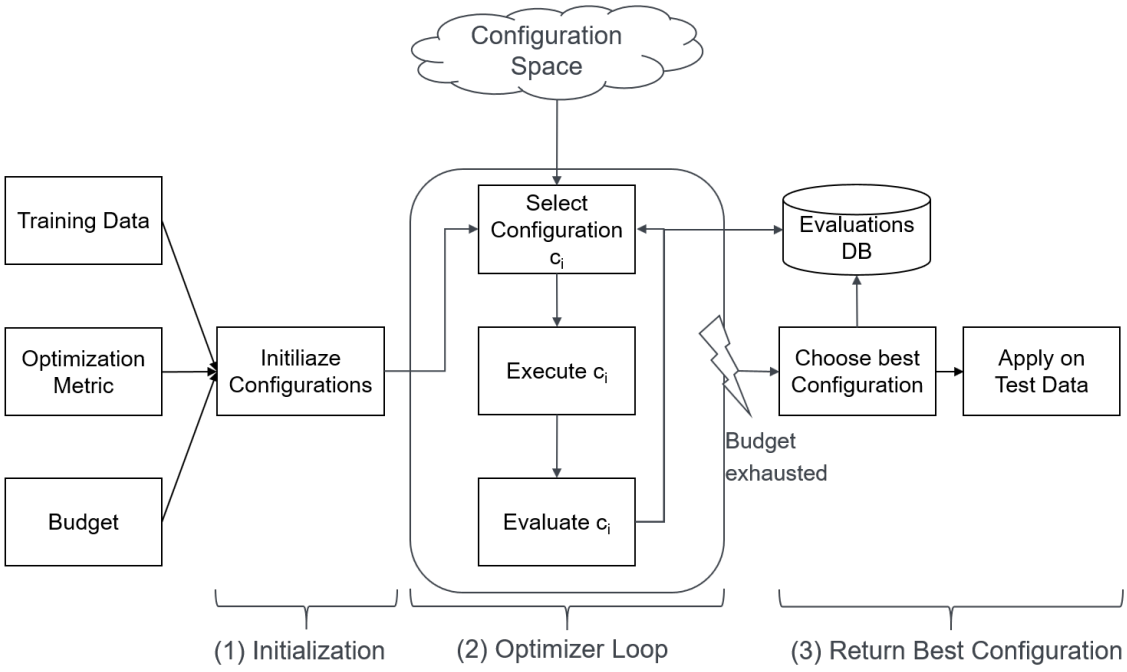


Figure 4.1: General architecture of AutoML systems.

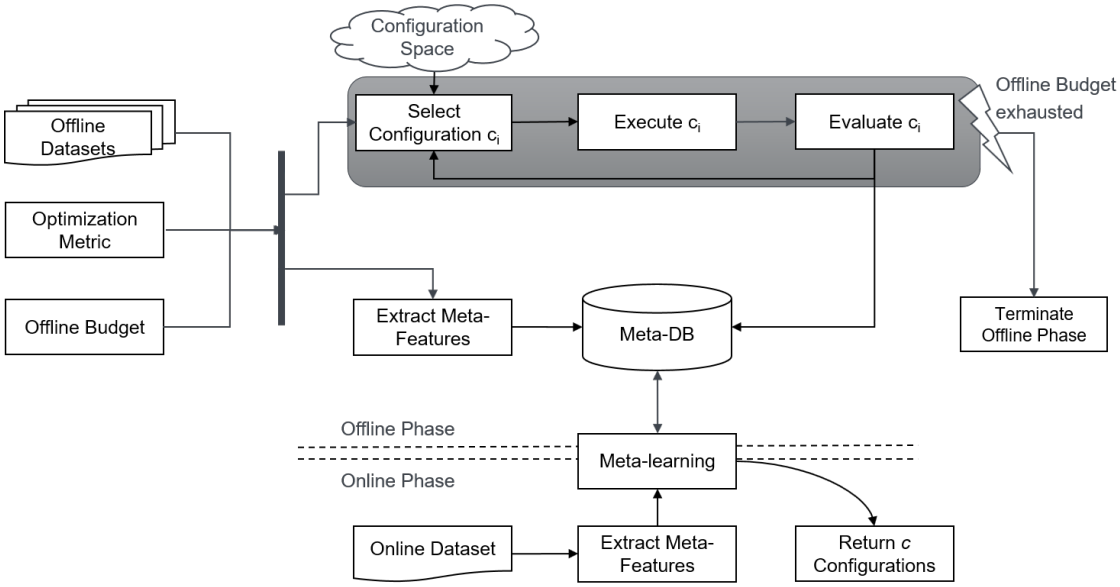


Figure 4.2: General architecture of meta-learning, divided into offline and online phase.

4.1 Initialize Configurations

The step *Initialize Configurations* in Figure 4.1 initializes a set of configurations. These are the configurations that the optimizer considers first. The initialization is typically performed at random [THHL13], but also more advanced methods like meta-learning can be applied [FKE+15]. The latter transfers knowledge from past evaluations of datasets to unseen datasets. Feurer et al. showed that this improves the performance of the optimizer and leads to better results. The random initialization is called *coldstart*. The initialization with meta-learning is referred to as *warmstart* [FSH15].

4.1.1 Meta-learning

Meta-learning predefines several configurations to be selected in the first *Optimizer Loop*. The knowledge of which configurations are suitable for this is transferred from past experiences. For this purpose, already evaluated configurations on datasets are required. Therefore, meta-learning is divided into two phases, an offline and an online phase, which is shown in Figure 4.2.

The offline phase is executed only once. In contrast to that, the online phase is executed each time a dataset must be classified. Furthermore, a higher budget in the offline phase can be used, since it is only executed once. However, the online phase can profit from more evaluations in the offline phase by providing better warmstart configurations.

The inputs for the offline phase are like the ones for AutoML systems, namely *Offline Datasets*, an *Optimization Metric* and an *Offline Budget*. The *Offline Datasets* are multiple datasets that are all in the format to apply supervised learning on them. The *Optimization Metric* is used for evaluating the configurations and is typically an external metric like the accuracy for classification. The *Offline Budget* is the budget that can be used for the offline phase to select, execute and evaluate configurations. The *Offline Phase* performs two procedures for each offline dataset:

1. The first procedure includes the following three consecutive steps: (a) *Select Configuration* c_i selects a configuration c_i from the *Configuration Space*, which contains all investigated configurations, (b) *Execute* c_i executes the selected c_i on one dataset of the *Offline Datasets* and (c) *Evaluate* c_i in order to evaluate the result of the executed c_i. To this end, the *Optimization Metric* is used to evaluate the result. The score of the metric is stored in the *Meta-DB*, which is a database that stores the configurations as well as their evaluations. The steps (a), (b) and (c) are executed several times on datasets with different characteristics. In practice, an optimizer performs these steps until the *Offline Budget* is exhausted [FEF+18; FKE+15]. For this, the implementation of the black-box function f includes the steps (b) and (c).
2. *Extract Meta-Features* extracts the meta-features from the offline dataset and saves the result in the *Meta-DB*. These meta-features are numerical values, e.g., the number of data points in a dataset or the number of attributes. Because of this, common distance metrics like the Manhattan distance can be applied directly on two meta-feature sets. This is useful for measuring the similarity between two datasets by measuring the distance between their meta-feature sets.

In the online phase, a configuration for a previously unseen *Online Dataset* must be determined. For this purpose, the *Extract Meta-Features* step is performed first, where the same set of meta-features as in the offline phase are extracted from the *Online Dataset*. Afterwards, the *Meta-learning* component uses the extracted meta-features to *Return c Configurations*, which are the warmstart configurations. For that purpose, it searches for the most similar offline dataset that was used in the offline phase based on the results saved in the *Meta-DB*. Here, the similarity is measured as the distance of the meta-features from an offline dataset to the meta-features of the online dataset.

4.2 Optimizer Loop

The *Optimizer Loop* in Figure 4.1 executes several iterations of an optimizer, which is an approach to approximate solid solutions for huge search spaces. To this end, the black-box function f must be implemented that takes as input a configuration from the *Configuration Space*, which denotes the search space. The function f has typically no closed form solution, which means there is no direct relationship between the input and the output. That is, the output cannot be calculated directly and in consequence it is not possible to calculate the gradients of f [BCD10]. In addition, evaluating f is very expensive for every configuration, which is why the optimizer approximates f by evaluating only some configurations instead of the whole configuration space. However, the goal of the optimizer is to optimize the function f. To this end, the optimizer performs three steps:

(1) *Select Configuration* c_i selects the next configuration to be executed. For this, different methods exist to select a configuration, which distinguishes the various optimizers. An overview of some optimizers with their approaches of selecting configurations is given in Section 2.2. In addition, in the first iterations, the optimizer selects in each iteration a configuration that is set in the *Initialize Configurations* step. (2) *Execute* c_i executes c_i on the *Training Data*. The results of this step are usually labels that, e.g., present the class affiliation for each data point. (3) *Evaluate* c_i evaluates the result of the executed configuration using the *Optimization Metric*. For supervised learning, the data is also divided into train and test data, because supervised learning methods include a training and an application phase [Bis06]. Methods like cross-fold validation can additionally be applied

to improve the generalization of the configuration. The result of step (3), i.e., the configuration with the value of the optimization metric, is saved in a database, which is the *Evaluations DB* in Figure 4.1. This is necessary to select the best configuration at the end.

The steps (1), (2) and (3) are executed iteratively until the budget is exhausted.

4.3 Return Best Configuration

The selection of the best configuration is typically done by selecting the configuration with the best value according to the optimization metric. There are also systems, e.g., Auto-sklearn, that use ensemble methods to combine configurations into an ensemble of configurations. For that, the best n configurations are used to build ensembles. The systems that do not use ensemble methods at the end, typically include them in the configuration space, which results in a significantly larger configuration space [FKE+15]. Subsequently, the step *Apply on Test Data* is performed, i.e., the configuration is used to classify test data.

5 Transfer of AutoML Concepts to Clustering

In Chapter 4, the typical concepts of existing AutoML systems were analyzed. In this chapter, a general concept that transfers the works from AutoML for supervised learning methods to clustering analyses is presented. To this end, the concept uses the components from the architectures of AutoML systems. However, the main challenge is to transfer these components in such a way that they are applicable for clustering.

This work focuses on partitional clustering algorithms due to their simplicity, low runtimes and the property that each iteration of the clustering algorithm yields a complete clustering result. In this context, complete means that each data point belongs to a cluster and subsequent iterations aim to improve the overall result.

Figure 5.1 shows the general concept for automatic algorithm selection and hyperparameter optimization for partitional clustering. This concept combines the different components of existing AutoML systems. In the following sections, each component of both phases and how they are adjusted in such a way that they can be used for clustering is explained. Therefore, Section 5.1

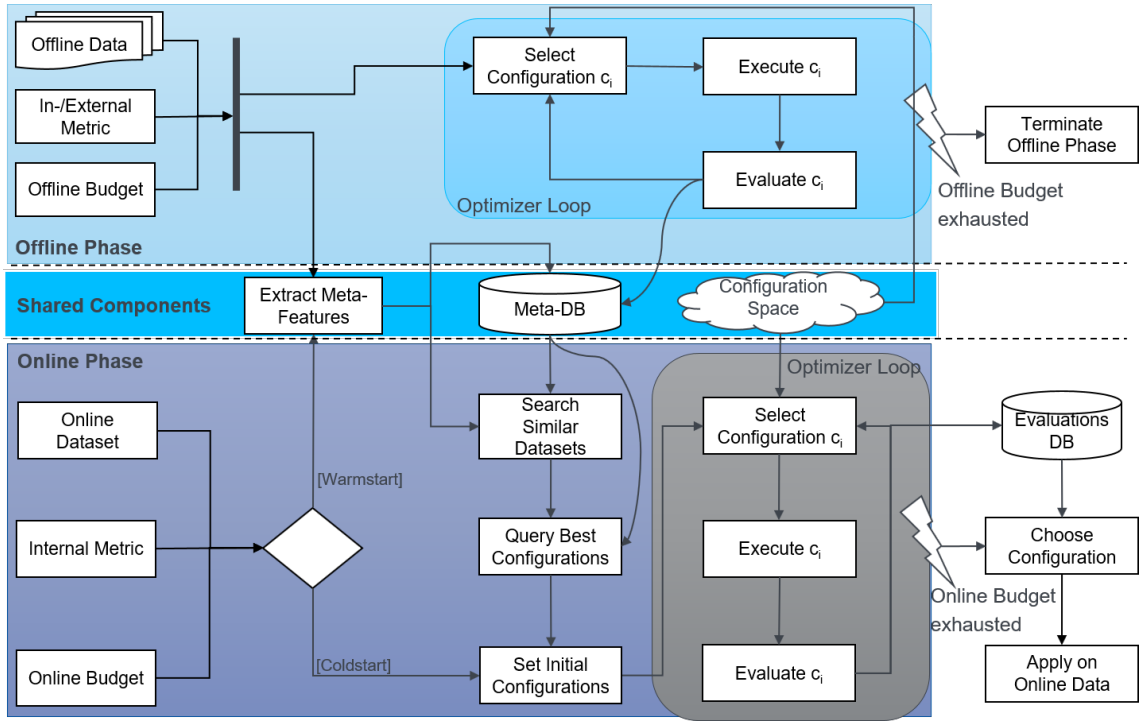


Figure 5.1: General architecture of the concept to combine the components from AutoML systems for clustering.

explains the *Shared Components*, i.e., components that are used by the *Offline Phase* and the *Online* phase. The specific components from the *Offline Phase* are addressed in Section 5.2. Section 5.3 describes the components from the *Online Phase*.

5.1 Shared Components

The *Shared Components* are the components that are used by the *Offline Phase* and *Online Phase* together. These components are the *Extract Meta-Features*, the *Meta-DB* and the *Configuration Space*.

5.1.1 Extract Meta-Features

The *Extract Meta-Features* step extracts meta-features from an input dataset, i.e., either *Offline Data* or the *Online Dataset*, and is an important component for applying meta-learning. Meta-features are meta information that describe characteristics of datasets. In AutoML systems, various meta-features are used that require the presence of ground truth labels. Although these meta-features can be used in the *Offline Phase* under the condition the *Offline Data* contains the ground truth labels, it is not beneficial to extract them. The reason is that the meta-features in the *Offline* and *Online Phase* must be identical. However, since clustering is an unsupervised learning task, the *Online Dataset* in the *Online Phase* does not contain any ground truth labels. To this end, only meta-features without the requirement of ground truth labels are applicable.

5.1.2 Meta-DB

The extracted meta-features from the *Extract Meta-Features* step are saved in the *Meta-DB*. In addition, the results from the *Optimizer* are saved in the *Meta-DB*. The information stored in the *Meta-DB* represents the knowledge of past evaluations and needs to be accessed by the *Online Phase* to apply this knowledge on an *Online Dataset*. Though the *Meta-DB* does only store the evaluations from the *Offline Phase*, it would also be possible to store the results from the *Online Phase*. That is the meta-features of each *Online Dataset* and the evaluated configurations. This information could be used to perform incremental learning [XLD+09], i.e., the system learns from each *Online Dataset*.

5.1.3 Configuration Space

The *Configuration Space* is the set $CS = A \times H$, where A is a set of clustering algorithms and $H = \{H_a | a \in A\}$, where H_a contains all hyperparameter values of an algorithm a. A configuration is an element of the *Configuration Space*. Since this work focuses on partitional clustering algorithms, the *Configuration Space* also contains only partitional clustering algorithms. This is especially relevant for optimizers that approximate a clustering algorithm like Hyperband or BOHB since they execute multiple configurations in parallel in one Successive Halving run (see Section 2.2). In this case, the number of iterations of a partitional clustering algorithm can be used for the approximation.

However, other kinds of clustering algorithms can also be used, but then the approximation of the clustering results must be done in a different way. This is particularly difficult when they do not proceed in an iterative manner.

Moreover, the *Configuration Space* is the same in the *Offline Phase* and the *Online Phase* in this work. Although it would be possible to use different spaces, this procedure should be treated with caution. If the *Offline Phase* contains more configurations than the *Online Phase*, these configurations cannot be used in the *Online Phase*, which means it would be useless to select and evaluate them. On the other hand, if the *Configuration Space* of the *Online Phase* is larger, the results of the offline phase do not contribute much, as they only cover a part of the search space.

5.2 Offline Phase

This section describes the *Offline Phase* of the concept which is shown in Figure 5.1. First, the inputs for the *Offline Phase* are described. Subsequently, the *Offline Phase* performs two procedures: *Extract Meta-features* and the *Optimizer Loop*. Since the *Extract Meta-Features* step is already described above (see Section 5.1.1) solely the inputs for the *Offline Phase* and the *Optimizer Loop* are described.

5.2.1 Inputs

The *Offline Phase* draws on three inputs. The first are the *Offline Datasets*. These can be datasets that contain the ground truth clustering labels for each data point. This is a difference to AutoML systems since it is also possible to use datasets that do not contain external information. However, the second input, the *In-/External Metric* depends on this choice. An internal metric evaluates clustering results only based on the internal structure, while external metrics compare clustering results to the ground truth clustering labels. This means external metrics are only applicable if the *Offline Data* contains the ground truth clustering labels [HBV01; KC88]. The third input for the offline phase is an *Offline Budget*. This specifies the budget for the *Optimizer Loop* in the *Offline Phase* to find suitable configurations. This can be any kind of budget like, e.g., a time constraint or to use a subset of the data. However, this work relies on the number of loops of an *Optimizer Loop* as budget.

5.2.2 Optimizer Loop

The second procedure in the *Offline Phase* is the execution of an optimizer framework, the *Optimizer Loop*, which requires the implementation of a black-box function f. Subsequently, the *Optimizer Loop* optimizes f by running multiple iterations until the *Offline Budget* is exhausted. That is, a maximum number of iterations is reached, i.e., the *Offline Budget* is exhausted, and then the *Offline Phase* is terminated. Although the same optimizers as in AutoML systems can be applied for clustering, the black-box function f must be adjusted. In the AutoML systems, f is implemented by two procedures; (a) execute c_i, i.e., execute a with the hyperparameters $h_1, \ldots, h_m$ on a dataset from the *Offline Datasets* and (b) evaluate the result of c_i with an external metric, e.g., classification accuracy. The output of f is then the score of the external metric.

These steps can also be applied for clustering and hence each iteration in the *Optimizer Loop* performs the following three steps consecutively: (1) *Select Configuration* c_i, (2) *Execute* c_i and (3) *Evaluate* c_i.

(1) Selects a configuration $c_i = (a, h_1, \ldots, h_m)$ from the *Configuration Space*. There, m is the number of hyperparameters for an algorithm a. Since this work focuses on partitional clustering algorithms the hyperparameter for each algorithm is only the number of clusters k, i.e., $m = 1$. However, when using different kinds of algorithms with different hyperparameters, the approach from Thornton et al. can be used. They rely on a hierarchical method to first select the algorithm and dependent on the algorithm the hyperparameters [THHL13].

Subsequently, the configuration c_i is used for step (2). Here, c_i is executed on a dataset of the *Offline Datasets*, i.e., execute a with the hyperparameters $h_1, \ldots, h_m$ on the dataset. For clustering, this is implemented in the same way, but instead of running a supervised learning algorithm, a partitional clustering algorithm is executed.

Step (3) evaluates the result of (2). It uses a clustering metric (see Section 2.3.2) to evaluate the result of the clustering algorithm. In contrast to AutoML systems, this can also be done by using an internal metric. Regardless of the metric used, the score of this metric is stored in the *Meta-DB*. This information of the performance of each configuration is utilized in the *Online Phase*.

The aforementioned steps are executed iteratively until the *Offline Budget* is exhausted. Once this happens, the *Offline Phase* terminates. The result of the *Offline Phase* that is used by the *Online Phase* are the extracted meta-features from *Offline Datasets* and the evaluated configurations for each offline dataset. This information can be accessed in the *Online Phase* by querying the *Meta-DB*.

5.3 Online Phase

In order to find resilient initial configurations for the *Online Dataset*, the data from the Meta-DB can be utilized. A detailed discussion about this can be found in Section 5.3.3 respectively Section 5.3.4. Moreover, each step of the *Online Phase* in Figure 5.1 is explained in the following.

5.3.1 Inputs

The inputs for the *Online Phase* are similar to the ones from the *Offline Phase*. Three inputs are required, which are the *Online Dataset*, an *Internal Metric* and an *Online Budget*. In contrast to the supervised AutoML systems, the *Online Dataset* does not contain any information about ground truth labels. Another difference is the used optimization metric. While this is typically an external metric in existing AutoML systems, this must be an internal metric for clustering. The *Online Budget* does not have any limitations regarding its appliance on unsupervised learning regarding the kind and the amount of the budget. However, it makes sense to use the same kind as for the *Offline Budget* and typically, the *Online Budget* is significantly lower as the *Offline Budget* since in the *Online Phase*, resilient results should be achieved faster.

5.3.2 Warm- or Coldstart

This step represents two variants included in this concept:

(1) *Warmstart* refers to setting initial configurations for the optimizer in advance. In this work it is the same as using meta-learning for setting initial configurations. However, this could also include other methods for warmstarting, e.g., setting initial configurations with transfer-learning or even from domain knowledge [BGSV08].

(2) *Coldstart* means no initial configurations are set in advance and the optimizer is applied directly. Since the AutoML systems typically set these configurations randomly, this is also applied here.

Depending on this decision, meta-learning in form of warmstarting is applied or not. If meta-learning is applied, then *Extract Meta-Features* (see Section 5.1.1) is the next step. Otherwise, the next step is *Set Initial Configurations* (see Section 5.3.5).

5.3.3 Search Similar Datasets

In this step the extracted meta-features from the previous step are used to search for similar datasets in the *Meta-DB*. To this end, the same procedure as in AutoML systems is applied. That is, the L_1 distance from the meta-features of the *Online Dataset* to the meta-features of each dataset of the *Offline Data* is calculated. The most similar datasets are used subsequently for the next step *Query Best Configurations*. However, the number of similar datasets that are used for the next step is not clear. Although Feurer et al. choose 25 datasets, this choice appears to be arbitrary [FKE+15].

5.3.4 Query Best Configurations

From the most similar datasets of the previous step, the best c configurations are queried from the *Meta-DB*. That is possible because the *Meta-DB* has stored all configurations for the *Offline Data* with their corresponding performance. The "best" c configurations are the ones with the best scores of the used metric in the *Offline Phase*. If several metrics are used in the *Offline Phase*, the value of two metrics cannot be directly compared. In particular, when internal metrics are used as most of them are not normalized. However, the results from external metrics can be utilized in this step to query the best configurations. Though, even when using external metrics, their scores are not directly comparable as, e.g., a score of 0.7 can have different meanings for different metrics. In this case, i.e., when the result from multiple metrics from the *Offline Phase* are considered, one approach is to select the metric based on the lowest k deviation. There, the lowest k deviation is the difference between the predicted value and the actual value of an offline dataset. Hence, for this approach, only datasets that contain the actual number of clusters are applicable.

5.3.5 Set Initial Configurations

This step sets c configurations that are first selected by the *Optimizer Loop* in the *Online Phase*. If *Warmstart* was selected previously, then the initial c configurations are the result from the *Query Best Configurations* step. These are the c configurations with the best performance on the most similar *Offline Dataset*. If *Coldstart* was selected, then these c initial configurations are set randomly since this is also typically done in AutoML systems [FKE+15; THHL13].

5.3.6 Optimizer Loop

After the initial configurations are set, the *Optimizer Loop* is executed. This is the same *Optimizer Loop* that is already used in the *Offline Phase*, i.e., an optimizer that executes three steps iteratively until the budget is exhausted. That means the same steps are performed as in the *Offline Phase*, the same *Configuration Space* and black-box function f is used. However, there are two differences; Firstly, the *Online Budget* is used instead of the *Offline Budget*, where the latter should be extremely larger in order to obtain solid results in the *Meta-DB* from the *Offline Phase*. Also, the *Optimizer Loop* is executed on the *Online Dataset* and the evaluation must be done with an *Internal Metric*, since no labels are available in the *Online Dataset*. In addition, assuming warmstart is applied, the *Optimizer Loop* does not start the first iterations from scratch, but uses the warmstart configurations. These are promising configurations on a similar dataset, which helps the optimizer to get more resilient configurations faster than if it started with random configurations. The results of all selected and evaluated configurations must be stored. For that purpose, Figure 5.1 has the *Evaluations DB*. There, all configurations and their performances are stored. These are used for the next step, which is *Choose Configuration*. Although the *Evaluations DB* is shown as database, any kind of storage system can be used here.

5.3.7 Choose Configuration

In this step, the best configuration is chosen from all configurations performed on the *Online Dataset*. These results are retrieved from the *Evaluations DB*. From there, the configuration with the best metric score is chosen. Subsequently, it is used for the step *Apply on Online Data*. However, it is also possible to choose the most resilient configurations for further processing steps as well.

5.3.8 Apply on Online Dataset

In contrast to AutoML systems, the chosen configuration is not applied on test data, but on the *Online Dataset*, which is the same dataset that is used within the *Optimizer*. The reason is that clustering does only contain an application phase and not a training phase [Bis06].

6 Protypical Implementation

This chapter describes a prototypical implementation of the concept described in Chapter 5. Therefore, Section 6.1 provides a general overview of the prototypical implementation. The implementation of the clustering algorithms and metrics that are used for the optimizers are explained in Section 6.2 and Section 6.3. Section 6.4 covers the implementation of the optimizers. The implementation of the meta-learning procedure is presented in Section 6.5. Section 6.6 explains the usage of the API of the prototypical implementation.

6.1 General Overview of the Prototype

The prototypical implementation can be found in a GitLab repository[1]. It uses the programming language Python [VD95], because of many available libraries, e.g., scikit-learn, that facilitate the implementation effort for all kinds of ML applications. The implementation contains 7 important packages, which are:

- **Algorithm**: This package contains the implementations for running the clustering algorithms. Currently this includes the k-Means [Mac67], k-Medoids [KR87] and GMM [Ras04] algorithms.
- **DB**: This package contains the whole code that is related to accessing the database. This is required to store the meta-features and the results of the optimizer runs. The *DBConfig.py* handles the connection to the database. The *DBService.py* is responsible for querying, inserting and updating any kind of entities from the database. The *DBSaver.py* saves the results from the optimizers in the offline phase, i.e., the meta-features and the evaluations of configurations into the database.
- **Metafeatures**: This package contains a list of the used meta-features as well as the Meta-feature Extractor which is responsible for extracting these meta-features. Additionally, the code for the k-Nearest-Neighbor (kNN) search, which is used to find the dataset with the most similar data characteristics, is implemented in this package.
- **Metrics**: Contains all code that is responsible for managing the metrics used for the optimizers. The *MetricHandler.py* file contains a collection of metrics together with a generic score function for each metric. For that, all currently available clustering metrics from scikit-learn are used[2].

[1] https://gitlab-as.informatik.uni-stuttgart.de/tschecds/automlclustering

[2] https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation

- **Experiment**: All code that is related to the evaluation in Chapter 7 can be found in this package.
- **Optimizer**: The implementation of all used optimizers can be found in this package. The implemented optimizers are random-search, Bayes optimizer, Hyperband and BOHB.
- **Utils**: Contains implementations for utility functions as well as file handling used across the other packages. All code that is related to handling files can be found in the *FileUtil.py* file. The *ResultDataCleaning.py* script manages a systematic output of the results.

6.2 Clustering Algorithms

The implementations of the clustering algorithms are taken from scikit-learn [PVG+11]. For this purpose, partitional clustering algorithms are used. These have in common that they partition a dataset iteratively into k clusters and the number of iterations can be limited. That is, this clustering family has the hyperparameter k in common. To this end, the k-Means and GMM implementations from scikit-learn are used. In addition, the k-Medoids algorithm is used. Since it is currently not implemented in scikit-learn, the implementation from the scikit-learn-extra[3] library is used. The k-Means algorithm is initialized with k-Means++ [AV07] and analogously is k-Medoids implemented with the k-Medoids++ [PJ09] method. To add a new algorithm, the file *ClusteringAlgorithms.py* in the package *Algorithm* must be adjusted. Firstly, a method for running the new algorithm must be implemented. This method must get the data (as numpy array) as input as well as the hyperparameters. Though, in the moment only the k hyperparameter, i.e., the number of clusters, is supported. Secondly, the *run_algorithm* method must be extended by an *if*-statement such that the method for the new algorithm is executed when the name is passed to the method.

6.3 Clustering Metrics

The implementations of all clustering metrics available in scikit-learn are used. These contain 6 external and 3 internal metrics. These are the same as presented in Section 2.3.2. For all metrics, a generic wrapper is implemented that calls the scoring function from scikit-learn but with some modifications. Firstly, all metrics that should be maximized, are multiplied with -1 to achieve a minimization problem for each metric. Secondly, due to the high runtime, the Silhouette Coefficient is only applied on a random sample of 10% of the data. This is done using random sampling, which is natively supported in scikit-learn. The metric wrapper can be found in the *MetricHandler.py* file. There, a metric can be added by creating a metric object in the *MetricCollection* class and adding this to the *all_metrics* list. This metric object must implement the *score_metric* method, contain a name and a type (either internal or external). If the type is internal, only the dataset without labels is used to calculate the score, otherwise the ground truth labels are used.

[3] https://scikit-learn-extra.readthedocs.io/

6.4 Implementation of the Optimizers

All optimizers are implemented as subclasses from the *AbstractOptimizer* class and implement the *optimize* method. Since the scikit-learn library is used, the scikit-optimize [HML+18] library is used to implement the random and the Bayes optimizer. The Hyperband and BOHB implementations are taken from [FKH18] and are also available on Github[4]. For all optimizer implementations, the configuration space must be defined, and the black-box function must be implemented. The latter one is done by running the clustering algorithm and evaluating the result with one of the metrics. To this end, the metric is passed as input when instantiating an optimizer. The configuration space is defined as a list of tuples with two elements, which are the name of the algorithm and a range of k values. Additionally, the dataset on which the optimizer is executed must be passed during the initialization as well as possible initial configurations. However, initial configurations are not natively supported for the Hyperband and BOHB implementations. To this end, Feurer et al. provided their Auto-sklearn implementation[5] that they used for the second AutoML Challenge [FEF+18].

For the implementation of Hyperband and BOHB, a minimum and maximum budget for each configuration must be defined. This budget is defined by the number of maximum iterations that clustering algorithms can perform. But other kinds of budgets could be used here as well, e.g., the runtime or a specific sample size of the dataset. The advantage of using the number of maximum iterations for the algorithm is that after each iteration it is guaranteed to get a clustering result where each data point belongs to a cluster. This means that each data point belongs to one cluster and the metric can be directly applied for the evaluation. This does not necessarily hold for the runtime. If the minimum budget for the runtime is too low and the first iteration of the algorithm is not finished, not all points are necessarily assigned to a cluster. This makes choosing appropriate budgets for the runtime very difficult since this is also heavily dependent on the characteristics of the dataset and the hyperparameter k.

When applying the concept described in Chapter 5, exactly one optimizer and one metric are used. The implementation of different optimizers and metrics is only for evaluation purposes. After all, it is not obvious what the best combination is for clustering.

6.5 Meta-learning Implementation

This section presents the implementation of the meta-learning procedure. For this, the implemented meta-features are described. Also, the implementations for the offline and online phase are depicted. These include specific details on how searching the most similar datasets and selecting the best warmstarting configurations is performed. Subsequently, the implementation of the Meta-DB with the used database schema is discussed.

[4] https://github.com/automl/HpBandSter

[5] https://github.com/automl/HpBandSter/issues/71

6.5.1 Meta-Features

The implementation of the meta-learning procedure draws on meta-features. The extraction of the meta-features is done using the *pymfe*[6] [RGS+18] library. This contains a set of meta-features that are used in popular AutoML systems. From this, all meta-features that do not require class labels are used. However, for a few datasets, it occurred that certain meta-features had *null* values. These values were removed after the extraction and replaced by the average value of the not-*null* meta-feature values. The code that is responsible for the meta-feature extraction can be found in the *MetaFeatureExtractor.py* file. The list of the used meta-features is in the *MetaFeatures.py* file.

6.5.2 Offline Phase

In the offline phase, the optimizers are executed on a set of offline datasets and with all implemented metrics. The results for each dataset are saved in the Meta-DB. Also, the meta-features are extracted from each offline dataset and saved in the same database. These meta-features are used for the online phase to find similar datasets based on an online dataset. For that purpose, a kd-tree is built on the meta-features of each offline dataset, which means the meta-features of one offline dataset represent one node in the tree. The reason for the use of a kd-tree is that the time to build the kd-tree does not bother that much as it happens in the offline phase. Also, the kd-tree allows an expected lookup of $O(klog(n))$ for the nearest neighbors in the online phase. In this context, k refers to the k nearest neighbors and n is the number of entities in a dataset. The nearest neighbors in this case are the similar datasets. The procedure for building the tree can be found in the file *KNNSearch.py*. The distance between datasets is defined by the Manhattan distance between the meta-features of the corresponding datasets. The resulting kd-tree is serialized in a *pickle*[7] file. The reason is that the tree also needs to be accessed in the online phase and this can be done efficiently by un-serializing this file.

6.5.3 Online Phase

In the online phase, the task is to find an appropriate configuration for clustering an unseen dataset. Therefore, the first step is to extract the meta-features with the meta-feature extractor. The extracted meta-features are then used to find the most similar datasets. Although the more than one similar dataset could be used, this implementation only considers the most similar dataset. This is done by un-serializing and using the already built kd-tree from the pickle file and looking up the most similar dataset. From the most similar dataset, the best c configurations must be determined. Since the offline phase executes all implemented metrics, all of them are utilized. However, only the best metric is used to find the best configurations. Determining the best metric by only comparing the score of the metrics is not possible, because they focus on different aspects. Therefore, the best metric is obtained by the one that achieved the lowest k deviation for the most similar offline dataset. This k deviation can be measured, because the offline datasets contain the actual number of clusters. If they do not contain them, it is not possible to use multiple metrics for the offline phase.

[6] https://pypi.org/project/pymfe/
[7] https://docs.python.org/3/library/pickle.html

All evaluated configurations from the offline phase are stored in the database. Therefore, the warmstarting configurations for the optimizer are queried from this database. To warmstart the optimizer, the initial configurations are set with the warmstarting configurations and the optimizer is then executed. In the online phase, the optimizer can only use internal metrics since the online dataset does not contain any ground truth labels.

6.5.4 Implementation of the Meta-DB

For the storage of the results of the optimizer runs the relation between the meta-features and the offline datasets as well as the relation to the optimizer results must be stored. These underlying relations can be well realized with relational databases. Also, once finished the schema is not intended to change. Therefore, a PostgreSQL[8] database is used. For accessing the PostgreSQL database instance, the Python library SQLAlchemy [Bay12] is used. The schema of the database is shown in Figure 6.1. Each entity in the database has an id. Each entity of the **offline_dataset** table has a file name and characteristics that describe it. These are the actual k value (*true_k*), the ratio of noise (*noise*), the number of features (*nr_features*) and the number of instances (*nr_instances*). The **offline_dataset** has a 1:n relation to the **metafeature** table which is modelled by the *dataset_id* foreign key in the metafeature table. Each **metafeature** has a name and a value. Moreover, the **offline_dataset** has a 1:n relationship to the table **optimizer_result**. One entity in the **optimizer_result** table represents the results for one optimizer, which includes the evaluations of different configurations with different metrics. Therefore, one optimizer result with one metric is represented by the **metric_result** table. This table saves the *k_deviation*, the lowest k deviation after the optmizer run and the *total_runtime*. The result for the metric evaluation of one algorithm for one k is saved in the table **metric_execution**. Additionally, for each **optimizer_result**, the warmstarting configurations are saved in the **warmstart_configuration** table. This table saves for each optimizer a number of warmstarting configurations. The pre-calculation and storage of the warmstart configurations is done in this way, because this only need to be done once in the offline phase and not for every unseen dataset in the online phase. The *metric* attribute in the table represents the name of the internal metric that is used in the online phase. The actual configurations are saved in the table **configuration**. Each row in this table contains the name of the algorithm (*algorithm_name*) and the value for the hyperparameter k (*k_value*). Also, the rank position (*rank_position*) is included, which indicates how good this configuration is as warmstart configuration. If c warmstart configurations must be selected, then all configurations with $rank\_position \leq c$ can be used. Hence, $rank\_position = 1$ indicates that this is the best configuration from the offline phase.

6.6 Usage of the API

This section unveils the use of the API of the prototypical implementation. To this end, there are some prerequisites that have to be met. These can be found in the *README.md*[9] file. After following the installation instructions there, the implementation can be used.

[8] https://www.postgresql.org/

[9] https://gitlab-as.informatik.uni-stuttgart.de/tschecds/automlclustering/blob/master/README.md

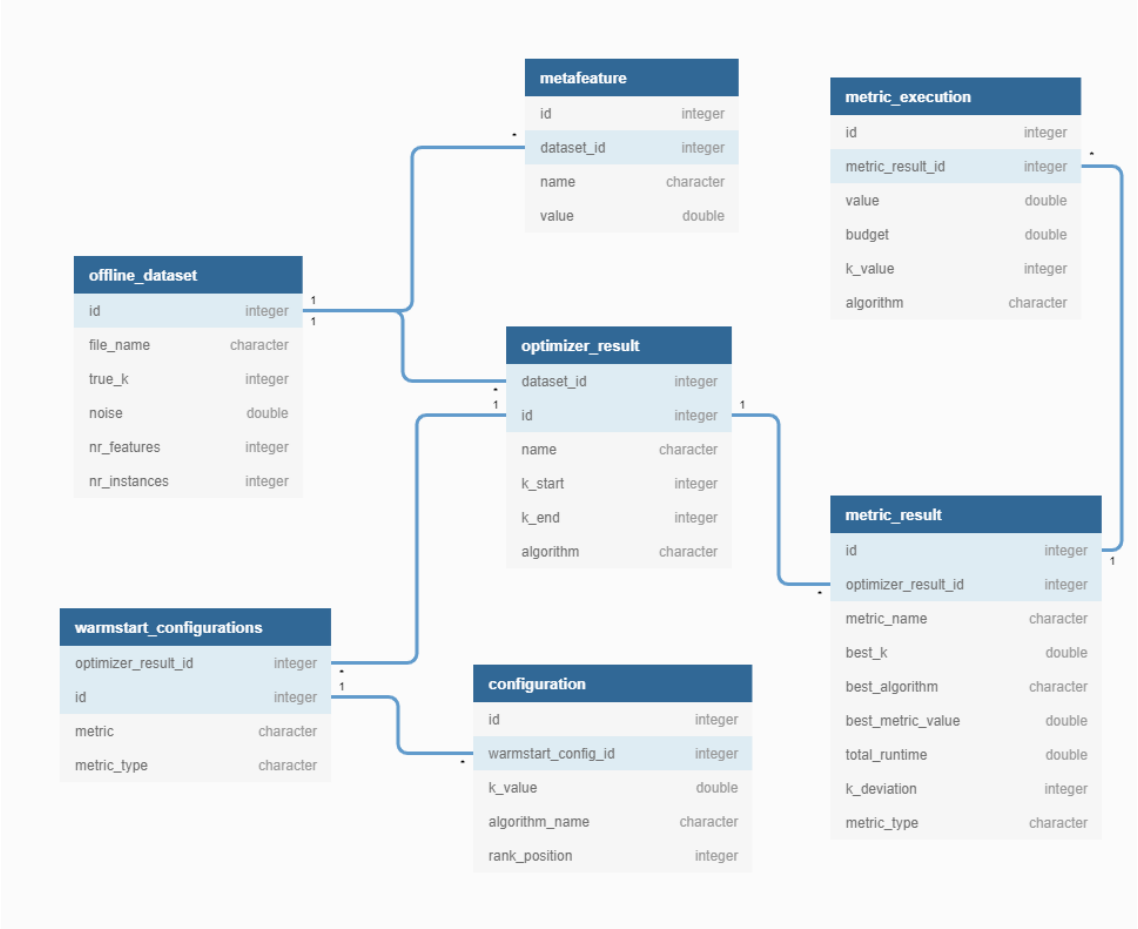


Figure 6.1: Overview of the database schema that is used to store the meta-features as well as the results of the optimizers.

An example of using the API is shown in Listing 6.1. First, the required libraries are imported in lines 1 - 5. In the lines 9 - 10, the *FileImporter* is used to load an online dataset. The online datasets can be found in the provided git repository as well. However, an arbitrary online dataset can be loaded as well. The *FileImporter* uses the path to the online datasets that are used for the evaluation in Chapter 7. For this example, the first dataset in the list of online datasets is used for the subsequent analysis (line 10).

For this dataset, the best c configurations are determined using meta-learning (line 14). The method *get_best_configurations* involves two main steps: (1) Searching for the most similar offline dataset by extracting the meta-features of the online dataset and (2) querying the best c configurations from the Meta-DB for the most similar offline dataset. In the example, the number of configurations c is set to $c = 10$ but could also be any $c \geq 1$ as long as $c \leq n_{iterations}$, where $n_{iterations}$ is the number of iterations to perform by the optimizer.

In lines 17 - 23 an instance of the *BayesOptimizer* is created. For this, a dataset and a metric are required. In this case, the dataset is the *online_dataset* and the used metric is the Davies-Bouldin index. However, there are also other metrics implemented that can be accessed by the *MetricCollection*. Note, that only internal metrics can be used here. The optional paramters are the

Listing 6.1 Example procedure for using the API.

```
1  from Algorithm import ClusteringAlgorithms
2  from Experiments.OptimizerOnline.algo_selection import Warmstart
3  from Metrics.MetricHandler import MetricCollection
4  from Optimizer.OptimizerAlgoSelection import BayesOptimizer
5  from Utils.FileUtil import FileImporter
6
7  # list the used online datasets and take one of them
8  # numpy array with shape n x d
9  online_datasets = FileImporter.list_online_data()
10 online_dataset = FileImporter.import_online_dataset(online_datasets[0])
11
12 # Optional: Define warmstart configurations.
13 # That are the configurations selected by Meta-Learning.
14 warmstart_configurations = Warmstart.get_best_configs(dataset=online_dataset, c=10)
15
16 # Create optimizer with the required parameters.
17 optimizer_instance = BayesOptimizer(dataset=online_dataset,
18                                     metric=MetricCollection.Davies_Bouldin,
19                                     # The dataset and metric parameters are required.
20                                     # The following are optional:
21                                     config_space=[ClusteringAlgorithms.algorithms, (2, 200)],
22                                     warmstarts=warmstart_configurations,
23                                     n_iterations=50)
24
25 # Perform the optimizer loop until the budget is exhausted.
26 result = optimizer_instance.optimize()
27
28 # Get the best configuration that is determined by the optimizer.
29 best_configuration = result.get_best_configuration(return_eval=False)
30
31 # Get list of tuples containing all configurations and their evaluations.
32 evaluations = result.get_all_configs(return_eval=True)
```

config_space, *warmstarts* and *n_iterations*. The *config_space* is optional, because if not specified the default one is used, which is the same as in the example, i.e. $k \in \{2, \ldots, 200\}$. The same applies for the $n_{iterations}$, i.e., the defalt for $n_{iterations}$ is set to 50. The *warmstarts* are optional, because if not specified, then no warmstarts are used, i.e., coldstarting is performed.

The execution of the optimizer is performed in line 26. Here, the optimizer performs the optimizer loop until the budget is exhausted. The budget is exhausted as soon as $n_{iterations}$ are performed. Line 29 shows how to acquire the best configuration that is found by the optimizer. By specifying the boolean parameter *return_eval*, the value of the metric for this configuration is also returned or not. Additionally, not only the single best configuration, but all executed configurations with their evaluations can be obtained like shown in line 32.

7 Evaluation

In this chapter, a comprehensive evaluation for the quantification of the proposed concept is done. In Chapter 6, a prototypical implementation of the concept is shown. However, the implementation contains various metrics and optimizers, but it is not yet clear which combination of optimizer and metric leads to accurate results in a reasonable amount of time. Moreover, the optimizers and the algorithms have parameters that affect the results. To this end, several datasets and parameters are used in two experiments. These are described in Section 7.1, which presents the experimental setup of the evaluation. The results of the offline phase are analyzed in Section 7.2. Section 7.3 discusses the results from the online phase. Here, the results for using coldstart, i.e., random selection of initial configurations of the optimizers, and warmstart via meta-learning are considered. The proposed concept is also compared to state-of-the-art methods for estimating hyperparameters of clustering algorithms in Section 7.4.

7.1 Experimental Setup

This section presents the design of the experiments that are performed. For this, the used hard- and software is presented. Subsequently, the datasets that are used in the offline and in the online phase are explained. Also, the setup of the optimizers is discussed, and the experiments are defined.

7.1.1 Hardware and Software

The experiments are performed on a virtual machine with a Linux distribution that operates on Ubuntu 18.04. It has a 6-core CPU with 2.5 GHz and 32 GB RAM. The prototypical implementation is based on Python. For this, a virtual python environment with Python 3.6.8 is used. The used libraries with their appropriate versions can be found in the *requirements.txt*[1] file in the GitLab repository.

7.1.2 Datasets

For the evaluation, offline and online datasets are required. The offline datasets are used in the offline phase to simulate past experiences with the evaluations on these datasets. Accordingly, the online datasets are used in the online phase. Both types of datasets are synthetically generated by

[1] https://gitlab-as.informatik.uni-stuttgart.de/tschecds/automlclustering/blob/master/requirements.txt

Table 7.1: The values of the input characteristics for the generation of the offline datasets divided into *Small*, *Medium* and *Large*.

Parameter	Small	Medium	Large
n	1,000	5,000	10,000
d	10	30	50
k_{act}	5	50	100
r	0	33	66

Table 7.2: The values of the input parameters for the generation of the online datasets divided into *Small-Medium*, *Medium-Large* and *Extra*.

Parameter	Small-Medium	Medium-Large	Extra
n	2,500	7,500	-
d	20	40	-
k_{act}	25	75	-
r	17	50	0

using the dataset generation tool described in [FS19]. All generated datasets that are used across the experiments are also in the GitLab repository[2]. The generation of the datasets is based on the following input characteristics:

- n: Number of instances of the dataset.
- d: Number of dimensions (or features) of the dataset. The value of each generated feature is in the interval [-10, 10].
- k_{act}: Actual Number of clusters of the dataset. Each cluster is generated according to the Gaussian distribution with mean at the center and standard deviation of 0.5. Also, each cluster contains $\frac{n}{k_{act}}$ instances.
- r: Ratio of outliers of the dataset. This means that $\frac{r}{100} \cdot n$ additional instances are added uniformly to the dataset.

For the generation of the offline datasets, the input characteristics are divided into three categories; *Small*, *Medium* and *Large*. Table 7.1 shows the value for each characteristic in each category. Since every value of one category is combined with the values of the other two categories, this results in 81 offline datasets.

The datasets for the online phase are generated by taking the average values between two consecutive categories for each characteristic. That is, the average value between *Small* and *Middle* as well as the average value between *Middle* and *Large* for each parameter. This is shown in Table 7.2. The only exception is the noise parameter, where one additional parameter is used. The value of 0% noise, i.e., no noise, is also considered, because the later evaluation unveils that the accuracy of results is dependent to the noise in the dataset. Since all datasets are synthetically generated, the

[2] https://gitlab-as.informatik.uni-stuttgart.de/tschecds/automlclustering/tree/master/datasets

Metric Type	Metric Name	Abbreviation	Reference
External	Adjusted Mutual Information	AMI	[VEB10]
	Adjusted Rand Index	ARI	[HA85]
	Completeness Score	CS	[RH07]
	Fowlkes-Mallows	FM	[FM83]
	Homogeneity	HG	[RH07]
	V-Measure	VM	[RH07]
Internal	Calinski-Harabasz Index	CH	[CH74]
	Davies-Bouldin Index	DBI	[DB79]
	Calinski-Harabasz Index	CH	[CH74]
	Sampled Silhouette Coefficient	SH10	[Rou87]

Table 7.3: Overview of internal and external metrics that are used in this work. The table shows the metric type (internal or external), the name of each metric as well as an abbreviation that is used in this work. Also, the literature reference for each metric is presented.

number of cluster as well as the assignment for each data point to its closest cluster is known in advance. However, the number of clusters and the assignments for each data point in the online datasets are only used for evaluation purposes.

7.1.3 Setup of the optimizers

The optimizers used for the evaluations are the Random, Bayes, Hyperband and BOHB optimizer. For the optimizers, several parameters must be defined. The configuration space contains the three clustering algorithms k-Means, k-Medoids and GMM. The hyperparameter for each algorithm is the number of clusters, k. The range R for selecting the k values is set to $R = \{2, \ldots, 2 * k_{max}\}$. There, k_{max} is the maximum k value of all generated datasets, i.e., $k_{max} = 100$. Although it is possible to set $R = \{2, \ldots, n-1\}$, this is not feasible in practice. The larger the k value is that the optimizer selects in a configuration the larger is the runtime. Therefore, it is assumed that the analyst employs his domain knowledge to limit R. This means the configuration space is $CS = \{\{k\text{−Means}, k - \text{Medoids}, \text{GMM}\}, \{2, \ldots, 200\}\}$. The GMM assigns probabilities for each data point that describe the likelihood that the point belongs to the cluster. To this end, the GMM algorithm is modified to return for each data point the cluster label with the highest probability.

All of the investigated clustering algorithms are executed in an iterative manner and the maximum number of loops can be defined. This is set to 10 for each algorithm, because Fritz et al. showed that with a proper initialization, e.g., k-Means++, a decent quality is already achieved after only a few loops [FBS19]. The other parameters of the clustering algorithms are left as the default scikit-learn parameters.

Besides the configuration space, the optimizers also need metrics that evaluate the results of the clustering algorithms. The metrics used for the experiments are shown in Table 7.3. These metrics are used in combination with each optimizer. Due to the high runtime, the Silhouette Coefficient is only applied of a random sample of the data, which is 10% of the data.

The budget for the optimizers is defined as the number of loops that the optimizer performs. However, this could also be any other budget like the runtime in seconds of the optimizer. The budget is set to 50 loops in the offline phase. This means that the optimizers in the offline phase are selecting, executing and evaluating at least 50 configurations. For Bayes and Random, these are exactly 50 configurations, for Hyperband and BOHB these are more configurations. The reason is that 50 loops of the optimizers refer to 50 Successive Halving (SH) iterations. For this, they need an additional budget which approximates the results for each configuration. In each SH iteration, multiple configurations are executed with the minimum budget and the best configurations are subsequently executed with more budget until only one configuration remains that is executed with the maximum budget. To this end, the number of loops of the clustering is used with minimum number of 1 and a maximum number of 10. This ensures on the one side that using the maximum number of loops is equal to running a configuration with another optimizer and on the other side that a complete approximation of the clustering result is guaranteed.

The number of optimizer loops for the online phase is set significantly lower since the goal here is to achieve satisfying results faster. This is because the offline phase must be executed only once, whereas the online phase is executed again for each online dataset.

In total, the offline phase performs at least 9 (# metrics) $\cdot$ 4 (# optimizers) $\cdot$ 81 (# offline datasets) = $2,916$ optimizer runs and $2,916 \cdot 50$ (optimizer loops) = $145,800$ runs of clustering algorithms respectively metric evaluations for each experiment. The online phase in contrast, performs 3 (# internal metrics) $\cdot$ 4 (# optimizers) $\cdot$ 24 (# online datasets) = 288 optimizer runs. The number of optimizer loops considered in the online phase are 4, 8 and 16. Due to the non-deterministic property of the clustering algorithms, each algorithm is also executed and evaluated 3 times. This means in the online phase $(4+8+16) \cdot 3 \cdot 288 = 24,192$ clustering algorithm executions respectively metric evaluations are performed for warmstart and coldstart separately.

7.1.4 Experiment Definitions

The evaluation comprises two experiments, which are performed separately from another. These are:

1. The **HPO** (hyperparameter optimization) experiment: This experiment uses the concept for evaluating the optimization of the hyperparameters of a single clustering algorithm. For this, the k-Means algorithm is used as clustering algorithm with the hyperparameter k.
2. The **CASH** (combined algorithm selection and hyperparameter optimization) experiment: This experiment additionally evaluates the automatic algorithm selection. To this end, the k-Means, GMM and k-Medoids algorithms are used. All have the number of clusters k as hyperparameter.

Note, that the HPO experiment is a special case of the CASH experiment as the only difference is that the configuration space is adjusted to only contain one algorithm. Both experiments include an offline and an online phase. Both are evaluated regarding accuracy and runtime. The runtime is measured by taking the median runtime for all offline and online datasets. For the evaluation of the accuracy, the difference of the actual k value of the dataset and the predicted k value of the optimizer is considered. This is feasible since the actual k value is known for all synthetic datasets. Equation (7.1) shows the computation of the k deviation for one dataset.

$$\Delta k = |k_{act} - k_{pred}| \tag{7.1}$$

There, k_{act} is the number of clusters in a given dataset and k_{pred} the predicted number of clusters. This value is calculated for each optimizer with each metric on each dataset. In the subsequent sections, median values are presented.

7.2 Offline Phase

The offline phase evaluates configurations on the offline datasets using each combination of optimizer and metric. However, the external metrics are not applicable in the online phase, because the online datasets does not contain the ground truth clustering labels. This means there are only internal metrics applicable. Though, the external metrics can be used for warmstarting an optimizer with an internal metric. The reason is that the metric with lowest k deviation from the offline phase is utilized to determine the warmstart configurations for the online phase. In particular, the results from either an external or an internal metric from the offline phase can be used to find warmstart configurations for an internal metric for the online phase. The metric choice depends on the accuracy of the most similar offline dataset. To this end, external metrics are used as well since the offline datasets, which are synthetically generated, contain the ground truth clustering labels for each dataset.

The goal of the offline phase is to find a suitable combination of optimizer and metric for clustering. In this sense the accuracy is especially relevant. The more accurate the results of the offline phase the more accurate can be the results in the online phase that use the best configurations from the offline phase. The runtime is typically negligible since the offline phase is executed only once. However, for the sake of completeness, and to have an idea of the runtime for optimizer and metric calculation, the results are presented in this section as well.

7.2.1 Accuracy

The results of the accuracy evaluation in terms of Δk for each metric and optimizer are shown for the CASH and the HPO experiment in Table 7.4.

For the HPO experiment (see Table 7.4a), all optimizers except Random have $\Delta k = 1$ as best result. Yet, Hyperband and BOHB perform in most cases better than Random and Bayes. Hyperband performs better for all metrics, except SH10, than Random and for most metrics better than Bayes. Also, compared to BOHB it has a lower deviation for 4/9 metrics, while they perform equally for 3/9 metrics. However, for each optimizer the best result is obtained with a different metric. In consequence, there is no metric that is best suited for each optimizer. Especially observable is that for each optimizer the best Δk value is related to an external metric, in particular, the AM, FM or VM metric. The only internal metric that also achieves the best Δk values for an optimizer is the DBI metric. It achieves them for Random and Hyperband. The HM and the SH10 metric seem to be unsuitable for each of the evaluated optimizers as they all produce high Δk values with these metrics. It is also noted that the internal metric CH attains solid results but only the combination with Bayes leads to a lower accuracy, namely $\Delta k = 11$. This is an example of how not only metrics and optimizers should be considered in isolation, but their combinations are crucial.

		Δk			
Metric Type	Metric	Random	Bayes	Hyperband	BOHB
External	AMI	4	**1**	2	**1**
	ARI	7	4	2	2
	CS	5	4	4	4
	FM	5	3	**1**	2
	HG	99	100	99	99
	VM	**3**	2	**1**	2
Internal	CH	6	11	2	3
	DBI	**3**	3	**1**	3
	SH10	23	41	44	39

(a) Δk results for HPO.

		Δk			
Metric Type	Metric	Random	Bayes	Hyperband	BOHB
External	AMI	4	4	**1**	**1**
	ARI	5	8	2	2
	CS	39	39	43	42
	FM	8	8	**1**	**1**
	HG	23	28	17	23
	VM	4	**1**	**1**	**1**
Internal	CH	4	21	**1**	2
	DBI	**2**	**1**	**1**	**1**
	SH10	10	**1**	7	6

(b) Δk results for the CASH experiment.

Table 7.4: Δk for each optimizer with each metric for (a) the HPO and (b) the CASH experiment. Bold results indicate the metric with lowest Δk value for an optimizer. Underlined values indicate the best value for each metric.

For the CASH experiment (see Table 7.4b), the best result for all optimizers except Random is $\Delta k = 1$. Like for the HPO experiment, the Hyperband optimizer performed best for most metrics compared to the other optimizers, followed by the BOHB optimizer. However, the optimizers achieve the best k deviations with more metrics as in the HPO experiment. The external metrics that achieve the best Δk values are the same as for the HPO experiment, i.e., the FM, AM and VM metric. In contrast to the HPO experiment, the Δk values for the HM metric improved but for the CS metric diminished. Nevertheless, the VM metric which is a combination of the HM and the CS metric still performs well. An improvement of the accuracy is also observed for the internal metrics. Here, the highest improvement is achieved by SH10. While the best Δk for the HPO experiment is 23, it is $\Delta k = 1$ for the CASH experiment. This is also reflected in the worst accuracy of SH10, which is 44 for the HPO experiment with Hyperband and 10 for the CASH experiment with Random.

Particularly outstanding from all metrics is DBI. It achieves the best Δk values for each optimizer. However, Bayes, Hyperband and BOHB also attain the best results with the VM metric. Hence, VM comes close to the DBI metric.

A surprising result at first glance is that for most metrics and optimizers better results for the CASH experiment are obtained than for the HPO experiment. Although the CASH experiment has a higher (in fact, three times higher) configuration space due to the three algorithms than the HPO experiment, CASH achieves better results. One explanation would be that the additional algorithms that are chosen from the optimizer are obtaining more resilient results than k-Means, e.g., k-Medoids for outliers. Another reason could be that the configuration space is not large enough to obtain robust results for both experiments [FKE+15; THHL13]. Besides this, because of the high number of optimizer executions in the offline phase, the experiments are only performed once, which means the better results of the CASH experiment could also be explained by the randomness of the optimizers and the algorithms.

The accuracy results of the optimizers and metrics are dependent on the noise in the datasets. This is shown in Figure 7.1. There, the results are representatively depicted for the Bayes optimizer with all metrics for the HPO and the CASH experiment. However, similar results are observed for the other optimizers as well. All metrics except CS accomplish a higher accuracy when only datasets without noise are used as when all datasets are considered. This does not only affect the k deviations, but also the minimum and maximum values. Especially for the HPO experiment there are maximum k deviations up to 195 when also using noisy data. However, they are improving to a maximum k deviation of 0 or 1 for several metrics, e.g., SH10 or DBI. The same is observed for the CASH experiment, though the improvement is not as drastic as for the HPO experiment. This is due to the reason that the results of the CASH experiment for the Bayes optimizer do not have that high maximum k deviations. They are only up to $\Delta k = 100$ for the CS, HG and CH metric.

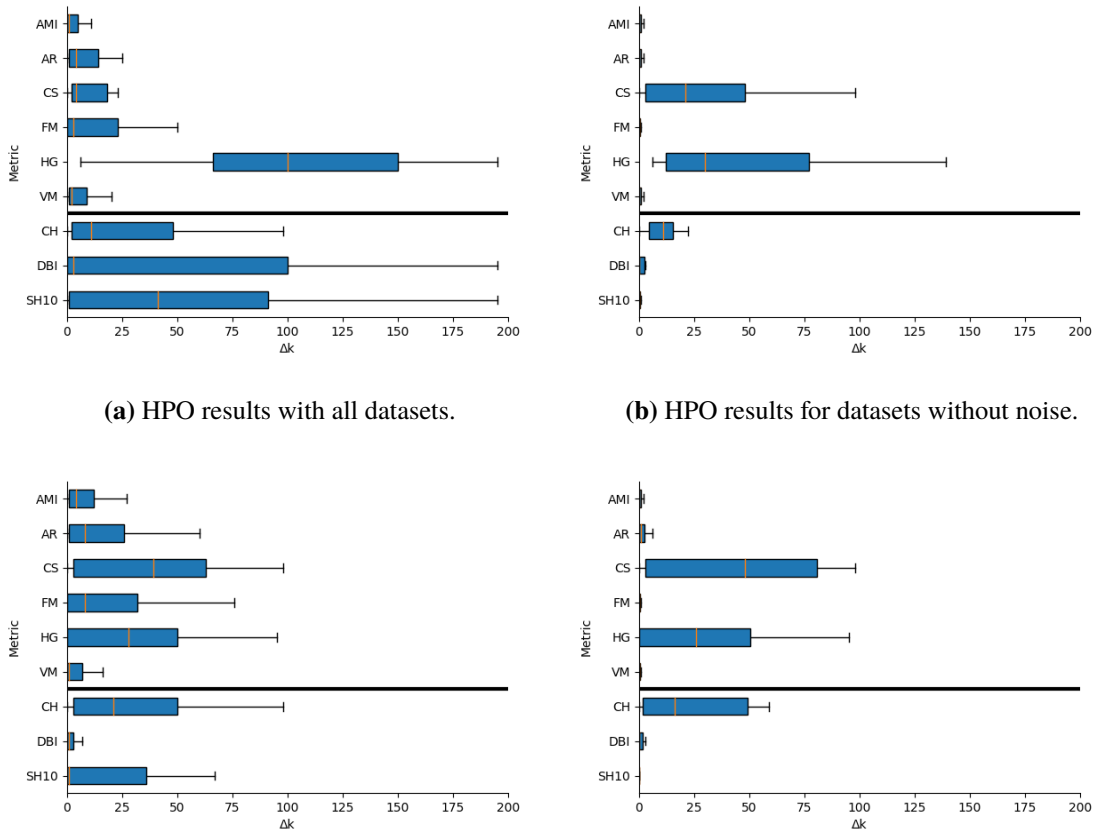
Concluding, the offline phase shows that optimizers and metrics can achieve good results for clustering algorithms. From metric perspective, VM and DBI obtain good results while from optimizer view Hyperband and BOHB perform better than Random and Bayes. Especially the performance of the DBI metric, which is an internal metric, promises accurate results for the online phase as only internal metrics are used there.

7.2.2 Runtime

In this subsection the runtime for the optimizers and different metrics are shown for the offline phase. Therefore, the median runtime results over all datasets are shown in Table 7.5.

For both experiments, Hyperband and BOHB have significantly higher runtimes than Random and Bayes. The reason is that both are performing 50 Successive Halving iterations, which means they are executing in total more than 50 configurations. Moreover, both implementations are designed for running on a cluster in parallel while the experiments are performed locally on a single virtual machine.

For the HPO experiment (see Table 7.5a), the CH metric achieves the lowest runtime for Bayes, Hyperband and BOHB. In addition, for 6/9 metrics, the Bayes optimizer has a lower runtime than Random, Hyperband and BOHB. Although the model-based optimizers (Bayes and BOHB) add additional overhead by updating the underlying model, they consider configurations that are closer



(a) HPO results with all datasets.

(b) HPO results for datasets without noise.

(c) CASH results with all datasets.

(d) CASH results for datasets without noise.

Figure 7.1: Boxplots for the Δk results for the Bayes optimizer for the HPO experiment when (a) all datasets and (b) only datasets without noise are used. In addition, the same results are shown for the CASH experiment for (c) all datasets and (d) datasets without noise.

to the assumed optimum. Since the k values in the considered range are relatively small, they can limit the search space better. This means they focus on lower k values than Random respectively Hyperband and hence they find configurations that are close to the actual k value faster.

The results for the CASH experiment (see Table 7.5b) are quite different. First, for each metric Random accomplishes the lowest runtimes. Also, the CS metric achieves for Random, Hyperband and BOHB the lowest runtimes, while CH accomplishes it for Bayes. However, as discussed in the previous subsection, these metrics also obtain very high Δk values, which means that a low runtime is not correlated to a good accuracy. In general, the runtimes for each metric and optimizer are considerably lower for the CASH experiment than for the HPO experiment. Especially for Hyperband and BOHB, the runtimes are only half as long. This is because it is observed that the k-Medoids algorithm has a significantly lower runtime than the k-Means algorithm and performs less iterations. This means that, e.g., for Random, the k-Medoids algorithm is chosen in roundabout 1/3 cases. Similar observations were made for the GMM algorithm, although it is not as significant as for k-Medoids. In contrast to that, the HPO experiment uses k-Means for each configuration.

		Runtime (s)			
Metric Type	Metric	Random	Bayes	Hyperband	BOHB
External	AMI	81.41	76.96	547.79	335.58
	ARI	**79.57**	77.36	522.09	346.33
	CS	81.41	58.59	518.76	320.32
	FM	85.33	75.84	525.96	353.85
	HG	79.69	132.18	622.93	754.62
	VM	80.02	79.34	555.31	400.87
Internal	CH	83.64	**57.04**	**505.82**	**270.15**
	DBI	83.74	83.89	609.22	476.24
	SH10	81.48	92.01	565.8	479.28

(a) Median runtime in seconds for HPO.

		Runtime (s)			
Metric Type	Metric	Random	Bayes	Hyperband	BOHB
External	AMI	59.88	90.16	245.47	226.54
	ARI	59.08	78.23	248.07	223.6
	CS	**55.17**	59.64	**223.58**	**150.94**
	FM	59.32	81.9	252.32	221.4
	HG	57.79	107.86	259.5	346.49
	VM	57.07	86.72	238.49	217.87
Internal	CH	55.61	**56.23**	246.49	161.76
	DBI	62.08	71.08	251.16	246.79
	SH10	61.33	90.47	254.79	303.19

(b) Median runtime in seconds for CASH.

Table 7.5: Runtime in seconds for (a) HPO and (b) CASH for each optimizer with the different metrics. Bold values indicate the lowest runtime for each optimizer and underlined values represent the lowest runtime per metric.

Concluding, the results of the offline phase show that the optimizers in combination with certain metrics can achieve an accuracy of $\Delta k = 1$ in a bit more than a minute, e.g., the Bayes optimizer with the DBI metric.

7.2.3 Relative Algorithm Frequencies for CASH

The relative algorithm frequencies show how often an algorithm is selected in the best configuration, i.e., the configuration that is returned by the optimizer per dataset. Here, the overall frequency across all datasets is presented for the CASH experiment. Note, that HPO only performs k-Means

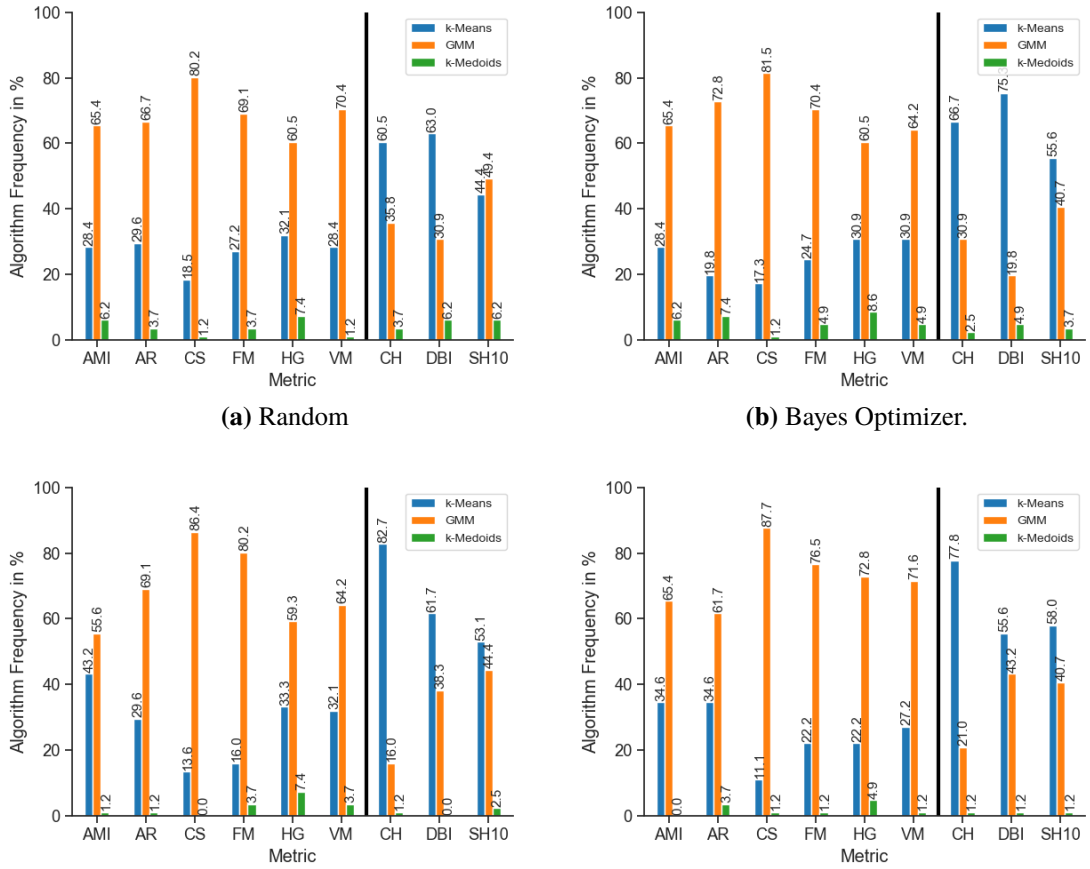


Figure 7.2: Result of the relative algorithm frequencies the CASH experiment in the offline phase. The results are shown for (a) Random, (b) Bayes, (c) Hyperband and the (d) BOHB optimizer, each for all investigated metrics.

executions on varying parameters. Since 81 offline datasets are used, this would mean that if an algorithm has 100%, it was chosen for all 81 datasets in the best configuration. Also, only the CASH experiment is examined since the HPO experiment contains a single algorithm.

Figure 7.2 shows the relative algorithm frequencies for each optimizer and metric. The external and internal metrics are separated by a black line in the figure.

For each optimizer, all internal metrics, which are the SH10, DBI and CH metric, prefer to choose k-Means over the other algorithms. The only exception is Random with the SH10 metric. In contrast, the external metrics choose all GMM most often as best algorithm. The k-Medoids algorithm is chosen very rare, i.e., in 7/81 cases. Although 2/3 of the synthetically generated datasets contain noise (1/3 has 33% noise and the other 1/3 has 66% noise) it is expected that the k-Medoids algorithm is chosen more often. Since the k-Medoids algorithm considers the medoid object of a cluster it is more robust to outliers [HKP12]. In particular, because k-Means and GMM are both sensitive for noise in the data [XW05]. However, the k-Medoids algorithm is minimizing the distances from each data point to its cluster medoid and not to the cluster center like k-Means. Since the internal

metrics are preferring results with a lower compactness, i.e., the distance to the cluster center, the k-Means results provide better metric values due to the synthetic cluster center. This also explains why the external metrics are choosing GMM more often than k-Means since their objective is not to minimize the compactness.

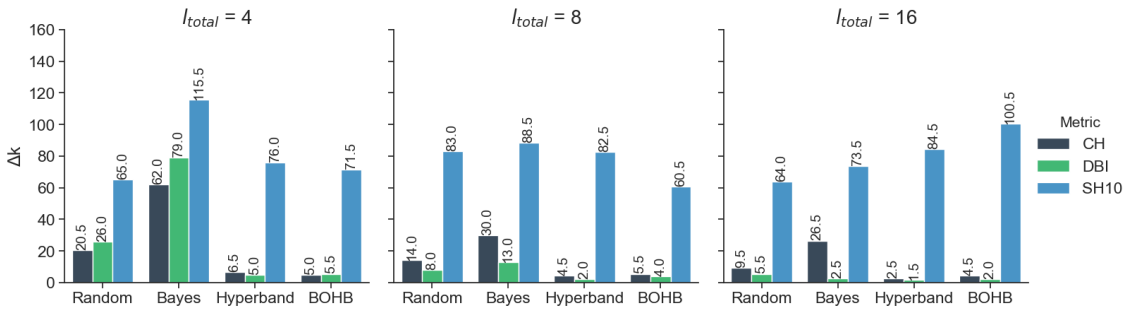
As shown in Table 7.4, the accuracy improves for the internal metrics for the CASH experiment compared to only choosing k-Means algorithm in the HPO experiment. Since the internal metrics in the CASH experiment also choose GMM for the best configuration, this leads to the conclusion that selecting GMM for certain datasets can improve the accuracy. However, for most datasets k-Means seems to be obtain better accuracy. The accuracy of the external metrics is not improving that much for the CASH experiment compared to the HPO experiment. They are choosing more often GMM in the best configuration than k-Means and they also choose GMM more often than the internal metrics. Since the accuracy of the external metrics is not improving that much for the CASH experiment compared to the HPO experiment as it improves for the internal metrics, this also supports the assumption that k-Means is better suited for most datasets than GMM. An observation that supports this thesis is that for Random and SH10 the only case occurs where GMM is chosen more often than k-Means for an internal metric. However, this is also the case where SH10 achieves the worst accuracy (see Table 7.4b).

As shown in Table 7.4b, the accuracy of the CASH experiment is better in most cases than for the HPO experiment. A reason for this can be that the GMM algorithm is also chosen regularly and this produces better results than k-Means for certain datasets.

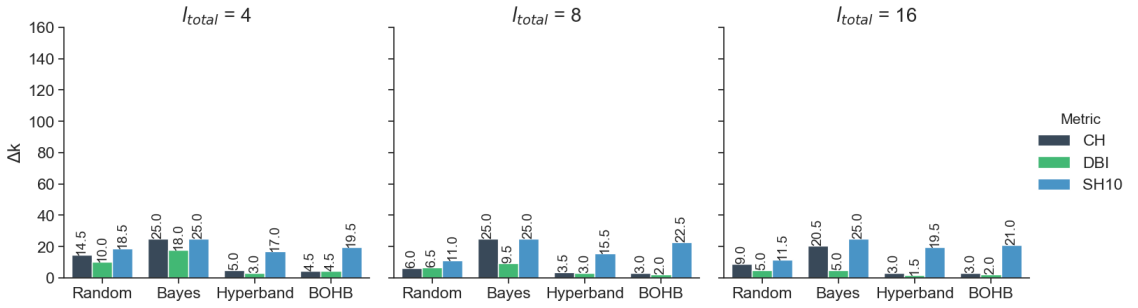
Moreover, the distribution of the selected clustering algorithm is similar for each optimizer when considering the same metrics. This leads to the assumption that the best-chosen algorithm is more dependent on the metric than on the optimizer. Yet, the accuracy of the different optimizers for the same metric still differs (see Table 7.4b) since the configuration also includes the k value and not only the algorithm.

7.3 Online Phase

In this section, the results for the online phase for the HPO and the CASH experiments are discussed. The online phase is the phase that is applied on a previously unseen dataset that is intended to be clustered and where no evaluations of past configurations are available for this dataset. A significant difference to the offline phase is that the results must be achieved faster while still obtaining resilient results. Thus, it is examined if the results are still as accurate with a lower budget, i.e., less optimizer loops. Also, since the online phase does not contain any information of the ground truth clustering labels, only the three internal metrics, DBI, CH and SH10 are applicable in the online phase. The evaluation of the online phase includes the evaluation of two procedures. The first is coldstart, which means the optimizers sets the initial configurations randomly and is discussed in Section 7.3.1. The second is warmstart, which uses meta-learning to set the initial configurations based on the experience of past evaluations from the offline phase. For this, the results from external metrics from the offline are also used to determine suitable warmstarting configurations. The results of this procedure are examined in Section 7.3.2. Both procedures are analyzed regarding the accuracy and the runtime. To this end, if a configuration is selected by an optimizer, then this configuration is executed three times and the configuration with the median Δk is selected. Also, if not stated



(a) The HPO experiment.



(b) The CASH experiment.

Figure 7.3: Accuracy results for coldstart in the online phase for the (a) HPO experiment and (b) the CASH experiment. For both experiments, the results are shown for $l_{total} = 4$, 8 and 16.

otherwise, the subsequent (sub-)sections show the median results across all datasets. For that purpose, 24 online datasets are used with different dataset characteristics (see Section 7.1). The results of the online phase for cold- and warmstart are discussed in Section 7.3.3.

7.3.1 Coldstart

The coldstart procedure sets the initial configurations of the optimizers randomly. This means that on the one hand side no advanced technique is used to set these configurations. On the other hand, there is no runtime overhead, e.g., for extracting meta-features. The budget for the optimizers varies, such that different number of loops are examined. These are $l_{total} = 4$, 8 and 16. The analysis is done regarding the accuracy and the runtime. It is also investigated how often the several algorithms are selected for the CASH experiment.

Accuracy

The accuracy results for the coldstart are shown in Figure 7.3 for the HPO and the CASH experiment. It can be observed that with increasing number of loops, the accuracy for each optimizer and metric, except the SH10 metric, is increasing. This is because the optimizers are selecting more configurations and hence also configurations that are closer to the actual k value of the online dataset.

In particular, the best improvement is observed for the Bayes Optimizer with the DBI metric. In general, for $l_{total} \geq 8$ the DBI metric attains the best accuracy for each optimizer. Especially, the combination of either Hyperbandand the DBI metric obtains always the best results.

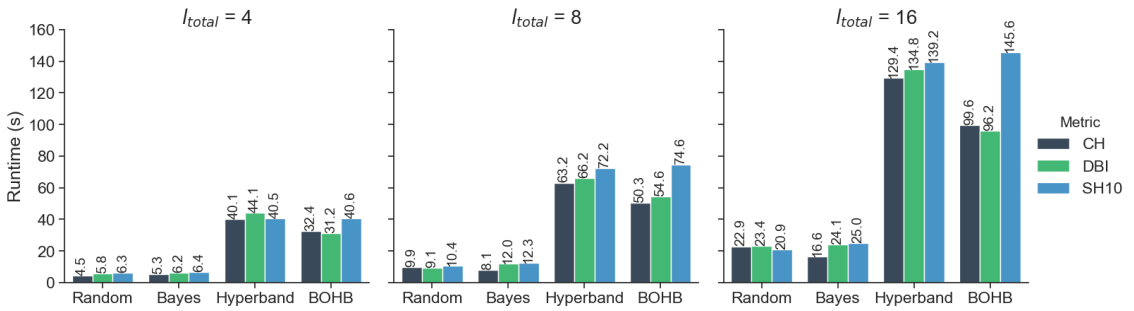
Comparing the different optimizers, the most accurate results are achieved with the Hyperband and BOHB optimizers for all l_{total} values. Since one loop for Hyperband and BOHB runs one SH iteration, they select multiple configurations in parallel with a lower budget in one loop. Hence, they are evaluating more configurations than the other optimizers. In addition, it can be observed that Random outperforms Bayes in almost all cases. The reason for this could be that the Bayes optimizer requires more configurations to build a model that predicts resilient configurations. As related work states, when the configuration space is not large enough, and the number of loops is not high enough that Random outperforms Bayes [BBBK11]. However, in this case, warmstart could be beneficial by improving the performance of the Bayes optimizer as the first configurations are selected in promising regions [FSH15].

The DBI metric obtains better results than the CH and the SH10 metric. The bad accuracy of SH10 can be explained by the fact that the metric is only applied on a random sample with 10% size of the original dataset. A reason for the better performance of DBI in contrast to CH could be that the DBI metric only considers the cluster that achieves the maximum value of the quotient of separation and compactness. The separation here means the one with the maximum separation to another cluster. This means, although if one cluster contains noise, this does not necessarily affect the maximum quotient of compactness and separation. In contrast, with increasing noise in the dataset the compactness measure of the CH metric increases as it considers the compactness of all clusters. Liu et al. also showed that the results of CH are instable in contrast to DBI when the dataset contains noise [LLX+10; LLX+13].

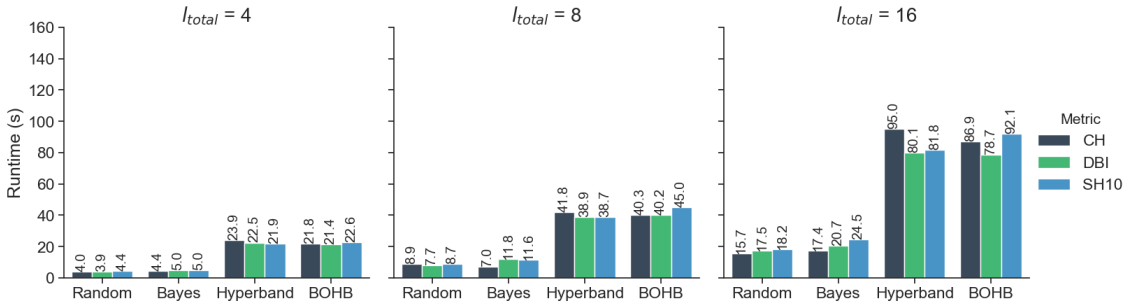
Moreover, similar to the offline phase, the results of the CASH experiment have a higher accuracy than the ones from the HPO experiment. There are only a few cases (in fact 3/36 cases from all loops and for each optimizer and metric combination) where HPO achieves a higher accuracy. The only difference between the HPO and the CASH experiment is the configuration space, which contains two additional algorithms for the CASH experiment. Hence, the higher accuracy must be related to these additional algorithms.

Runtime

The results for the runtime for the HPO and the CASH experiment are shown in Figure 7.4. There, it can be seen that the runtimes for the experiments are increasing with increasing l_{total} for each optimizer and metric. It is also increasing linearly such that the runtimes for $l_{total} = 8$ ($l_{total} = 16$) are almost twice as large as for $l_{total} = 4$ ($l_{total} = 8$). In addition, the Hyperband and BOHB optimizers have always higher runtimes than Bayes and Random, in fact more than 5 times as large. This is due to the fact that Hyperband and BOHB are selecting more than l_{total} configurations because they are running l_{total} Successive Halving iterations. Also, both implementations are designed to run in parallel. The runtime of the CH metric and the DBI metric barely differ in most cases. However, the SH10 metric has in most cases the highest runtime, although it is only calculated on a random sample subset with 10% size of the original dataset. The reason for this is that the SH10 metric calculates the distances from each point to each other point in the same cluster.



(a) The HPO experiment.



(b) The CASH experiment.

Figure 7.4: Median runtime results for coldstart for (a) the HPO experiment and (b) the CASH experiment. For both, the results are shown for $l_{total} = 4,\ 8,\ 16$.

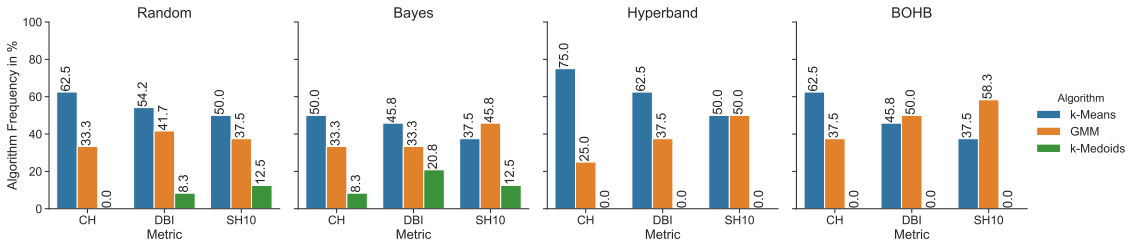
The CH and DBI metrics only consider the distance from each point to the affiliated cluster center. This illustrates that it is not feasible to calculate the SH10 metric on the whole dataset since the runtime would be at least 10 times higher, if not even more.

Like in the offline phase, the runtimes for the CASH experiment are lower than for the HPO experiment. This can be explained by the runtimes of the additional algorithms GMM and k-Medoids, which are observed to be lower than for k-Means. However, the runtime does not only depend on the algorithm but also on the k values that the optmizers select. If the number of actual clusters in a dataset is relatively high, the optimizers and metrics will have higher runtimes and vice versa since the runtime of the algorithms is dependent on the number of clusters. The actual number of clusters of the online datasets is relatively small in both experiments. In consequence, the lower runtimes for the CASH experiment in comparison to the HPO experiment can also be explained by the higher accuracy.

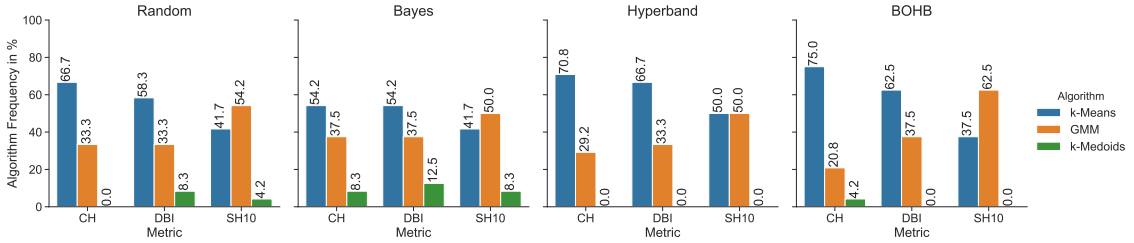
Frequency of best-performing Algorithms for CASH

Figure 7.5 unveils how often each of the three algorithms are selected in the best configuration.

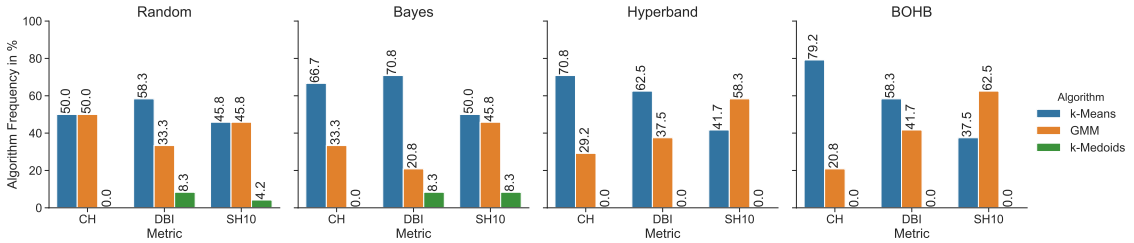
It can be seen that the frequency for the CH and the DBI metric is for k-Means in nearly all cases more than 50% compared to the other algorithms for each optimizer and each l_{total}. The reason is that these metrics have the objective to minimize the compactness and the separation. Especially k-Means focuses on the same goal of minimizing the compactness. For SH10, GMM is chosen



(a) $l_{total} = 4$.



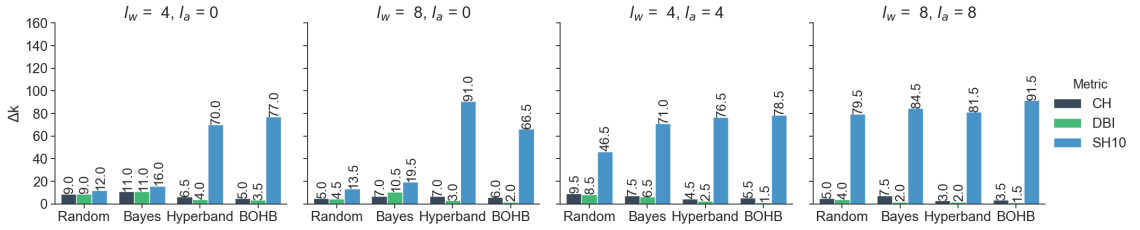
(b) $l_{total} = 8$.



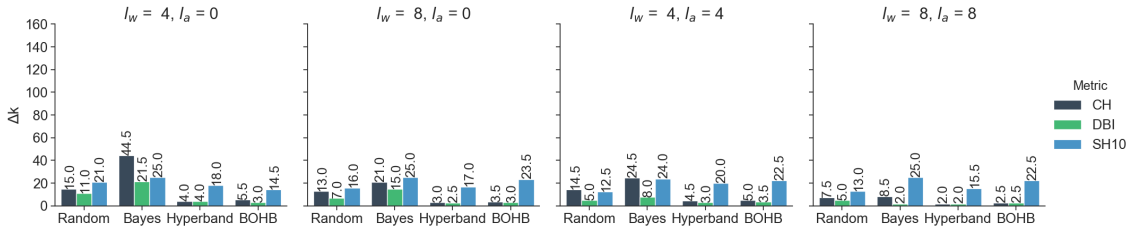
(c) $l_{total} = 16$.

Figure 7.5: Algorithm frequency in percent for the CASH experiment with (a) $l_{total} = 4$, (b) $l_{total} = 8$ and (c) $l_{total} = 16$. The results are shown for each optimizer and metric combination for each of the three clustering algorithms.

most frequently. The reason for this is that compared to the CH and DBI metrics, the SH10 metric does not focus on the compactness of the clusters but on the silhouettes of every data point. For similar reasons, the k-Medoids algorithm is chosen less frequently, i.e., mostly in no cases. Because the objective of k-Medoids is to minimize the distances of each data point to the medoid of the affiliated cluster instead of the distances to the cluster center. Moreover, with an increasing number of optimizer loops, the frequency for k-Medoids decreases further. Also, Hyperband and BOHB, which are executing more configurations than the other optimizers, do not choose k-Medoids, except for one case. This leads to the assumption that with a greater number of loops and consequently more configurations that are executed and evaluated, the frequency of k-Medoids is infinitesimally low. Though, this can also be due to the form of the clusters in the synthetic datasets. These all follow a Gaussian distribution and it has to be examined if the same behavior occurs for other kinds of clusters and data characteristics as well.



(a) The HPO experiment.



(b) The CASH experiment.

Figure 7.6: Accuracy results for (a) the HPO experiment and (b) CASH experiment for only using warmstart configurations and warmstart configurations with additional loops.

7.3.2 Warmstart

In contrast to coldstart, warmstart means that the initial configurations are determined via meta-learning. For this, it uses the evaluations from the offline phase on the offline datasets. For the existing AutoML systems it is shown that this technique can improve the performance regarding accuracy and runtime of the optimizers. However, in this subsection the application of meta-learning for clustering is examined regarding the accuracy and the runtime. In this context, the initial or warmstarting configurations are determined by finding the most similar offline dataset for a given online dataset. Subsequently, the (internal or external) metric with the lowest Δk from the offline phase for a given optimizer is determined and from this metric, the best l_w configurations are chosen as warmstart configurations. To this end, different number of optimizer loops and different number of warmstart configurations are evaluated.

The number of warmstarts is denoted by l_w and the number of additional loops is denoted by l_a. This means that the optimizer performs in total $l_{total} = l_w + l_a$ loops. For the number of warmstarts the values 4 and 8 are considered. For the number of additional loops of the optimizer, two cases are examined: (1) $l_a = 0$: Only using the warmstart configurations and no additional loops. (2) $l_a = l_w$: Using the warmstart configurations and the same number of additional loops like the number of warmstart configurations.

Accuracy

The accuracy results for the HPO experiment and the CASH experiment are shown in Figure 7.6.

The first observation is that with the lowest number of total loops a high accuracy is obtained. That is when using 4 warmstart configurations without additional optimizer loops. This is especially observable for the BOHB and Hyperband optimizer. When increasing the number of total loops, either with more warmstart loops or additional loops, the accuracy is also increasing. The reason is that the optimizer can execute more configurations.

For the warmstart loops without additional loops, i.e., for $l_w = 4$ and $l_w = 8$, each optimizer executes the same warmstart configurations independently of the metric. The reason is that only the results from the best metric of the offline phase are considered for this optimizer. However, the accuracy still differs between different metrics. This means that although the metrics evaluate the same configurations, they evaluate these configurations quite differently. This can especially be observed for the SH10 metric which has in almost all cases the lowest accuracy. The bad performance of the SH10 metric can be explained by the fact that is evaluated only a random sample of 10% from the original dataset. In contrast, the DBI metric attains always the highest accuracy. The reason is that it is not as sensitive to noise as the CH metric [LLX+10; LLX+13].

Considering only warmstart configurations, Hyperband and BOHB do not only execute the warmstart configurations but they execute l_w optimizer loops. This means they run l_w Successive Halving iterations and in consequence execute more than l_w configurations. Due to this, the results for them differ greatly from the ones for Random and Bayes, which only execute l_w configurations.

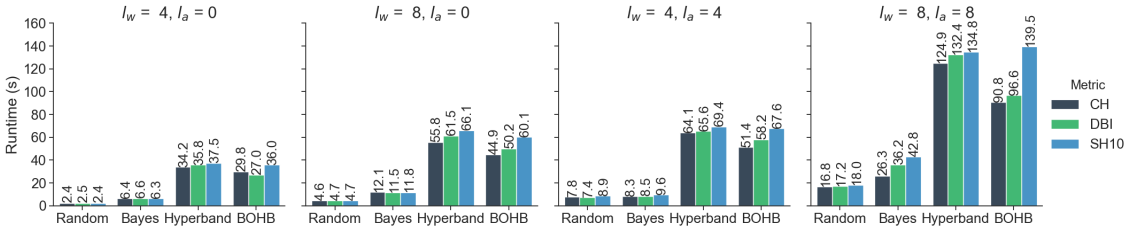
Although the CASH experiment attained a higher accuracy than the HPO experiment for the offline phase and for coldstarting, the HPO experiment has a higher accuracy for warmstarting (except for the SH10 metric). A reason for this could be that the warmstart configurations are not necessarily the best configurations for the online dataset since they only performed best on a similar offline dataset. In consequence, due to the higher configuration space of CASH, it is less probable that the warmstart configurations are closer to the best configuration. Another reason is, that the accuracy only states the deviation of the best configuration but not how well suited the second, third etc. warmstart configuration is.

Moreover, for the HPO experiment, when comparing the warmstart results to the coldstarting results, the greatest improvement can be observed for the Bayes optimizer. In particular, for warmstarts with additional optimizer loops. The reason is that the Bayes optimizer requires several iterations until the underlying model can predict resilient configurations. Because of the warmstarting configurations the model is already well fitted and can predict resilient configurations. These improvements are also observable for BOHB, which uses the same underlying model as Bayes. Yet, the improvement is not so vast since the accuracy is already quite high for coldstart.

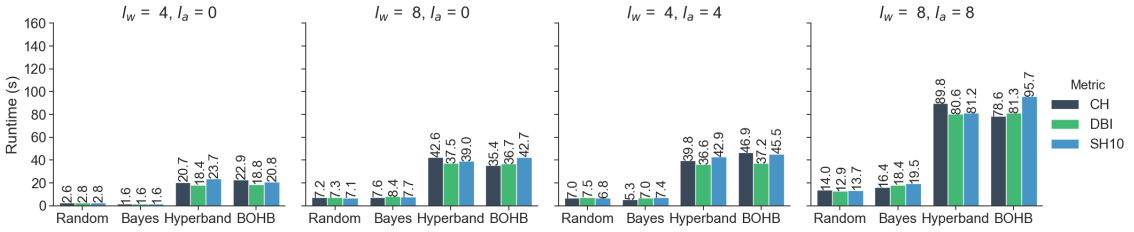
Runtime

The median runtime results are shown in Figure 7.7. There, the results for each optimizer and metric for both experiments are depicted.

For both experiments, the runtime is almost twice as large when the number of total optimizer loops is doubled. This means the it seems that the runtime is linearly increasing with the number of optimizer loops. However, note that the runtime is dependent on the configuration that the optimizers



(a) The HPO experiment.



(b) The CASH experiment.

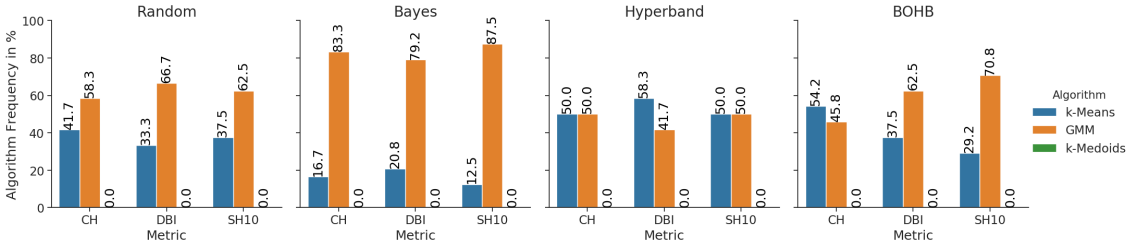
Figure 7.7: Median runtime results for (a) the HPO experiment and (b) the CASH experiment in seconds for only using warmstart configurations and for using warmstarts with additional optimizer loops.

execute. The k range is relatively small, i.e., $R = \{2, \ldots, 200\}$. Therefore, it is very likely that it does not have such a great influence in the performed experiences. But when considering larger k ranges this must be considered.

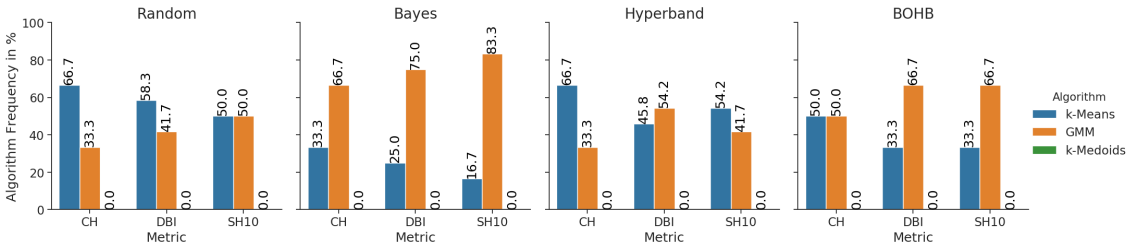
Figure 7.7 does not show the runtime for the overhead of extracting the meta-features in the online phase. The **time for extracting the meta-features is in median 22.63 seconds** and is independent of the (a) optimizer, (b) metric, (c) number of warm start configurations (l_w) and (d) the number of additional optimizer loops (l_a). It is only dependent on the characteristics of the online dataset. However, the produced overhead of the meta-feature extraction is especially for the Random and Bayes optimizer very high when comparing to solely the runtime of them. But with increasing number of loops, the relative overhead of the meta-feature extraction becomes more and more negligible.

The runtime for the CASH experiment is always lower than for the HPO experiment, independently of the optimizer and metric combination. This is already discussed in the evaluation of the offline phase (see Section 7.2) and for coldstart (see Section 7.3.1). It is due to the lower runtime that is observed for GMM and k-Medoids than for k-Means.

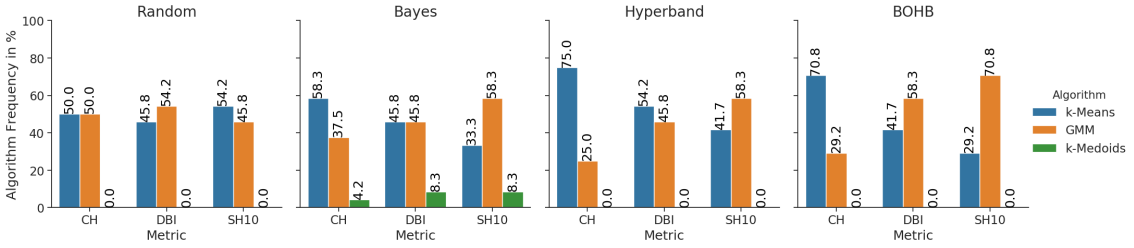
Concluding, the warmstart with meta-learning adds additional overhead because of the meta-feature extraction, but even without this overhead the runtime for warmstart is not significant different than for warmstart. However, as shown for the accuracy, the additional runtime overhead can lead to a higher accuracy. In addition, when examining more optimizer loops the overhead becomes negligible since this is only dependent on the dataset characteristics. Moreover, in this work, all meta-features that do not use any kind of ground truth labels from Rivolli et al. are used [RGS+18]. But other works showed that using only distance-based meta-features can improve the accuracy [GN15]. Hence, it must be analyzed if less, respectively more specific meta-features can improve the accuracy while also reducing the runtime overhead.



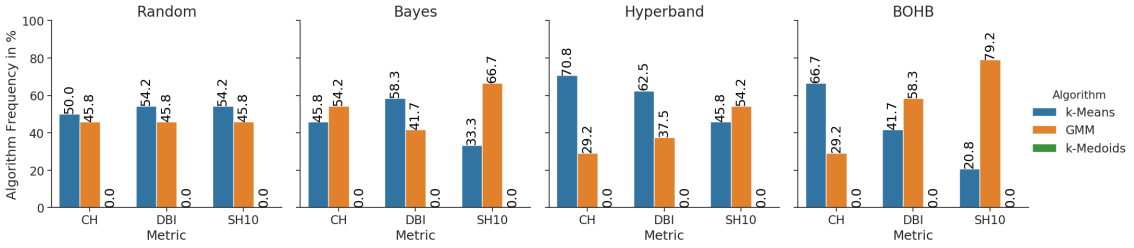
(a) $l_a = 0$ and $l_w = 4$.



(b) $l_a = 0$ and $l_w = 8$.



(c) $l_a = l_w = 4$.



(d) $l_a = l_w = 8$.

Figure 7.8: Algorithm frequencies in percent for the HPO and the CASH experiment for only using warmstart configurations with (a) $l_w = 4$ and (b) $l_w = 8$ and with additional optimizer loops for (c) $l_a = l_w = 4$ and (d) $l_a = l_w = 8$.

Frequency of best-performing Algorithms for CASH

Figure 7.8 shows the frequencies of the best-performing clustering algorithms for the CASH experiment. When only using warmstarts, the GMM algorithm is chosen most frequently in the best configuration. This is due to the results from the external metrics from the offline phase. Since warmstarting uses the metric with the lowest Δk for the most similar dataset, this can also be an external metric. In fact, it is observed that it this is most often an external metric. As discussed for the offline phase, the external metrics choose GMM most frequently and because of this, the warmstart configurations also contain the GMM algorithm most frequently. However,

when using additional optimizer loops ($l_a = 4, 8$), the metrics CH and DBI are choosing k-Means more frequently. The reason is that that these metrics preferring clustering results that reduce the compactness and increase the separation, which is the same objective as for k-Means. The SH10 metric still chooses GMM more frequently than k-Means, even with additional optimizer loops. The reason is that the SH10 metric considers the silhouette of each data point.

It can also be observed that the k-Medoids algorithm is almost never chosen except for the Bayes optimizer with $l_a = l_w = 4$. Since the best configurations from the offline phase also provide k-Medoids very rarely as best configuration (see Figure 7.2), this also applies for the warmstart configurations. However, even with additional optimizer loops, the k-Medoids is chosen is almost never chosen. The reason is that the k-Medoids minimizes the distances from each data point to the medoid point of the affiliated cluster, but the metrics are choosing results that minimize the distance to the synthetically center.

7.3.3 Disscusion of the Results

In the previous (sub-)sections, the results of the online phase for cold- and warmstarting were shown for various optimizers and metric combinations. In the following, it is discussed which optimizers and metric combinations should be used for certain requirements. Also, it is analyzed if initialising the initial configurations with coldstart or warmstart is more beneficial. However, there is no general answer to these questions and it is examined which of them is most suited for certain use cases.

Optimizer

The first considerations for the use of an optimizer is the size of the configuration space and the budget that can be used. If the configuration space is not very large, Random can be a valid choice due to its low runtimes. The same applies if the budget is small. At least for a small number of optimizer loops for coldstart, the Random optimizer performs better than Bayes in terms of accuracy and has a much lower runtime than Hyperband and BOHB. However, in terms of accuracy, Hyperband and BOHB outperform Random. This means that if the accuracy is more important than the runtime, Hyperband and BOHB are more suitable. In addition, if resources are available to parallelize them, the runtime can be drastically reduced. If there are no resources to parallelize the optimizer, at least a medium or large budget and configuration space are used and the runtime and accuracy are both equal important, then the Bayes optimizer would be the best choice.

Metrics

The experiments showed that the best performance regarding accuracy can be achieved with the DBI metric. It achieved independent of the optimizer the highest accuracy, especially because it is more robust [LLX+13]. However, it did not have the lowest runtime always, but the runtime hardly differed in comparison to the other metrics. Because of this, from the investigated metrics, the DBI metric is the best choice.

Coldstart or Warmstart

The results showed that accurate results are possible with warmstart, even with just 4 warmstart configurations. The runtime hardly differs for cold- and warmstart, but the overhead of the meta-feature extraction must be considered for warmstart. However, the more budget is used, the more the overhead for extracting meta-featues becomes negligible. Another aspect for warmstarting is the required offline phase. To this end, evaluations for configurations on offline datasets are required. It is not necessary to run multiple metrics here, but one is sufficient. This can also be an internal metric, at best the DBI metric, such that the offline data does not require the actual number of clusters. Although in the experiments all metrics, in particular external metrics, are used, the DBI metric achieved high accuracy in the offline phase, which makes it also suitable. However, it has also to be taken into account that the offline phase is a time-consuming procedure that is only worth if the results are used on several online datasets.

7.4 Comparison to existing Estimation Methods

In the previous sections, the accuracy and runtimes that can be achieved with the prototypical implementation of the concepts were shown. In this section, the accuracy and runtime are compared to state-of-the-art methods for estimating hyperparameters for clustering. Since this work focuses on partitional clustering algorithms, the comparison is performed with two methods that estimate the hyperparameter k for k-Means. To this end, the G-Means [HHE03] and X-Means [PM00] algorithms are used. Both are executed three times and the median results are examined. The basic procedure of these methods is to start with an initial number of clusters k_{start}. Then for each cluster they perform the following procedure:

1. Split each current cluster into two clusters by running k-Means with $k = 2$ on the data of each cluster.
2. For each split, check if the split should be accepted or not.
3. If at least one split is accepted, repeat from step 1.

The difference between the two methods is step 2, i.e., the criterion for accepting or regretting a split. For G-Means, the Anderson-Darling test is used and for X-Means, the Bayesian Information Criterion (BIC) [Sch78] is applied. While X-Means uses the split clusters after each iteration to proceed with the next iteration, the G-Means method runs k-Means with the current number of clusters on the whole dataset after all splits are performed.

In the experiments, the initial number of clusters for both algorithms is set to 2 and an additional stop criterion is added, which stops the execution when a maximum of 200 clusters are discovered. The X-Means algorithm is implemented with the pyclustering library [Nov19] and the G-Means algorithm with an implementation from Github[3]. The results of them are compared against the results of the HPO experiment with coldstart and warmstart and also all optimizer and metric

[3] https://github.com/flylo/g-means

		Optimizer				Estimation Methods	
	Metric	Random	Bayes	Hyperband	BOHB	X-Means	G-Means
Coldstart	CH	20.5	62	6.5	5	65	72.5
($l_{total} = 4$)	DBI	26	79	5	5.5		
	SH10	65	115.5	76	71.5		
Warmstart	CH	9	11	5	6.5		
($l_w = 4$, $l_a = 0$)	DBI	9	11	3.5	4		
	SH10	12	16	77	70		

Table 7.6: Comparison of the accuracy results in terms of Δk for X-Means and G-Means with coldstart and warmstart of the optimizers and each metric.

combinations. For the budget of the optimizers, the lowest budget from the previous experiments is used, which is $l_{total} = 4$ for coldstart and $l_w = 4$, respectively $l_a = 0$ for warmstart. The comparison is done regarding the accuracy, in terms of Δk, and the median runtime.

7.4.1 Accuracy

The results of the accuracy in terms of Δk for both methods are shown in Table 7.6. There, X-Means and G-Means have very high Δk values. The results of the optimizers for cold- and warmstart have a significant better accuracy. Neglecting the SH10 metric all combinations of optimizers and metric achieve better results than X-Means and G-Means. This means that each optimizer performs better than these two estimation methods regarding the accuracy. A reason for the bad accuracy for X-Means and G-Means is the noise in the datasets. It is observed that X-Means achieves $\Delta k = 0$ if only datasets without noise are used. However, this shows that the optimizers and the metrics are more robust to the presence of noise than the estimation methods. Also, G-Means still has $\Delta k = 48.5$ even when only datasets without noise are used. Moreover, the lowest number of optimizer loops is considered, which mans the optimizers are even more accurate in comparison to X-Means and G-Means when more loops are considered.

7.4.2 Runtime

The median runtime results for coldstart and warmstart of the optimizers as well as X-Means and G-Means are shown in Table 7.7. There, it can be seen that the X-Means and G-Means have a much lower runtime than Hyperband and BOHB. Especially the X-Means method has the lowest runtime in comparison to all optimizers. The reason is that the estimation methods execute the k-Means algorithm not always on the whole dataset but only on subsets of the data. This is much less effort than executing and evaluating configurations on the whole dataset. The higher runtime of G-Means is due to the fact that after each iteration when the clusters are split, k-Means is performed with the current number of clusters. Because of this, the warmstart results for Random have a lower runtime than G-Means. The runtime of Bayes is higher than for G-Means since the optimizer produces additional overhead by updating the underlying model after executing each configuration.

		Optimizer				Estimation Methods	
	Metric	Random	Bayes	Hyperband	BOHB	X-Means	G-Means
Coldstart	CH	4.5	5.3	40.1	32.4	0.8	5.4
($l_{total} = 4$)	DBI	5.8	6.2	44.1	31.3		
	SH10	6.3	6.5	40.5	40.6		
Warmstart	CH	2.4	6.4	29.8	34.2		
($l_w = 4$, $l_a = 0$)	DBI	2.5	6.6	27.0	35.8		
	SH10	2.5	6.3	36.1	37.5		

Table 7.7: Median runtime results in seconds for all optimizers and the two estimation methods.

However, the runtimes are shown for only 4 optimizer loops. When using more optimizer loops, the runtimes of X-Means and G-Means are much lower than for the optimizers.

Concluding, it is shown that X-Means and G-Means have a significant lower runtime. In addition, if no noise is used in the datasets, X-Means achieved $\Delta k = 0$. However, most of the real-world datasets contain noise and the results of the optimizers are significantly more accurate on all datasets, in particular on datasets with noise. The higher accuracy is also achieved with cold- and warmstarting and even when using only 4 optimizer loops in total.

8 Conclusion

Because of the huge amount of data in the Big Data Era, it becomes crucial to perform data analyses to gain meaningful insights out of the sheer size of data. The analyses typically include machine learning methods to extract patterns out of the data. Due to the various machine learning algorithms and their hyperparameters, choosing a solid configuration is a highly exploratory task. However, the current research in the area of AutoML resulted in systems that automatically select an algorithm with appropriate hyperparameters. Their objective is, given a budget as input, select and execute different configurations and return the configuration with the best performance when the budget is exhausted. For this, they use optimizers that select the configurations in such a way that they can approximate an optimum in a reasonable time. Yet, these systems are only applied for supervised learning tasks. Because of missing ground truth labels, applying these concepts to unsupervised learning tasks is more challenging.

In this work, a concept was developed that transfers these concepts to the unsupervised learning problem of clustering. To this end, the components of the existing AutoML systems were examined. Subsequently, a concept which transfers these concepts was proposed. The concept relies on the same components as existing AutoML systems, but the components are modified in a way that they are applicable for clustering. This concept involves two phases; an offline and an online phase. The offline phase is used to apply meta-learning for warmstarting the optimizers. For this, various configurations are executed and evaluated on offline datasets. In the online phase, the goal is to find a suitable configuration with clustering algorithms for an unseen dataset. To this end, the concept provides two methods, which are coldstarting, which means that the optimizer selects the first configurations at randomly, and warmstarting, which means that the initial configurations are selected via meta-learning. The meta-learning method uses the information from the offline phase to select configurations that performed well on previously already seen datasets.

The concept was also prototypically implemented. The implementation was used for a comprehensive evaluation. To this end, 4 optimizers, 9 metrics and 3 partitional clustering algorithms were used throughout the evaluation. The experiments unveiled that the more budget the optimizers have at their disposal, the more precise are the results. Yet, precise results were attained with very low budgets for the optimizers. With an increasing budget, more precise results were achieved, yet with the cost of a higher runtime. The results for warmstarting showed that the performance of optimizers could be increased even further. Nevertheless, when using warmstarting the effort for the offline phase as well as the overhead for extracting the meta-features must be considered. However, the more budget is used the more negligible is the overhead for meta-feature extraction, as it is only performed once.

The best results regarding the accuracy and runtime were obtained when the Davies-Bouldin Index (DBI) was used. A reason for this is that several datasets with noise were used and DBI is more robust against outliers in contrast to the other investigated internal metrics. For the optimizers, there is no general conclusion as the choice of the specific optimizer depends on the requirements. For

example, Hyperband and BOHB attained always the highest accuracy but also the highest runtime, in fact more than four times higher than for the other optimizers. On the other hand, the Bayes optimizer had a considerable lower runtime, while still obtaining precise results, especially for warmstarting.

The proposed concept was also compared against two estimation methods for estimating the number of clusters for k-Means. The results unveiled that that the proposed concept achieves a higher accuracy than the estimation methods, yet with a higher runtime. Especially for datasets with noise, the transferred AutoML concept for clustering analyses outperforms existing estimation methods for the number of clusters in datasets.

9 Outlook

This work presented a concept for transferring AutoML concepts for the unsupervised learning problems of clustering analyses. However, since there is no work that also applies meta-learning and the optimizers for the combined algorithm selection and hyperparameter optimization problem, there are still some aspects to be investigated in future work.

The first aspect are the inputs for the optimizers. These are a dataset, a metric and a budget that can be used by the optimizer. Since the optimizers evaluate configurations with the given metric, the choice of a metric is crucial for the overall result of the optimizers. Moreover, as the evaluation unveiled, certain metrics are better suited for noise in the dataset. Considering the optimizers, it must be examined to which extend parallelism can reduce the runtime. In particular, for Hyperband and BOHB, which had significant higher runtimes than the other optimizers, an improvement is expected [FKH18]. Moreover, the optimizers have additional parameters, e.g., the acquisition function for Bayes, which were not further investigated in this work but can have a major impact on the performance of the optimizer. For each optimizer the budget must be defined. Yet, it can be analyzed if other types of budgets are suited as well. For this, it can also be investigated what a suitable amount of budget is, in particular, when do the optimizers find the best configuration or when is the improvement in accuracy of the configurations negligible.

Strongly related to the budget is the configuration space. With an increasing configuration space, it becomes more challenging to find a solid configuration. In this context, an approach that involves a human expert would also be possible. In this setting, the human expert could add his domain knowledge, e.g., in form of constraints or conditions, that shrink the configuration space. In addition, it must be determined if it is beneficial to use AutoML systems for human-in-the-loop data analytics. For this, a human expert could use the AutoML system in an exploratory manner, i.e., the expert starts with a small budget and depending on the resuts, refines the budget, the configuration space and maybe even the optimization metric.

Additionally, it can be examined if the information from an optimizer loop can be utilized for subsequent optimizer loops. Although certain optimizers, e.g., Bayes and BOHB, use the information of evaluated configurations to select the next configuration, they do not apply information from past configurations on newly selected configurations. Fritz and Schwarz reuse the initial cluster centers of configurations for subsequent configurations [FS19]. They showed that their *Delta Initialization* method has significant runtime savings compared to state-of-the-art methods for initializing the cluster centers. Therefore, it could be investigated if it is beneficial to use this method for determining the number of clusters for k-Means in combination with the optimizers.

This work focuses on partitional clustering algorithms, hence, it can be examined if the concept can also be applied on other families of clustering algorithms as well. However, this introduces several challenges. The first is that a budget Must be defined that approximates the clustering results. To this end, the number of iterations of the clustering algorithms was used in this work. This is possible since the partitional clustering algorithms are proceeding in an iterative manner, i.e., they compute

in each iteration a clustering result that contains for each data point a clustering label. Nevertheless, the DBSCAN [EKSX96] algorithm, for example, does not proceed in an iterative manner, but yields one cluster after another. In this case, other kinds of budgets like samples of the dataset could be used for the approximation, but choosing these samples is another challenge. Moreover, the experiments showed that two of the three investigated metrics favour k-Means because they aim for the same objective. Due to this, it could be examined which metric is suitable when different families of clustering algorithms are considered.

Another aspect that affects the performance are the initial configurations. The evaluation of this work unveiled that the performance can be improved when using meta-learning to warmstart an optimizer. However, how well the meta-learning procedure performs depends on the number of evaluated configurations, the characteristics of the offline datasets and the extracted meta-features. For the latter one, it could be examined which sets of meta-features is most suited. This means that the overhead for extracting the meta-features is tolerated while still achieving resilient results. In the literature, works exist that investigate the use of different meta-feature sets for clustering [FC12; GN15; PC19], but they only focus on the problem of algorithm selection. They do not take the combined problem of algorithm selection and hyperparameter optimization into account. Also, they do not use the optimizers that are used in existing AutoML systems.

Last but not least, existing AutoML systems also use other concepts that are not incorporated in this work. One of them is the use of ensemble building methods [VM02]. Auto-sklearn uses this method to improve the performance by combining the best performing configurations [FKE+15]. There are methods that use ensemble methods for clustering, typically called consensus clustering [VR11]. Also, there are works that use these ensembles to evaluate clustering algorithms and their according hyperparameters [HLMB18]. Therefore, it could be examined if these methods can be included in AutoML systems for clustering.

Bibliography

[AV07] D. Arthur, 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 Philadelphia, 2007, pp. 1027–1035 (cit. on pp. 28, 50).

[Bay12] M. Bayer. “SQLAlchemy”. In: *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. Ed. by A. Brown, G. Wilson. aosabook.org, 2012. URL: http://aosabook.org/en/sqlalchemy.html (cit. on p. 53).

[BB12] J. Bergstra, Y. Bengio. “Random Search for Hyper-parameter Optimization”. In: *Journal of Machine Learning Research* 13.1 (2012), pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2503308.2188395 (cit. on p. 24).

[BBB19] A. C. Benabdellah, A. Benghabrit, I. Bouhaddou. “A survey of clustering algorithms for an industrial context”. In: *Procedia Computer Science* 148 (Jan. 2019), pp. 291–302. ISSN: 1877-0509. DOI: 10.1016/J.PROCS.2019.01.022. URL: https://www.sciencedirect.com/science/article/pii/S1877050919300225 (cit. on p. 30).

[BBBK11] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl. “Algorithms for hyper-parameter optimization”. In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*. 2011. ISBN: 9781618395993 (cit. on pp. 24, 33, 69).

[BCD10] E. Brochu, V. M. Cora, N. De Freitas. “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning”. In: *arXiv preprint arXiv:1012.2599* (2010) (cit. on pp. 25, 41).

[BGSV08] P. Brazdil, C. Giraud-Carrier, C. Soares, R. Vilalta. *Metalearning: Applications to data mining*. 1st ed. Springer Publishing Company, Incorporated, 2008. ISBN: 9783642092312 (cit. on pp. 31, 47).

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN: 0387310738 (cit. on pp. 27, 29, 41, 48).

[BMV+12] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii. “Scalable κ-means++”. In: *Proceedings of the VLDB Endowment* 5.7 (2012), pp. 622–633. ISSN: 21508097. DOI: 10.14778/2180912.2180915 (cit. on p. 28).

[CH74] T. Caliński, J. Harabasz. “A Dendrite Method For Cluster Analysis”. In: *Communications in Statistics* 3.1 (1974), pp. 1–27. ISSN: 00903272. DOI: 10.1080/03610927408827101 (cit. on pp. 31, 59).

[DB79] D. L. Davies, D. W. Bouldin. “A Cluster Separation Measure”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (1979), pp. 224–227. ISSN: 01628828. DOI: 10.1109/TPAMI.1979.4766909 (cit. on pp. 28, 31, 36, 59).

[DLR77] A. P. Dempster, N. M. Laird, D. B. Rubin. “ Maximum Likelihood from Incomplete Data Via the EM Algorithm ”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* (1977). ISSN: 0035-9246. DOI: 10.1111/j.2517-6161.1977.tb01600.x (cit. on p. 29).

[DPS+08] M. C. De Souto, R. B. Prudêncio, R. G. Soares, D. S. De Araujo, I. G. Costa, T. B. Ludermir, A. Schliep. “Ranking and selecting clustering algorithms using a meta-learning approach”. In: *Proceedings of the International Joint Conference on Neural Networks*. 2008, pp. 3729–3735. ISBN: 9781424418213. DOI: 10.1109/IJCNN.2008.4634333 (cit. on pp. 34, 35).

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*. 1996 (cit. on pp. 23, 84).

[EMS19] R. Elshawi, M. Maher, S. Sakr. *Automated Machine Learning: State-of-The-Art and Open Challenges*. 2019 (cit. on pp. 20, 34).

[FAT+14] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, A. Bouras. “A survey of clustering algorithms for big data: Taxonomy and empirical analysis”. In: *IEEE Transactions on Emerging Topics in Computing* 2.3 (Sept. 2014), pp. 267–279. ISSN: 21686750. DOI: 10.1109/TETC.2014.2330519 (cit. on p. 28).

[FBS19] M. Fritz, M. Behringer, ·. H. Schwarz. “Quality-driven early stopping for explorative cluster analysis for big data”. In: *SICS Software-Intensive Cyber-Physical Systems* 34 (2019), pp. 129–140. DOI: 10.1007/s00450-019-00401-0. URL: https://doi.org/10.1007/s00450-019-00401-0 (cit. on p. 59).

[FC12] D. G. Ferrari, L. N. de Castro. “Clustering algorithm recommendation: a meta-learning approach”. In: *International Conference on Swarm, Evolutionary, and Memetic Computing*. Springer. 2012, pp. 143–150 (cit. on pp. 34, 84).

[FEF+18] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, F. Hutter. “Practical automated machine learning for the automl challenge 2018”. In: *International Workshop on Automatic Machine Learning at ICML*. 2018, pp. 1189–1232 (cit. on pp. 20, 41, 51).

[FKE+15] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter. “Efficient and robust automated machine learning”. In: *Advances in neural information processing systems*. 2015, pp. 2962–2970 (cit. on pp. 20, 24, 25, 33, 39–42, 47, 48, 63, 84).

[FKH18] S. Falkner, A. Klein, F. Hutter. “BOHB: Robust and Efficient Hyperparameter Optimization at Scale”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy, A. Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1437–1446. URL: http://proceedings.mlr.press/v80/falkner18a.html (cit. on pp. 24, 27, 34, 51, 83).

[FM83] E. B. Fowlkes, C. L. Mallows. “A method for comparing two hierarchical clusterings”. In: *Journal of the American Statistical Association* (1983). ISSN: 1537274X. DOI: 10.1080/01621459.1983.10478008 (cit. on pp. 31, 59).

[FPS96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth. "Advances in Knowledge Discovery and Data Mining". In: ed. by U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996. Chap. From Data, pp. 1–34. ISBN: 0-262-56097-6. URL: http://dl.acm.org/citation.cfm?id=257938.257942 (cit. on p. 19).

[Fra18] P. I. Frazier. *A Tutorial on Bayesian Optimization*. Tech. rep. 2018. URL: https://arxiv.org/pdf/1807.02811.pdf (cit. on p. 25).

[FS19] M. Fritz, H. Schwarz. "Initializing k-Means Efficiently: Benefits for Exploratory Cluster Analysis". In: *On the Move to Meaningful Internet Systems: OTM 2019 Conferences*. Ed. by H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, R. Meersman. Cham: Springer International Publishing, 2019, pp. 146–163. ISBN: 978-3-030-33246-4 (cit. on pp. 28, 58, 83).

[FSH15] M. Feurer, J. T. Springenberg, F. Hutter. "Initializing Bayesian Hyperparameter Optimization via Meta-Learning". In: *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)* (2015), pp. 1128–1135. DOI: 10.1002/9781444307399 (cit. on pp. 40, 69).

[GBC+15] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, Tin Kam Ho, N. Macia, B. Ray, M. Saeed, A. Statnikov, E. Viegas. "Design of the 2015 ChaLearn AutoML challenge". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2015, pp. 1–8. ISBN: 978-1-4799-1960-4. DOI: 10.1109/IJCNN.2015.7280767. URL: http://ieeexplore.ieee.org/document/7280767/ (cit. on p. 33).

[GKR91] J. E. Gentle, L. Kaufman, P. J. Rousseuw. "Finding Groups in Data: An Introduction to Cluster Analysis." In: *Biometrics* (1991). ISSN: 0006341X. DOI: 10.2307/2532178 (cit. on p. 28).

[GN15] D. Gomes Ferrari, L. Nunes de Castro. "Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods". In: *Information Sciences* 301 (2015), pp. 181–194. DOI: 10.1016/j.ins.2014.12.044. URL: http://dx.doi.org/10.1016/j.ins.2014.12.044 (cit. on pp. 34, 35, 74, 84).

[GSB+19] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W.-W. Tu, E. Viegas. "Analysis of the AutoML Challenge Series 2015–2018". In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, J. Vanschoren. Cham: Springer International Publishing, 2019, pp. 177–219. ISBN: 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5{_}10. URL: https://doi.org/10.1007/978-3-030-05318-5_10 (cit. on p. 34).

[HA85] L. Hubert, P. Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (Dec. 1985), pp. 193–218. ISSN: 01764268. DOI: 10.1007/BF01908075 (cit. on pp. 31, 59).

[HBV01] M. Halkidi, Y. Batistakis, M. Vazirgiannis. "On clustering validation techniques". In: *Journal of Intelligent Information Systems* 17.2-3 (Dec. 2001), pp. 107–145. ISSN: 09259902. DOI: 10.1023/A:1012801612483 (cit. on pp. 30, 36, 45).

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten. "The WEKA data mining software". In: *ACM SIGKDD Explorations Newsletter* (2009). ISSN: 19310145. DOI: 10.1145/1656274.1656278 (cit. on p. 33).

[HHCB09] F. Hutter, H. H. Hoos, K. U. Ca, S. A. Be. *ParamILS: An Automatic Algorithm Configuration Framework*. Tech. rep. 2009, pp. 267–306. URL: http://www.ilog.com/products/cplex/ (cit. on p. 23).

[HHE03] G. Hamerly, G. Hamerly, C. Elkan. "Learning the K in K-Means". In: *In Neural Information Processing Systems* 17 (2003). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.3574 (cit. on pp. 28, 36, 77).

[HHL11] F. Hutter, H. H. Hoos, K. Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *International conference on learning and intelligent optimization*. Springer, Berlin, Heidelberg, 2011. ISBN: 9783642255656. DOI: 10.1007/978-3-642-25566-3{_}40 (cit. on pp. 25, 33).

[HKP12] J. Han, M. Kamber, J. Pei. *Data Mining : Concepts and Techniques (3rd Edition)*. Morgan Kaufmann, 2012. ISBN: 978-0-12-381479-1. DOI: 10.1016/B978-0-12-381479-1.00001-0. URL: http://linkinghub.elsevier.com/retrieve/pii/B9780123814791000010 (cit. on pp. 27, 28, 66).

[HLMB18] L. Helfmann, J. von Lindheim, M. Mollenhauer, R. Banisch. "On Hyperparameter Search in Cluster Ensembles". In: (Mar. 2018). URL: http://arxiv.org/abs/1803.11008 (cit. on p. 84).

[HML+18] T. Head, MechCoder, G. Louppe, I. Shcherbatyi, fcharras, Z. Vinícius, cmmalone, C. Schröder, nel215, N. Campos, T. Young, S. Cereda, T. Fan, rene-rex, K. (Shi, J. Schwabedal, carlosdanielcsantos, Hvass-Labs, M. Pak, SoManyUsernamesTaken, F. Callaway, L. Estève, L. Besson, M. Cherti, K. Pfannschmidt, F. Linzberger, C. Cauet, A. Gut, A. Mueller, A. Fabisch. "Scikit-optimize". In: (Mar. 2018). DOI: 10.5281/ZENODO.1207017. URL: https://zenodo.org/record/1207017 (cit. on p. 51).

[Jai10] A. K. Jain. "Data clustering: 50 years beyond K-means". In: *Pattern recognition letters* 31.8 (2010), pp. 651–666 (cit. on p. 28).

[JT16] K. Jamieson, A. Talwalkar. "Non-stochastic best arm identification and hyperparameter optimization". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*. 2016 (cit. on p. 26).

[KC88] A. K. Jain, R. C. Dubes. *Algorithms for clustering data*. Prentice Hall, 1988, pp. 1–320. ISBN: 0-13-022278-X (cit. on pp. 28, 30, 45).

[KMN+02] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu. "An efficient k-means clustering algorithms: Analysis and implementation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2002). ISSN: 01628828. DOI: 10.1109/TPAMI.2002.1017616 (cit. on p. 28).

[KMP17] A. Kaul, S. Maheshwary, V. Pudi. "AutoLearn — Automated Feature Generation and Selection". In: *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, Nov. 2017, pp. 217–226. ISBN: 978-1-5386-3835-4. DOI: 10.1109/ICDM.2017.31. URL: http://ieeexplore.ieee.org/document/8215494/ (cit. on p. 23).

[KR87] L. Kaufman, P. Rousseeuw. "Clustering by means of Medoids". In: *Statistical Data Analysis Based on the L1–Norm and Related Methods*. Ed. by Y. Dodge. 1987, pp. 405–416 (cit. on pp. 29, 49).

[KRTT15] O. Kettani, F. Ramdani, B. Tadili, B. Tadili. "AK-means: an automatic clustering algorithm based on K-means". In: *Journal of Advanced Computer Science & Technology* 4.2 (June 2015), p. 231. ISSN: 2227-4332. DOI: 10.14419/jacst.v4i2.4749. URL: http://www.sciencepubco.com/index.php/JACST/article/view/4749 (cit. on p. 28).

[KSS17] G. Katz, E. C. R. Shin, D. Song. "ExploreKit: Automatic feature generation and selection". In: *Proceedings - IEEE International Conference on Data Mining, ICDM* (2017), pp. 979–984. ISSN: 15504786. DOI: 10.1109/ICDM.2016.176 (cit. on p. 23).

[KTH+17] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, K. Leyton-Brown. *Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA*. Tech. rep. 2017, pp. 1–5. URL: http://automl.org/autoweka (cit. on p. 33).

[KTSP17] U. Khurana, D. Turaga, H. Samulowitz, S. Parthasrathy. "Cognito: Automated Feature Engineering for Supervised Learning". In: *IEEE International Conference on Data Mining Workshops, ICDMW* (2017), pp. 1304–1307. ISSN: 23759259. DOI: 10.1109/ICDMW.2016.0190 (cit. on p. 23).

[Lan01] D. Laney. "3D Data Management: Controlling Data Volume, Velocity, and Variety." In: *Application Delivery Strategies* (2001). ISSN: 09505849. DOI: 10.1016/j.infsof.2008.09.005. arXiv: 0402594v3 [cond-mat] (cit. on p. 19).

[LJD+17] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar. "Hyperband: a novel bandit-based approach to hyperparameter optimization". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816 (cit. on pp. 24, 26, 34).

[Llo82] S. P. Lloyd. *Least Squares Quantization in PCM*. Tech. rep. 2. 1982 (cit. on p. 28).

[LLX+10] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu. "Understanding of internal clustering validation measures". In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2010, pp. 911–916. ISBN: 9780769542560. DOI: 10.1109/ICDM.2010.35 (cit. on pp. 69, 73).

[LLX+13] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, S. Wu. "Understanding and enhancement of internal clustering validation measures". In: *IEEE Transactions on Cybernetics* 43.3 (June 2013), pp. 982–994. ISSN: 21682267. DOI: 10.1109/TSMCB.2012.2220543 (cit. on pp. 69, 73, 76).

[Mac67] J. MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*. 1967 (cit. on pp. 28, 49).

[Nov19] A. Novikov. "PyClustering: Data Mining Library". In: *Journal of Open Source Software* 4.36 (Apr. 2019), p. 1230. ISSN: 2475-9066. DOI: 10.21105/joss.01230 (cit. on p. 77).

[NPDC09] A. C. Nascimento, R. B. Prudêncio, M. C. De Souto, I. G. Costa. "Mining rules for the automatic selection process of clustering methods applied to cancer gene expression data". In: *Artificial Neural Networks – ICANN 2009*. Vol. 5769 LNCS. PART 2. Springer Berlin Heidelberg, 2009, pp. 20–29. ISBN: 3642042767. DOI: 10.1007/978-3-642-04277-5{_}3 (cit. on pp. 34, 35).

[PC19] B. A. Pimentel, A. C. de Carvalho. "A new data characterization for selecting clustering algorithms using meta-learning". In: *Information Sciences* 477 (Mar. 2019), pp. 203–219. ISSN: 0020-0255. DOI: 10.1016/J.INS.2018.10.043. URL: https://www.sciencedirect.com/science/article/pii/S0020025518308624 (cit. on pp. 34, 36, 84).

[PJ09] H. S. Park, C. H. Jun. "A simple and fast algorithm for K-medoids clustering". In: *Expert Systems with Applications* (2009). ISSN: 09574174. DOI: 10.1016/j.eswa.2008.01.039 (cit. on p. 50).

[PM00] D. Pelleg, A. W. Moore. "X-means: Extending K-means with Efficient Estimation of the Number of Clusters". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 727–734. ISBN: 1-55860-707-2. URL: http://dl.acm.org/citation.cfm?id=645529.657808 (cit. on pp. 28, 36, 77).

[PVG+11] 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, E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 33, 50).

[QHJQ06] W. Qiu Brigham, B. Harry Joe, H. Joe, W. Qiu. "Generation of Random Clusters with Specified Degree of Separation". In: *Journal of Classification* 23 (2006), pp. 315–334. DOI: 10.1007/s00357-006-0018-y (cit. on p. 37).

[QMH+18] Y. Quanming, W. Mengshuo, J. E. Hugo, G. Isabelle, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, Y. Yang. "Taking human out of learning applications: A survey on automated machine learning". In: *arXiv preprint arXiv:1810.13306* (2018) (cit. on p. 34).

[Ras04] C. E. Rasmussen. "Gaussian Processes in machine learning". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3176 (2004), pp. 63–71. ISSN: 03029743. DOI: 10.1007/978-3-540-28650-9{_}4 (cit. on pp. 25, 29, 49).

[RGS+18] A. Rivolli, L. F. Garcia, C. Soares, J. Vanschoren, A. C. de Carvalho. "Towards reproducible empirical research in meta-learning". In: *arXiv preprint arXiv:1808.10406* (2018) (cit. on pp. 32, 52, 74).

[RH07] A. Rosenberg, J. Hirschberg. "V-Measure: A conditional entropy-based external cluster evaluation measure". In: *EMNLP-CoNLL 2007 - Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 2007 (cit. on pp. 31, 59).

[Rou87] P. J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20.C (1987), pp. 53–65. ISSN: 03770427. DOI: 10.1016/0377-0427(87)90125-7 (cit. on pp. 28, 31, 36, 59).

[SC19] J. A. Sáez, E. Corchado. "A Meta-Learning Recommendation System for Characterizing Unsupervised Problems: On Using Quality Indices to Describe Data Conformations". In: *IEEE Access* 7 (2019), pp. 63247–63263. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2917004 (cit. on pp. 34, 36).

[Sch78] G. Schwarz. “Estimating the Dimension of a Model”. In: *The Annals of Statistics* 6.2 (Mar. 1978), pp. 461–464. ISSN: 0090-5364. DOI: 10.1214/aos/1176344136 (cit. on p. 77).

[SJ03] C. A. Sugar, G. M. James. “Finding the Number of Clusters in a Dataset: An Information-Theoretic Approach”. In: *Journal of the American Statistical Association* (2003). ISSN: 01621459. DOI: 10.1198/016214503000000666 (cit. on pp. 28, 36).

[SKK00] M. Steinbach, G. Karypis, V. Kumar. “A Comparison of Document Clustering Techniques”. In: *KDD workshop on text mining* (2000). ISSN: 978-1-4244-2874-8. DOI: 10.1109/ICCCYB.2008.4721382 (cit. on p. 28).

[SLA12] J. Snoek, H. Larochelle, R. P. Adams. “Practical bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems*. 2012, pp. 2951–2959 (cit. on p. 25).

[SLD09] R. G. F. Soares, T. B. Ludermir, F. A. T. De Carvalho. “An analysis of meta-learning techniques for ranking clustering algorithms applied to artificial data”. In: *International Conference on Artificial Neural Networks*. Springer. 2009, pp. 131–140 (cit. on pp. 34, 35).

[SSW+16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, N. De Freitas. “Taking the human out of the loop: A review of Bayesian optimization”. In: *Proceedings of the IEEE*. Vol. 104. 1. 2016, pp. 148–175. ISBN: 0018-9219. DOI: 10.1109/JPROC.2015.2494218 (cit. on p. 25).

[THHL13] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown. “Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’13*. New York, New York, USA: ACM Press, 2013, p. 847. ISBN: 9781450321747. DOI: 10.1145/2487575.2487629. URL: http://dl.acm.org/citation.cfm?doid=2487575.2487629 (cit. on pp. 20, 23, 25, 33, 39, 40, 46, 48, 63).

[Tho53] R. L. Thorndike. “Who belongs in the family?” In: *Psychometrika* 18.4 (Dec. 1953), pp. 267–276. ISSN: 00333123. DOI: 10.1007/BF02289263 (cit. on p. 36).

[VD95] G. Van Rossum, F. L. Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995 (cit. on p. 49).

[VEB10] N. X. Vinh, J. Epps, J. Bailey. “Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance”. In: *Journal of Machine Learning Research* (2010). ISSN: 15324435 (cit. on pp. 31, 59).

[VM02] G. Valentini, F. Masulli. “Ensembles of learning machines”. In: *Italian workshop on neural nets*. Vol. 2486 LNCS. Springer, Berlin, Heidelberg, 2002, pp. 3–20. DOI: 10.1016/b978-0-12-417295-1.00020-5. URL: http://link.springer.com/10.1007/3-540-45808-5_1 (cit. on p. 84).

[VR11] S. Vega-Pons, J. Ruiz-Shulcloper. “A survey of clustering ensemble algorithms”. In: *International Journal of Pattern Recognition and Artificial Intelligence* 25.3 (May 2011), pp. 337–372. ISSN: 02180014. DOI: 10.1142/S0218001411008683 (cit. on p. 84).

[WKR+08] X. Wu, V. Kumar, Q. J. Ross, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. H. Zhou, M. Steinbach, D. J. Hand, D. Steinberg. "Top 10 algorithms in data mining". In: *Knowledge and Information Systems* 14.1 (Jan. 2008), pp. 1–37. ISSN: 02191377. DOI: 10.1007/s10115-007-0114-2 (cit. on p. 28).

[XLD+09] X. Xie, K.-M. Lam, Q. Dai, X. Peng, J. Yang, J. Yang, X. Geng, K. Smith-Miles, S. Choi, S. C. Dass, S. Pankanti, S. Prabhakar, Y. Zhu, K. Uchida, P. Grother, R. Rakvic, R. Broussard, L. Kennell, R. Ives, R. Bell, D. L. Woodard, K. Ricanek, J. R. Matey, N. Bartlow, N. Kalka, B. Cukic, A. Ross, J. Daugman, J. L. Cambier, N. A. Schmid, J. R. Matey, Y.-H. Li, M. Savvides, J. Daugman, J. Daugman, C. Downing, Y.-h. Li, M. Savvides, J. Thornton, B. V. K. Vijaya Kumar, N. A. Schmid, S. W. Park, M. Savvides, D. Harrington, R. Triplett, Y.-H. Li, M. Savvides, G. V. Dozier, M. Savvides, K. Bryant, T. Munemoto, K. Ricanek, D. Woodard, P. Campisi, E. Maiorana, A. Neri. "Incremental Learning". In: *Encyclopedia of Biometrics*. Springer US, 2009, pp. 731–735. DOI: 10.1007/978-0-387-73003-5{_}304 (cit. on p. 44).

[XT15] D. Xu, Y. Tian. "A Comprehensive Survey of Clustering Algorithms". In: *Annals of Data Science* (2015). ISSN: 2198-5804. DOI: 10.1007/s40745-015-0040-1 (cit. on p. 28).

[XW05] R. Xu, D. Wunsch. *Survey of clustering algorithms*. 2005. DOI: 10.1109/TNN.2005.845141 (cit. on pp. 28, 66).

[ZH19] M.-A. Zöller, M. F. Huber. "Survey on Automated Machine Learning". In: (Apr. 2019). URL: http://arxiv.org/abs/1904.12054 (cit. on p. 34).

All links were last followed on March 17, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature