Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

# Analysis and Integration of Data Preprocessing Steps in AutoML for Clustering

Connor Reed

**Course of Study:** Data Science

**Examiner:** Prof Dr.-Ing Bernhard Mitschang

**Supervisor:** Dennis Tschechlov, M.Sc.

**Commenced:** January 4, 2023

**Completed:** April 25, 2023

## Abstract

This work explores the field of Automated Machine Learning (AutoML) and its application in unsupervised learning, specifically clustering. While AutoML for clustering systems exist, none of these fully integrate and analyze preprocessing steps.
Therefore the contributions of this work include the following: Analysis and selection of promising preprocessing methods. The selection is based on supervised AutoML systems with preprocessing and clustering use-cases. Extension of the existing package AutoML4Clust with the selected preprocessing methods and their respective hyperparameters. Addition of categorical clustering algorithms to the search space in an attempt to improve results on data with categorical features. Implementation of multiobjective optimization strategies which aim to reach a trade-off between accuracy and runtime. Reduction in runtime with sampling, which optimizes on only a subset of the data. Rigorous evaluation of the new implementations.
The work concludes that AutoML can significantly benefit from the addition of preprocessing and that the proposed methods show promising results for future development.

# Kurzfassung

Diese Arbeit erforscht das Gebiet des automatisierten maschinellen Lernens (AutoML) und seine Anwendung in unüberwachtem Lernen, insbesondere Clustering. Es gibt zwar AutoML für Clustering Systeme, aber keines von ihnen integriert und analysiert Vorverarbeitungsschritte systematisch. Daher umfasst der Beitrag dieser Arbeit Folgendes: Analyse und Auswahl von vielversprechenden Vorverarbeitungsmethoden. Die Auswahl basiert auf überwachten AutoML-Systemen mit Preprocessing und auf Clustering Anwendungsfällen. Erweiterung des bestehenden Pakets AutoML4Clust um die ausgewählten Preprocessing-Methoden und ihre jeweiligen Hyperparameter. Hinzufügen von kategorischen Clustering-Algorithmen zum Suchraum, um die Ergebnisse bei Daten mit kategorischen Merkmalen zu verbessern. Implementierung von Multiobjective Optimierungsstrategien, die einen Kompromiss zwischen Genauigkeit und Laufzeit anstreben. Reduzierung der Laufzeit durch Sampling, bei dem nur auf einer Teilmenge der Daten optimiert wird. Rigorose Evaluierung der neuen Implementierungen.
Die Arbeit kommt zu dem Schluss, dass AutoML von der Hinzufügung von Vorverarbeitungsmethoden stark profitieren kann und dass die vorgeschlagenen Methoden vielversprechende Ergebnisse für die zukünftige Entwicklung zeigen.

# Contents

**1 Introduction** **15**

**2 Background** **17**
2.1 AutoML . . . . . . 17
2.2 Clustering . . . . . . 20
2.3 Cluster Validation Measures . . . . . . 23

**3 Related Work** **25**
3.1 Supervised AutoML systems with preprocessing . . . . . . 25
3.2 AutoML systems with Clustering . . . . . . 27
3.3 Clustering with Preprocessing . . . . . . 28
3.4 Summary of Related Work . . . . . . 28

**4 Transfer of Preprocessing to Clustering** **31**
4.1 Preprocessing Methods for Clustering . . . . . . 31
4.2 Selection of Preprocessing Methods . . . . . . 34
4.3 AM4PC . . . . . . 36

**5 Prototypical Implementation** **39**
5.1 External Python Packages . . . . . . 39
5.2 General Overview of the Prototype . . . . . . 40
5.3 Additional Functionality . . . . . . 45

**6 Evaluation** **49**
6.1 Evaluation Setup . . . . . . 49
6.2 Accuracy . . . . . . 52
6.3 Runtime . . . . . . 56
6.4 Detailed Analysis of Preprocessing and Clustering Algorithms . . . . . . 61

**7 Conclusion and Outlook** **67**

**Bibliography** **69**

**A Appendix** **75**

# List of Figures


1.1 Overview of the steps constituting the KDD process [FPS96] . . . . . . . . . . . 15

3.1 An example tree-based pipeline from TPOT. Each circle corresponds to a ML operator, and the arrows indicate the direction of the data flow [OEM16]. . . . . . 26

4.1 PCA applied on two-dimensional feature data. The red and green arrows are the resulting axis of the new features. . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.2 AM4PC implemented methods from which one preprocessing method and one clustering method can be selected and applied. The colors specify which subcategory of feature transformation or feature extraction preprocessing methods fall into. . . 36
4.3 AM4PC conceptual flowchart (Full version in Figure 5.2). . . . . . . . . . . . . . 37

5.1 AM4PC package overview where blue markings indicate additions compared to AM4C. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
5.2 AM4PC full flowchart with additional settings. . . . . . . . . . . . . . . . . . . 45

6.1 Synthetic datasets generated with scitkit-learn. . . . . . . . . . . . . . . . . . . . . 49
6.2 Best mean AMI up to optimizer loop calculated for the evaluation scenario *AM4PC real*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
6.3 Best configurations mean AMI scores for different synthetic benchmark types. These are calculated for AM4C and AM4PC over five evaluation scenario runs. . 52
6.4 The four highest accuracy evaluation scenarios on real-world data with 5-10 evaluation runs each. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.5 Change in selection frequency of clustering algorithms when comparing *AM4PC* and *AM4PC + Multiobjective*. Mean runtimes increase from left to right. . . . . 59
6.6 Change in selection frequency of preprocessing methods when comparing *AM4PC* and *AM4PC + Multiobjective*. Mean runtimes increase from left to right. . . . . 59
6.7 Trade-off between accuracy and runtime visualized for 7 evaluation scenarios. . . 61
6.8 Number of winning configurations in which a clustering algorithm or preprocessing method occurs calculated over all 85 evaluation runs. . . . . . . . . . . . . . . . 62
6.9 Clustering algorithms mean runtime compared to AMI. . . . . . . . . . . . . . . 63
6.10 Preprocessing methods mean runtime compared to AMI. . . . . . . . . . . . . . 64

# List of Tables


2.1 Common HPO algorithms where $n$ is the total number of hyperparameter values and $k$ the number of hyperparameters [YS20]. . . . . . . 20
2.2 Overview of relevant clustering algorithms based on [Abd22; Alo13; AYS17; Hua98; RA18; XT15]. . . . . . . 22

5.1 Example CASH search space for preprocessing methods. . . . . . . 43
5.2 Example cash SearchSpace for clustering algorithms . . . . . . . 44

6.1 Meta-Features of real-world benchmark suite. . . . . . . 50
6.2 AM4C compared to AM4PC evaluated on real-world datasets mean AMI across all evaluation runs. Improvements are marked in bold. . . . . . . 53
6.3 Pearson correlation between CH on differently preprocessed datasets compared and AMI / ARI. . . . . . . 55
6.4 AM4C versus AM4PC optimized on different internal CVMs. . . . . . . 55
6.5 Pearson correlation between internal and external CVMs on real-world datasets . 56
6.6 AM4C vs. AM4PC mean runtime per optimization loop in seconds calculated on 10 evaluation scenario runs. . . . . . . 57
6.7 AM4PC vs. AM4PC + Multiobjective mean runtime per optimization loop calculated on 10 evaluation scenario runs. Improvements are marked in bold. . . 58
6.8 Different sampling + tuning strategies AMI and runtime compared to optimization on the full dataset (AM4PC). . . . . . . 60
6.9 Highest correlation between preprocessing methods and clustering algorithms / datasets. . . . . . . 65

A.1 Evaluation Scenarios, where deviations from the standard scenario *AM4PC real* are underlined. . . . . . . 75
A.2 Preprocessing methods selection counts per dataset . . . . . . . 76

# List of Algorithms

2.1 SMBO algorithm [HZC21] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

# Acronyms

**AM4C** AutoML4Clust. 16

**AM4PC** Automatic Machine Learning for Preprocessing and Clustering. 39

**AMI** Adjusted Mutual Index. 24

**ARI** Adjusted Rand Index. 24

**AutoML** Automated Machine Learning. 16

**BO** Bayesian Optimization. 18

**CASH** Combined Algorithm Selection and Hyperparameter optimization. 17

**CH** Calinski-Harabasz. 23

**CVM** Cluster Validation Measure. 17

**DBCV** Density Based Cluster Validation. 23

**GMM** Gaussian Mixture Mode. 22

**HPO** Hyperparameter Optimization. 17

**ICA** Independant Component Analysis. 33

**KDD** Knowledge Discovery in Databases. 15

**KNN** K-Nearest-Neighbour. 34

**LDA** Linear Discriminant Analysis. 33

**ML** Machine Learning. 15

**PCA** Principle Component Analysis. 28

**SMAC** Sequential Model-based Algorithm Configuration. 19

**SMBO** Sequential Model-Based Optimization. 19

**SVD** Single Value Decomposition. 33

# 1 Introduction

Data is being created at a record rate. From the year 2010 to 2014 data creation increased by roughly a factor of 72 from around 1ZB to a predicted 72ZB [VOE11]. This is largely due to the Internet of Things [LXZ15; VOE11] and its continued rise in popularity.

To deal with this unprecedented amount of data new methods had to be developed, as current technology is far more capable of storing and gathering data than analyzing it [FPS96]. Nowadays analysis is often done with Machine Learning (ML). The usual process of training and building ML pipelines requires significant expertise in both ML and the problem domain from which the data originates [OEM16]. This combination can be difficult to find, as significantly more domain experts exist than data scientists [OEM16]. Villars et. al. predict that those who embrace this new reality will thrive, while those unable to adapt to this shift will increasingly be left behind [VOE11].

The process of extracting value from data is often coined as Knowledge Discovery in Databases (KDD) [FPS96] and contains the steps shown in Figure 1.1.
The first step is data selection. Optimally this occurs by extracting the data from a structured database, but in reality, often requires the tedious merging of various semi-structured data sources with different physical locations and formats. This step is finished when all sources are unified in a homogeneous format that can be easily used for further steps.

After selecting and formatting the data it can be preprocessed. This is the main focus of this work and involves cleaning as well as modifying and expanding the data. The cleaning involves steps such as dropping duplicates, removing outliers, and filling in missing values. Once clean, the data is usually modified by scaling, encoding or reducing certain features before they are expanded and combined via feature engineering.

After preprocessing, the data is transformed into a format that can be used in the next step Data Mining. This involves selecting and tuning algorithms on the formatted data with the goal of maximizing performance. Finally, the results can be interpreted and for example, used to make business decisions.

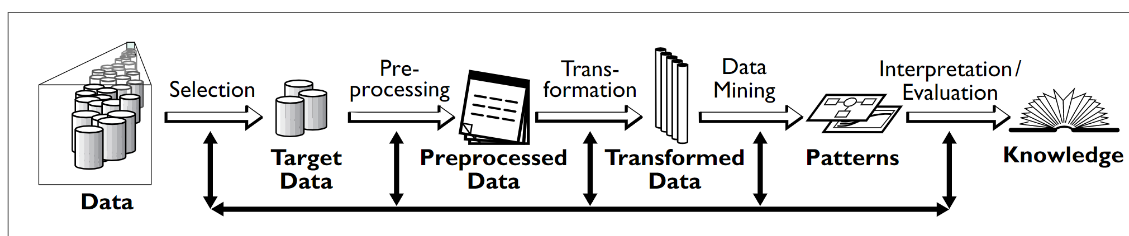


**Figure 1.1:** Overview of the steps constituting the KDD process [FPS96]

To reduce the complexity and increase the accessibility to non-experts, the discipline of Automated Machine Learning (AutoML) has been continuously studied for the past decade. It has found widespread use in both industrial as well as academic settings [HZC21]. AutoML can cover feature engineering, model selection, algorithm selection, and evaluation while letting humans focus on difficult-to-automate tasks such as data collection [YWC+18]. Multiple widespread and easy-to-use implementations exist such as TPOT [OEM16], Auto-WEKA [THHL13], and auto-sklearn [FKE+15]. Most AutoML systems focus on supervised learning, a problem setting in which both the data and the optimal classification are known. This allows the system to compare predicted labels with ground truth labels, which in turn can be used as feedback on how well the specific pipeline is performing. A pipeline hereby defines an end-to-end combination of data processing steps and ML Models. That information can then be used in the following optimization procedure, to decide whether to pursue similar pipelines (exploitation) or try different pipelines (exploration).
Significantly less progress has been made in the field of unsupervised AutoML, where no labels exist and algorithms need to rely on intrinsic characteristics of the data to find patterns and generate knowledge. The most common and important example of unsupervised learning is clustering, which aims to partition unknown data into clusters that are as similar as possible while being as different as possible from other clusters [XT15].

Nowadays multiple AutoML solutions for clustering, namely AutoML4Clust [TFS21], AutoCluster [LW21], AutoClust [PDK20], cSmartML [ELS21] and TPE-AutoClust [ES22], have been developed. In general, these do not consider prepossessing, unlike most currently popular AutoML solutions for supervised learning such as TPOT [OEM16], Auto-WEKA [THHL13], and auto-sklearn [FKE+15]. Since preprocessing data has been shown to improve results [DH04; LLL+18; WGL20], especially in cell-type clustering [WGL20], the focus of this work lies in improving the package AutoML4Clust (AM4C) [TFS21], largely by adding preprocessing.

The contributions of this work are:

- Analysis of existing preprocessing methods in regards to clustering.
- Expansion of the search space with preprocessing methods and their respective hyperparameters.
- Addition of categorical clustering algorithms.
- Runtime optimization while not neglecting quality by way of multiobjective optimization.
- Runtime optimization through sampling.
- Rigorous evaluation on the above-mentioned concepts

This work is structured as follows: Chapter 2 contains the necessary knowledge about AutoML, clustering and cluster validation measures. This is followed by Chapter 3, which summarizes related work. Chapter 4 analyses and selects promising preprocessing methods while Chapter 5 discusses their implementation as well as further features such as sampling. These implementations are evaluated in Chapter 6 upon both accuracy and runtime before the findings are summarized in Chapter 7.

# 2 Background

This chapter provides the necessary background for this work. It begins with AutoML in Section 2.1 which describes AutoML and Hyperparameter Optimization (HPO). The clustering algorithms which are applied during this process are described in Section 2.2. Finally Cluster Validation Measures (CVMs) are summarized in Section 2.3.

## 2.1 AutoML

To offset development costs and improve accessibility, AutoML, which includes automating large parts of the classic ML pipeline, was developed. This reduces the demand for data scientists, enables domain experts to analyze their own data [HZC21], and allows humans to focus on difficult-to-automate tasks such as data collection [YWC+18].

Yao et al. [YWC+18] defines AutoML as

$$\max_{\text{configrations}} \quad \text{performance of learning tools}$$
$$\text{s.t.} \begin{cases} \text{limited (or no) human assistance} \\ \text{limited computational budget} \end{cases}$$

where a configuration contains all factors except the model parameters, which are obtained through training, that influence the performance of the ML pipeline. Further, the core goals are described as good generalization performance, less assistance from humans, and high computational efficiency.

Auto-Weka [THHL13] was one of the first works that both selected an algorithm and tuned its hyperparameters. This is called the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem and is defined as

$$A^*_{\lambda^*} \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}\right).$$

$\mathcal{A}$ represents the set of algorithms and $\Lambda^{(j)}$ the associated hyperparameter spaces. $\mathcal{D}$ represents the dataset which which is split into $k$ splits $\mathcal{D}^{(i)}_{\text{train}}$ and $\mathcal{D}^{(i)}_{\text{valid}}$ via cross-validation. Therefore, $A^*_{\lambda^*}$ represents the best algorithm and hyperparameter combination validated on k splits.

As the combined space if often high dimensional and the loss function $\mathcal{L}$ noisy, finding a solution is time-consuming and computationally expensive [THHL13]. This process is dubbed HPO (Hyperparameter Optimization).

### 2.1.1 Hyperparameter Optimization Techniques

HPO aims to find well performing solutions to the CASH problem (see Section 2.1) in a maximally efficient and robust manner.
This is challenging, as the objective function is usually a non-convex non-differentiable function which prevents most traditional optimization techniques from finding success. The search space also contains continuous, discrete, categorical, and conditional hyperparameters, which further eliminates traditional solutions based on purely numerical or continuous variables. Further many black-box optimization models neglect function evaluation time, which is a significant factor in HPO for ML models [THHL13; YS20].

While many different HPO techniques exist, they usually follow the same base steps [YS20]:

1. Define an objective function, performance measures, and relevant hyperparameters.
2. Start with a large search space, then narrow the search space to well-performing areas (exploitation) and periodically explore new areas (exploration).
3. Return the best-performing configuration.

In the following, popular HPO techniques are described, each with their own different strengths and drawbacks as summarized in Table 2.1 [YS20].

#### Grid and Random Search

Grid search systematically divides the search space configurations, whereas random search selects random configurations. Configurations can contain both algorithms and their respective hyperparameters. Both return the best performing configuration [HZC21; YS20].
Random Search has been empirically and theoretically proven to be more practical and efficient than grid search [HZC21].

#### Bayesian Optimization

Bayesian Optimization (BO) is a more efficient method for optimizing expensive black-box functions [HZC21]. BO generates a probabilistic model, often referred to as a surrogate function, that is used to predict good hyperparameter combinations. This method avoids training and testing too many computationally expensive ML models [EFH+13; YS20]. During this process, a balance between the exploration of new areas and the exploitation of promising areas [EFH+13] is reached.

The most popular acquisition function that combines these two goals is *expected improvement* [SWJ98] which determines the next hyperparameter configuration [EFH+13] is shown below:

$$\mathbb{E}_{\mathcal{M}}\left[I_{f_{\min}}(\boldsymbol{\lambda})\right] = \int_{-\infty}^{f_{\min}} \max\{f_{\min} - f, 0\} \cdot p_{\mathcal{M}}(f \mid \boldsymbol{\lambda}) df.$$

Here $f$ represent previous point evaluations, $p_{\mathcal{M}}$ the surrogate model and $\lambda$ a configuration. The uncertainty is quantified by $\max\{f_{\min} - f, 0\}$ while $p_{\mathcal{M}}(f \mid \lambda)$ the predicted performance. If either uncertainty or predicted performance is high the configuration $\lambda$ is likely to be chosen.

Sequential Model-Based Optimization (SMBO) combines this into the Algorithm 2.1.

**Algorithm 2.1** SMBO algorithm [HZC21]

**INPUT** $\mathcal{M}, \Theta, \mathbb{E}_{\mathcal{M}}, p_{\mathcal{M}}$
  $\mathcal{D} \leftarrow INITSAMPLES(\mathcal{M}, \Theta)$
  **for** `i in [1,2,...T]` **do**
    $p_{\mathcal{M}}(y|\theta, \mathcal{D}) \leftarrow FITMODEL(p_{\mathcal{M}}, \mathcal{D})$
    $\theta_i \leftarrow argmax_{\theta \in \Theta} \mathbb{E}_{\mathcal{M}}(\theta, p(y, \theta, \mathcal{D}))$
    $y_i \leftarrow \mathcal{M}(\theta_i)$    // Expensive step
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\theta_i, y_i)$
  **end for**

Here $\mathcal{M}$ represents the evaluation function, $\Theta$ the search space, $\mathbb{E}_{\mathcal{M}}$ the acquisition function and $p_{\mathcal{M}}$ the surrogate model.

First, the record dataset $\mathcal{D}$ is filled with some sample pairs $(\theta_i, y_i)$ using the evaluation function $\mathcal{M}$ and the search space $\Theta$. Then the probabilistic model $p_{\mathcal{M}}$ is fitted using the record dataset $\mathcal{D}$. The probabilistic model enables the acquisition function $\mathbb{E}_{\mathcal{M}}$ to select the next promising configuration $\theta_i$. That configuration $\theta_i$ is evaluated by $\mathcal{M}$ and the result added to the record dataset $\mathcal{D}$.

A popular implementation of SMBO is Sequential Model-based Algorithm Configuration (SMAC), which uses random forests as a surrogate model [EFH+13; HHL11].

#### Multi-fidelity Algorithms

Due to the long execution times, which scale with both the size of the search space and the size of the dataset, multi-fidelity algorithms were developed.

This type of algorithm starts by evaluating a small sample of the dataset on many different configurations. Promising configurations are then continuously evaluated on larger samples, while poorly performing configurations are dropped. This continues until the configurations are evaluated on the whole dataset [YS20].

Two of the most well-known implementations of this concept are successive halving and its extended version hyperband [LJRT18; YS20].
Successive halving starts by running many configurations with a low budget, the top 50% go on to be evaluated with twice the previous budget and so on. Both the number of configurations and the budget are predetermined with the initialization of the algorithm.
Hyperband aims to improve upon this concept by dynamically allocating both the budget and the number of sampled configurations. It has been shown to outperform successive halving [YS20].

| HPO Method | Strengths | Limitations | Time Complexity |
|---|---|---|---|
| *GS* | Simple<br>Parallelization | Time-Consuming<br>Only Efficient with categorical HPs | $O(n^k)$ |
| *RS* | More efficient than GS<br>Parallelization | Does not consider previous results<br>Not efficient with conditional HPs | $O(n)$ |
| SMAC | Efficient with all types of HPs | Poor capacity for parallelization | $O(nlogn)$ |
| *Hyperband* | Parallelization | Not efficient with conditional HPs<br>Subsets with small budgets must be representative | $O(nlogn)$ |

**Table 2.1:** Common HPO algorithms where $n$ is the total number of hyperparameter values and $k$ the number of hyperparameters [YS20].

## 2.2 Clustering

Unlike supervised learning which relies on ground truth labels, unsupervised learning focuses on data with unknown labels relying only on the inherent structure of the data. Clustering is one of the most important unsupervised learning methods and aims to partition data into clusters. These clusters strive to fulfill the following goals [XT15]:

- Instances in the same cluster should be as similar as possible.
- Instances in different clusters should be as dissimilar as possible.
- Measures for similarity and dissimilarity (CVM) must be unambiguous, fitting and practical.

Jain et. al. [JD88] list a number of challenges this task poses:

(a) What is a cluster?

(b) What features should be used?

(c) Should the data be normalized?

(d) Does the data contain any outliers?

(e) How do we define the pair-wise similarity?

(f) How many clusters are present in the data?

Successful clustering can provide insight into the underlying structure, possible hypotheses, anomalies, and defining features. Additionally, it acts as a type of natural classification comparable with supervised learning. Lastly, it can compress, organize and group data by using the cluster centers [Jai10].

In the following sections well known partitional, hierarchical, distribution-based, and density-based clustering algorithms are discussed. These are summarized in Table 2.2.

### 2.2.1 Partitional Clustering Algorithms

The core idea of these types of algorithms is to use the center of clusters (centroids) to assign datapoints to clusters [XT15].
They have the advantages of high efficiency and low complexity, but the disadvantages of only working on convex data, sensitivity to noise, only providing local optima, and requiring prior knowledge of the number of clusters [XT15].

#### K-Means and K-Medoids

In the K-Means algorithm, the cluster centroids are randomly initialized by sampling the datapoints. Then every point is assigned to its nearest centroid. The centroids are newly calculated as the mean value of datapoints contained in each cluster before the cycle repeats [Mad12].
K-Medoids functions similarly, except that instead of simply using the calculated mean, it uses the nearest datapoint to it, as the new centroid. This reduces sensitivity to outliers [XT15].

#### K-modes and K-Prototypes

K-Means is one of the most widely used clustering algorithms, but it requires all data to be numerical. K-Modes [Hua98] solves this problem by using a simple matching dissimilarity measure instead of the classic distance measure which quantifies how many categorical values overlap. Centroids are instead chosen as modes, which represent the most common category of every attribute in a cluster. This algorithm is further improved upon to work on mixed categorical and numerical data as K-Prototypes [Hua97; Hua98].

#### Affinity Propagation

This algorithm calculates an affinity between datapoints based on Euclidean distance and a greedy approach. If this value is high there is a higher chance of the datapoints becoming a centroid [FD07]. Advantages are requiring no knowledge about the number of clusters, and resistance to outliers, while disadvantages are the high time complexity and sensitivity to hyperparameters [XT15].

### 2.2.2 Hierarchical clustering Algorithms

The basic idea of hierarchical clustering algorithms is to determine hierarchical dependencies. For agglomerative clustering, every datapoint is initialized as a cluster. The most similar clusters are then recursively merged until a threshold, such as the number of cluster centers or minimum CVM, is reached [XT15].
The advantages of this type of algorithm are the applicability to most data shapes and scalability. This type of algorithm suffers from a high complexity and requires prior knowledge of the number of clusters present in the data.

Prominent examples of this type of clustering algorithm are birch [ZRL96] and agglomerative clustering with different linkage criteria such as *ward* or *complete* [GK78].

### 2.2.3 Distribution Based Clustering Algorithms

Distribution based clustering algorithms rely on the concept, that the datapoints are sampled from multiple different mathematical distributions. If a datapoint is part of such a distribution it is assigned to the respective cluster with a certain probability depending on where in the distribution it lies [XT15].
This provides many advantages, one of these is the fact that a probabilistic assignment to a cluster is more realistic than a black-and-white true or false. Further, these types of algorithms are highly scalable and rooted in traditional statistics. Whereas its disadvantages are a high time complexity and the many hyperparameters that need to be tuned.

A classic example of this type of algorithm is Gaussian Mixture Mode (GMM) [Ras99], which uses Gaussian distributions.

### 2.2.4 Density Based Clustering Algorithms

The premise of these types of algorithms is that areas with a high datapoint density represent clusters [XT15].
The advantages are efficiency and applicability to all data shapes. At the same time, this type of algorithm is heavily reliant on a uniform datapoint density [Mad12; XT15] and sensitive to its hyperparameters.

The most popular implementation is DB-SCAN [EKSX+96], which starts by identifying so-called core points. These points must have at least $k$ nearby points with less than a certain distance defined by $eps$. A cluster is initialized by selecting a random core point, and adding all core points recursively that are near enough. Once no further core points can be added, non-core points are added that are near enough before a new cluster is initialized by randomly selecting a new yet-to-be-classified core point.
Further popular implementations include OPTICS [ABKS99] and Mean-Shift [Che95].

| **Category** | **Algorithm** | **Complexity** | **Scalability** | **Shape** | **Sensitive to noise** |
|---|---|---|---|---|---|
| *Based on partition* | K-Means | Low | Middle | Convex | Highly |
| | K-Medoids | High | Low | Convex | Little |
| | Mini-Batch-K-Means | Low | High | Convex | Medium |
| | K-Modes | Middle | Middle | Convex | Little |
| | K-Prototypes | Middle | Low | Convex | Little |
| | Affinity Propagation | High | Low | Convex | Little |
| *Based on hierarchy* | Birch | Low | High | Convex | Little |
| *Spectral Clustering* | Spectral | High | Low | Arbitrary | High |
| *Based on distribution* | GMM | High | High | Arbitrary | Little |
| *Bases on density* | DB-SCAN | Middle | Middle | Arbitrary | Little |
| | Mean-Shift | High | Low | Arbitrary | Little |

**Table 2.2:** Overview of relevant clustering algorithms based on [Abd22; Alo13; AYS17; Hua98; RA18; XT15].

## 2.3 Cluster Validation Measures

In supervised learning a common performance measure is the F-score. In clustering this is more difficult since there is typically no a-priori knowledge of the ground-truth labels [CBL15]. CVM (Cluster Validation Measures) that do not use ground truth labels are called internal validation measures, while ones that rely on ground truth labels are referred to as external validation measures. These are especially useful for scientific evaluation due to their exactness but find little applicability in actual use-cases.

### 2.3.1 Internal Validation Measures

Internal validation measures use intrinsic properties in the data to score the clustering result [CBL15]. Usually, properties such as the compactness and separation of clusters are used as these represent the intrinsic goal of clustering [JD88]. Methods differentiate themselves in how these concepts are defined and combined.

#### Calinski-Harabasz

The validation measure Calinski-Harabasz (CH) [CH74; LLX+10] is based on the ratio of between-group sum of squares $BGSS$ in comparison to the within-group sum of squares $WGSS$ and is defined as follows:

$$CH = \frac{BGSS}{k-1} / \frac{WGSS}{n-k}.$$

Here $k$ is the number of clusters and $n$ the number of datapoints. CH ranges from 0 to $\infty$ where a higher value represents a higher accuracy.

#### Density Based Cluster Validation

Density Based Cluster Validation (DBCV) [MJC+14] transforms the datapoint to a space containing reachability distances with the aim of quantifying density in different regions. Then minimum spanning trees are calculated for each cluster to detect non-convex shapes. The final score is then calculated by combining compactness, here defined as the maximal weight in each minimum spanning tree, and separation, defined as the minimum reachability distance between internal datapoints in a cluster [CBL15; MJC+14].
CH ranges from $-1$ to 1 where a higher value represents a higher accuracy. The main advantage of this CVM is the ability to adequately score non-convex data.

### 2.3.2 External Validation Measures

For development purposes accurate validation is paramount, which internal validation measures do not always provide. This is why external methods are relevant.
It is important to understand that validating internal measures with external measures can be dangerous [CBL15], as results can be misleading. This stems from the fact that multiple equally good clustering results can exist, but external CVMs will always prefer the one nearest to the ground truth labels.
The most widely used external validation measure for AutoML clustering is the Adjusted Rand Index (ARI) [LW21; PDK20].

#### ARI and AMI

The ARI (Adjusted Rand Index) external CVM is an improved version of the Rand Index. The main improvement is correcting for random chance, as the rand index will not return a constant value when used upon random distributions [HA85; VEB09]. This measure suffers when applied to unbalanced or small clusters, which is corrected for with the Adjusted Mutual Index (AMI) [RVBV16].

# 3 Related Work

Little to no research addresses the CASH problem for a combined search space of both clustering algorithms and preprocessing methods. What does exist are a multitude of supervised AutoML packages that integrate preprocessing, unsupervised AutoML systems that focus on clustering, and use-cases where the combination of preprocessing with clustering is analyzed.

Therefore, this chapter discusses related work specifically in the area of AutoML with preprocessing, AutoML for clustering, and the combination of preprocessing and clustering.

## 3.1 Supervised AutoML systems with preprocessing

This section focuses on the two most popular, according to GitHub stars[1], supervised AutoML packages that include preprocessing and do not focus on neural networks. Namely *TPOT*[2] [OEM16] and *auto-sklearn*[3] [FKE+15].

*TPOT* is an Python package based on scitkit-learn [PVG+11]. It uses genetic programming to optimize a series of preprocessing methods and ML models with the goal of maximizing classification accuracy.
*TPOT* implements methods that modify the data in some way before returning it such as Standard Scaler, Robust Scaler, MinMax Scaler, MaxAbs Scaler, Randomized PCA, Binarizer and Polynomial Features.
Additionally, the package implements feature selection methods that reduce the number of features in the form of VarianceThreshold, SelectKBest, SelectPercentile, SelectFwe, and RecursiveFeatureElimination.

Genetic programming is used to automatically evolve and improve a population (collection) of tree-like pipelines as seen in Figure 3.1.
This is implemented with the Python package DEAP[4] and follows a standard genetic programming procedure: 100 random pipelines are created and evaluated. The 20 top-performing pipelines are chosen via a multiobjective approach, which aims to both maximize accuracy and minimize tree size.
These are then used to construct the next generation, where 5% are crossed over with other offspring. This is done with one-point crossover, which splits the pipelines of both the parents and swaps them

[1] https://github.com/topics/automl
[2] https://github.com/EpistasisLab/tpot
[3] https://github.com/automl/auto-sklearn
[4] https://github.com/DEAP/deap

at a certain point. 90 % of the remaining offspring are randomly mutated. The pipelines are then trained and tested. Finally, the best (non-dominated) solutions are updated and the cycle repeats for 100 iterations (generations) before the highest accuracy pipeline in the Pareto front is returned.

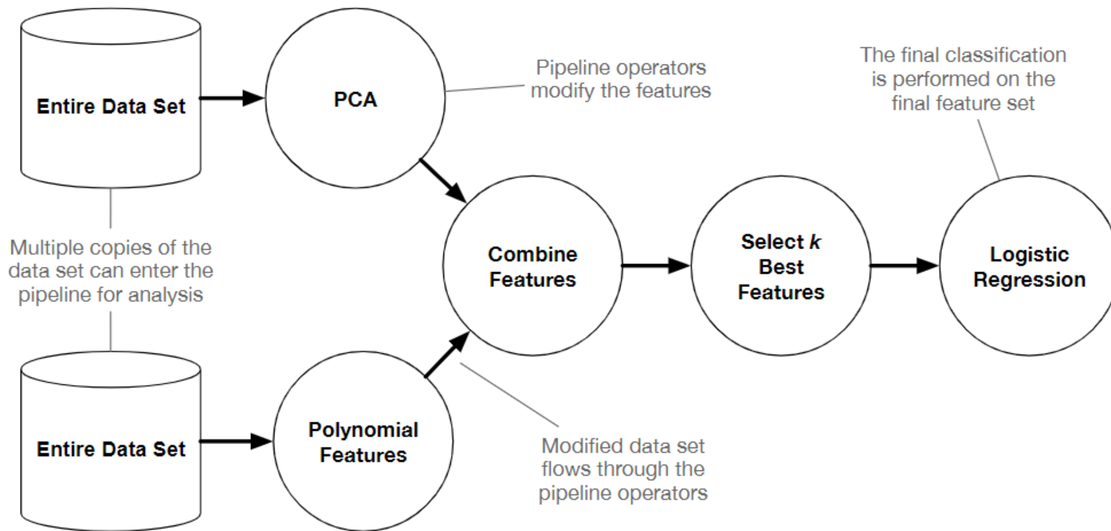


**Figure 3.1:** An example tree-based pipeline from TPOT. Each circle corresponds to a ML operator, and the arrows indicate the direction of the data flow [OEM16].

Evaluated on 150 benchmarks and compared to a random forest with 500 trees, *TPOT* outperforms in 21 cases, underperforms in 4 cases and provides comparable results in 125 cases.

Similarly *auto-sklearn* is also based on scikit-learn [PVG+11] and used the HPO optimizer SMAC (see Section 2.1.1) implemented in SMAC3 (see Section 5.1).
The Python package implements 15 classification algorithms, 14 preprocessing methods as well as 4 data preprocessors. In total this results in 110 hyperparameters.

*Auto-sklearn* defines data preprocessors as methods that adapt feature values and are used when whenever applicable. These include one hot encoding, imputation, balancing, and rescaling.
Feature preprocessing methods modify the feature values and include methods such as feature selection, kernel approximation, matrix decomposition, embeddings, feature clustering, and polynomial feature expansion.

These packages demonstrate that preprocessing can lead to a significant improvement in supervised learning.

## 3.2 AutoML systems with Clustering

The selection of packages implementing AutoML for clustering is small, prototypical, and in large disregards preprocessing with the exception of *TPE-AutoClust* [ES22] and *autocluster*[5].

One such package is *AM4C*(AutoML4Clust) [TFS21], a generic package in the sense that it provides multiple different optimizers, clustering algorithms, and validation measures.
Available optimization strategies are RandomSearch, BO (Bayesian Optimization), Hyperband, and combined Bayesian Optimization and Hyperband. These are used to optimize partitional clustering algorithms including K-Means, Mini-Batch-K-Means, K-Medoids, and GMM. Clustering results can be evaluated on different internal CVMs such as CH (Calinski-Harabasz), Davies-Bouldin Index, and Silhouette Index as well as compared with external validation measures such as AMI.
Benchmarking is done on both synthetic and real-world datasets. The synthetic benchmarks are generated with different numbers of datapoints, dimensions, number of clusters, and amounts of noise.
The real-world datasets consist of five classification datasets from the UCI machine learning repository[6]. These are stripped of ground truth labels, non-numeric features, IDs, timestamps, and empty values.
Both Benchmark sets are evaluated with a budget of 60 optimizer loops on the external CVM AMI (Adjusted Mutual Index). Results on synthetic benchmarks are comparable to an exhaustive search and better when evaluated on real-world datasets. This is achieved with runtimes roughly 437x and 276x faster than exhaustive search.
When comparing which internal CVM and optimizer to use, CH with Hyperband achieves the best combination of low runtime yet high AMI.

*AutoCluster* [LW21] is another AutoML clustering package based on scikit-learn [PVG+11].
Meta-learning is used to extract clustering-oriented meta-features such as spacial randomness and data distribution. These meta-features are used in combination with 150 OpenML datasets[7] to train a model which predicts what clustering algorithm will perform best depending on the dataset. Performance is hereby measured according to the external CVM ARI.
The package implements K-Means, Affinity Propagation, Mean Shift, Agglomerative Clustering, DB-SCAN and BIRCH as its clustering algorithms selections. After a model is selected its hyperparameters are further optimized by way of GridSearch (see Section 2.1.1).
To combat the fact that internal CVMs are imperfect, multiple different clustering algorithms are optimized with different validation measures. These are then combined in an ensemble model via majority voting, which reduces errors in measurement correlated to the internal CVMs.
Compared to the six clustering algorithms with default hyperparameters and 3 further partitional clustering algorithms, AutoCluster outperforms in 15 out of 33 Benchmarks and is otherwise comparable.

[5] https://github.com/wywongbd/autocluster
[6] https://archive.ics.uci.edu/ml/index.php
[7] https://www.openml.org/search?type=data&sort=runs&status=active

*TPE-AutoClust* [ES22], an improved version of *cSmartML*, does implement rudimentary preprocessing. They achieve this by adding the four preprocessing methods Standard Scaling, SimpleImputer, KNNImputer and Principle Component Analysis (PCA) to their search space including relevant hyperparameters.

Further notable packages which focus on meta-learning are *AutoClust* [PDK20] and *cSmartML* [ELS21] which also aims to combat CVM uncertainty by using multiple validation measures and a multiobjective approach. Lastly *autocluster*[8] exists which includes rudimentary dimensionality reduction as part of its search space.

## 3.3 Clustering with Preprocessing

Clustering is extensively used in the field of scRNA-seq [KAH19] to identify cell types. This field is especially challenging for clustering due to the prevalence of high dimensional large datasets and noisy data [KAH19].
To combat these factors, Wang et. al. [WGL20] implement and analyze multiple preprocessing methods and clustering algorithms. Five common RNA clustering algorithms are used: Dynamic-Treecut, T-SNE + K-Means, SNN-clip, PCA Reduce, and SC3. These are combined with one of four preprocessing methods: Log Transform, Standard Scaler, ScTransform, and No Transformation.
Benchmarking is done on eight popular scRNA-seq datasets [WGL20] and evaluated on ARI. Results indicate that different clustering algorithms react differently to different preprocessing methods depending on the dataset. For example, DynamicTreecut performs best with Log Transform, but only for six of the eight benchmark datasets.
Log Transform and ScTransform are usually applicable, but sometimes No Preprocessing or the Standard Scaler also performs well.
Preprocessing is shown to offer significant improvements, depending on the dataset, of up to 3 times.

## 3.4 Summary of Related Work

It is clear from packages such as *TPOT* [OEM16] and *auto-sklearn* [FKE+15] that preprocessing plays a large role in supervised AutoML systems.

Significantly less research exists concerning unsupervised learning, but packages such as *AutoML4Clust* [TFS21] and *AutoCluster* [LW21] prove that AutoML for clustering is possible and leads to significant improvements over baseline models. The plausibility of integrating preprocessing methods into a AutoML clustering system is additionally proven in TPE-AutoClust [ES22] though not specifically analyzed.

Lastly, work such as [WGL20] indicates the potential improvements that can be gained from preprocessing data before clustering.

[8] https://github.com/wywongbd/autocluster

While no systematic analysis of preprocessing methods in AutoML clustering exists, the prior factors suggest there to be significant merit in adding and analyzing preprocessing methods in an AutoML clustering system.

# 4 Transfer of Preprocessing to Clustering

Preprocessing plays a major role in most KDD pipelines, but has largely been neglected in unsupervised AutoML up to now (see Chapter 3). Additionally, the literature comprises far less research into preprocessing for unsupervised learning when compared to supervised learning [GRL+16]. This makes the task of integrating preprocessing into an AutoML for clustering system challenging.

Integrating preprocessing is essential, as model performance strongly correlates with the quality of the data [YWC+18]. This is especially important when dealing with high-dimensional and possibly redundant, unscaled, or noisy data [GRL+16; KKP06; SJ13] as ML algorithms performance strongly degrades otherwise.
Even when dealing with clean datasets, preprocessing remains important as it formats the data for the subsequent ML algorithm being used [GRL+16]. For the *K-Means* algorithm this could be reducing the dimensionality with *PCA* to avoid the Curse of Dimensionality [IM98]. The selection of preprocessing methods usually requires human expertise with extensive domain knowledge, as well as trial and error.
Therefore, the addition of preprocessing in AutoML has the goal of reducing human time investment and intervention [YWC+18]. It has been shown to successfully improve supervised AutoML (see Section 2.1).

In clustering systems and use-cases, most datasets are preprocessed by hand to improve clusterability as in [PDK20]. The goal of this work is the integration and automation of this step in an AutoML clustering system.
To achieve this, available preprocessing methods are surveyed and categorized in Section 4.1. The choice of preprocessing methods to implement is described in Section 4.2. This choice is made under the consideration of several factors, including the size of the resulting search space, prevalence in other AutoML packages, and popularity of use in combination with clustering algorithms. Finally, the implementation of the selected preprocessing methods is discussed in Section 4.3.

## 4.1 Preprocessing Methods for Clustering

It is difficult to clearly categorize and group preprocessing methods as literature differs in both terminology and assignment. Here definitions are used similar to [GRL+16; HZC21] and [GLH15]. Therefore this work differentiates between feature engineering and data reduction.

### 4.1.1 Feature Engineering

It is generally accepted that the dataset and its features determine the upper limit of ML performance. Algorithms and models can only aim to approximate this limit [HZC21].
Therefore feature engineering has the goal of maximizing the quality of features from the raw data. These methods can further be categorized into methods that select a subset of features, methods that transform existing features, and methods that extract new features [HZC21].

#### Feature Selection

As the average size and feature count of datasets continually increase, it can be important to only consider the most informative features while dropping redundant, less informative features [HZC21; TM21]. This has the added benefit of improving robustness to outliers and can make ML algorithms both faster and more effective [KKP06].
Common methods include *VectorSlicing*, *RFormula* and *Chi-SquaredSelector* [GRL+16]. For example, *Chi-SquaredSelector* uses the Chi-Squared test [Pea00] to determine which features most strongly influence the ground truth labels and selects these. This method is therefore only applicable in supervised learning. Unsupervised selection methods such as *SVD-Entropy* do exist as described by Fernandez et. al. [SCM20] but are less commonly used.

#### Feature Transformation

Unlike feature selection, which reduces the feature space, feature transformations maintain the size of the original feature space only transforming existing features.

These include simple scaling methods such as standardization, normalization, feature discretization, and binarizing [GRL+16; YWC+18]. These types of methods alleviate sensitivity to distance measures [Gia16] and ensure the validity of the model [FCW+21]. This is important when feature range or distribution differs as otherwise, a feature that ranges from $[0, 100]$ will heavily impact the clustering result while a feature with a range $[0, 1]$ makes little difference. The most popular methods include *MinMaxScaling* and *StandardScaling* [FCW+21].

#### Feature Extraction

This type of feature engineering aims to create a new set of more informative features [GRL+16]. A common approach is to combine features in a linear or non-linear way [FCW+21], for example with polynomial features expansion (*Poly Features*). This method multiplies features with themselves or other features. From $[a, b]$ the features $[1, a, b, a^2, ab, b^2]$ could be extracted.
Further features can be constructed via the conjugation and disjunction of boolean features. For numerical features, methods such as minimum, maximum, addition, subtraction, and mean can be used. Lastly, methods such as the cartesian product can be used for nominal features [YWC+18]. The combination of features can improve clustering, for example when clustering families into

groups, it could be beneficial to create a feature *total income* instead of considering family members individually. Due to the numerous and diverse methods, the application largely depends on human expertise.

Further important methods are dimensionality-reducing mappings such as the popular method PCA which finds a set of maximally uncorrelated features [PF01] and extracts them as seen in Figure 4.1.

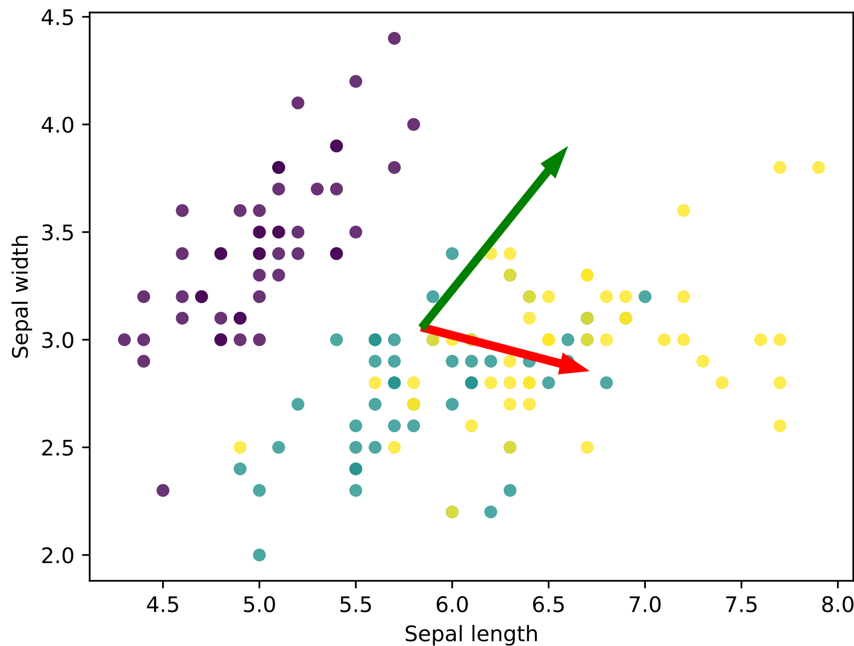


**Figure 4.1:** PCA applied on two-dimensional feature data. The red and green arrows are the resulting axis of the new features.

Additional common examples of the type of method are Linear Discriminant Analysis (LDA) [HZC21; YWC+18], Independant Component Analysis (ICA), Single Value Decomposition (SVD) and Isomap [HZC21]. Similarly to feature selection, this type of feature extraction mitigates the Curse of Dimensionality [IM98] with the added benefit of less information loss, as features are combined and not dropped.

### 4.1.2 Data Reduction

Another important area of Preprocessing is data reduction. The main reason is to combat increasing runtimes.
Data reduction can be achieved by reducing features with either feature selection [GRL+16] or feature extraction [YWC+18], but also by reducing the number of instances.
This second case is especially prevalent with datasets that are too large to train a model on. Here common methods include random sampling and stratified sampling, a method which aims to balance class distribution [KKP06]. The second method is not applicable to clustering as it requires ground truth labels.

## 4.2 Selection of Preprocessing Methods

From these preprocessing methods, a selection of those most applicable to clustering is made. Some preprocessing methods that are chosen are always applied when reasonable while some are only applied when selected by the optimizer.

### 4.2.1 Methods that are always applied

A number of methods are chosen that are always applied when relevant. This approach is similar to auto-sklearn [FKE+15] and other clustering use-cases such as [ES22].

Auto-sklearn includes four methods in their implementation:
**One-Hot encoding**, which transforms a categorical feature into a matrix of L1-columns [FCW+21] since real-world problems contain both categorical and numerical features [GRL+16; KKP06]. For every categorical feature additional features are created that contain either 0 or 1 for each unique value.
**Imputation**, the process of filling missing values, can be achieved with methods such as selecting the most common entry, mean substitution, regression [KKP06] and K-Nearest-Neighbour (KNN) [FCW+21]. Imputation must be handled with care as it can lead to wrong conclusions and biases [GRL+16] if performed poorly. KNN has been shown to achieve good results even when 5-15% of values are missing [GRL+16] and is used in packages such as TPE-Autoclust (see Chapter 3).
**Balancing** is another method that auto-sklearn implements, which tries to achieve an even class distribution in the datapoints.
**Scaling**, which normalizes feature ranges and distributions.

As without One-Hot encoding, a large portion of datasets can not be clustered as most clustering algorithms require numerical features, it is chosen to be applied in this work. Due to One-Hot encoding significantly expanding the number of features, label encoding is also sometimes applied if the number of unique categorical features is high. To avoid issues stemming from missing data, KNN is applied in this work and used to fill in any missing values. Balancing is only applicable when ground truth labels are present and therefore disregarded. Since multiple inherently different scaling methods exist, this work adds them to the search space instead of always applying one(see Section 4.2.2).
Therefore the only two preprocessing steps that are always applied when relevant are *One-Hot encoding* and *KNN* imputation.

### 4.2.2 Methods which are optimized

As the size of the search space heavily impacts runtime, it is important to only consider preprocessing methods that have the potential to improve clustering. Therefore, the selection of preprocessing methods to implement takes two factors into consideration:

- Prevalence in existing AutoML packages with preprocessing.
- Precedence as a method used in combination with clustering algorithms.

The analysis of other AutoML packages that implement preprocessing, namely TPOT [OEM16] and auto-sklearn [FKE+15], as well as use-cases and systems that combine preprocessing and clustering (see Chapter 3), yields the following observations.

**Feature selection** methods occur in both TPOT and auto-sklearn, for example *SelectPercentile*. But these methods add an additional degree of complexity [Tal99] and often require ground-truth labels. This combined with the lack of use-cases where feature selection is applied prior to clustering leads to them not being considered in this work.

**Feature transformation**, especially standardization and normalisation, is prevalent in both AutoML packages and clustering use-cases. TPOT implements *Standard Scaler*, *Robust Scaler*, *MinMax Scaler*, *MaxAbs Scaler* and *Binarizer* which largely overlaps with the methods auto-sklearn implements. Multiple clustering systems and use-cases implement the *Standard Scaler* and *Log Transformation* as well such as [ES22; WGL20].
As these methods fulfill both requirements listed in Section 4.2.2, a selection of them is added to the search space and evaluated (see Figure 4.2) in this work. More complex features that are constructed by combining features with operations such as conjugation, addition, etc. are not considered due to little precedent in other work and adding a large degree of complexity.

**Feature extraction** is similarly prevalent in both AutoML packages and clustering use-cases. Especially dimensionality-reducing mappings such as *PCA* are very popular. These are implemented in TPOT, auto-sklearn, and applied in multiple clustering use-cases [DH04; ES22; KCA15; WGL20]. Feature extraction methods that combine features are less popular with the exception of *Poly Features* as backed both by TPOT and auto-sklearn. Therefore mainly dimension-reducing mapping with the exception of *Poly Features* are added in this work.

This leads to the selection shown in Figure 4.2. These methods are implemented and added to the search space in combination with their relevant hyperparameters.

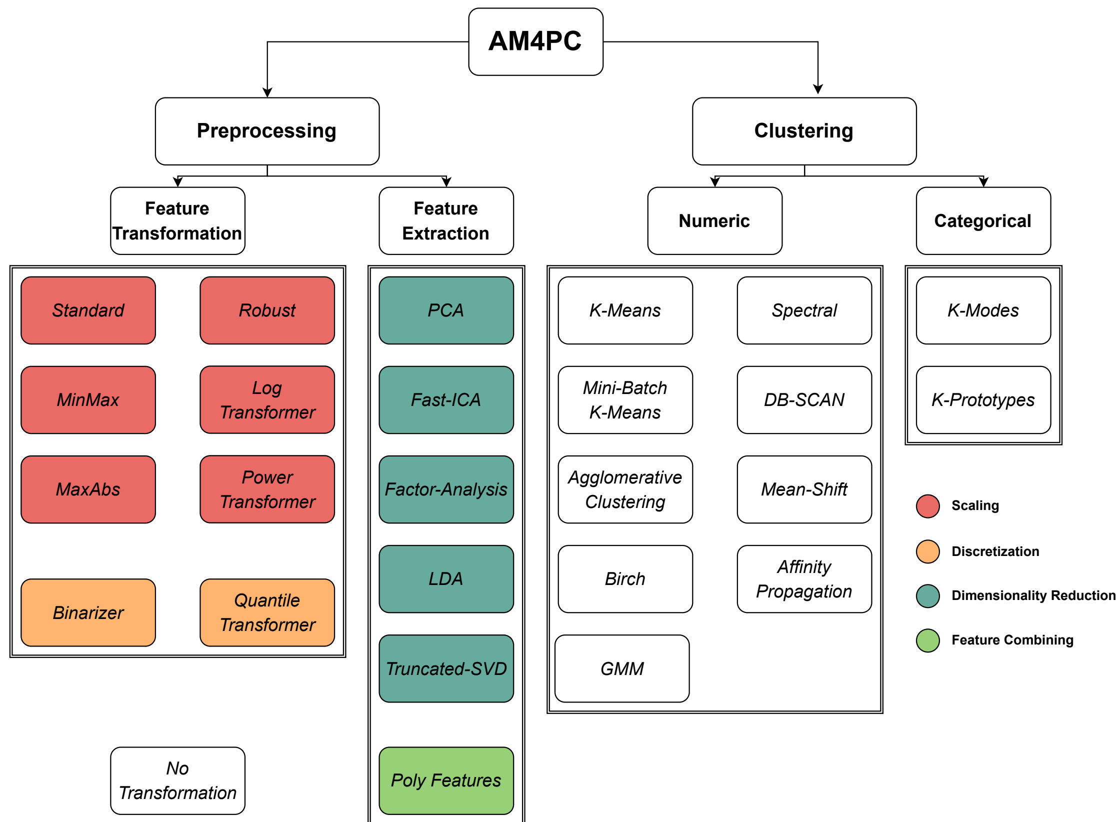


**Figure 4.2:** AM4PC implemented methods from which one preprocessing method and one clustering method can be selected and applied. The colors specify which subcategory of feature transformation or feature extraction preprocessing methods fall into.

## 4.3 AM4PC

The package AM4C (AutoML4Clust) [TFS21] offers many advantages such as being generic, containing many validation measures and optimizers, as well as being modular. This is particularly advantageous as its structure is very similar to auto-sklearn, which means it can be extended in a similar fashion.

The AM4C package has been continuously developed since its publication described in [TFS21]. Therefore [TFS21] does not reflect the starting point of the package. AM4C currently includes a multitude of internal and external validation measures, multiple optimizers, and nine clustering algorithms with a selection of their respective hyperparameters.

AM4C is extended by adding a benchmarking framework, data formatting steps, and preprocessing methods. This results in the following steps as shown in Figure 4.3.
It begins with the user either providing a dataset or loading an available benchmarking dataset. The dataset is then formatted in such a fashion that the subsequent preprocessing methods and clustering algorithms can be executed without any issues. This involves filling missing values and converting categorical features to numerical ones with One-Hot encoding or label encoding. Then

the configurations which include preprocessing methods, clustering algorithms, and their respective hyperparameters, are iteratively selected with BO (Bayesian Optimization) from the search space and executed. These configurations are limited to one preprocessing method and one clustering algorithm similar to auto-sklearn. A preprocessing pipeline that combines multiple preprocessing methods as in TPOT (see Chapter 3) would massively increase the search space and therefore runtime and complexity. In the end, the predicted labels and all tried configurations with their results are returned.

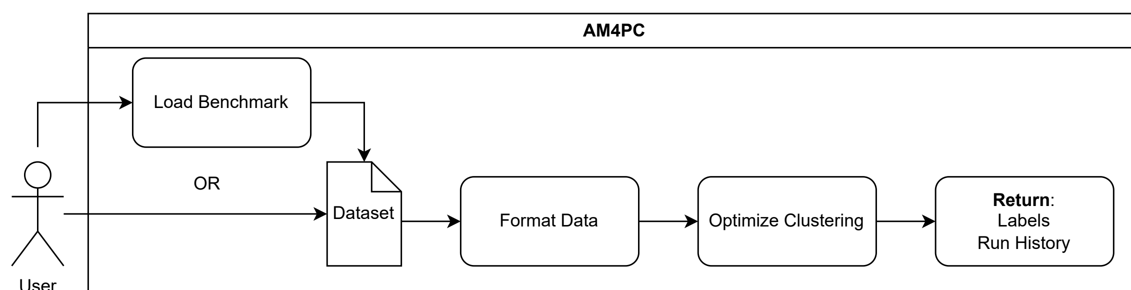


**Figure 4.3:** AM4PC conceptual flowchart (Full version in Figure 5.2).

AM4C is additionally extended by adding: categorical clustering algorithms to the search space, an option for multiobjective optimization, and sampling.
Categorical features are prevalent in clustering datasets. The conversion to numerical values with One-Hot encoding is imperfect and leads to high-dimensional and sparse feature data. Therefore the categorical clustering algorithms *K-Modes* and *K-Prototypes* are included as options in the search space.
Runtime is the major issue that AutoML faces as otherwise all combinations could be tried exhaustively. Adding preprocessing methods and their respective hyperparameters results in a large search space. Therefore methods that alleviate runtime are explored. Multiobjective optimization aims to reach a trade-off when selecting configurations. In this case between accuracy and runtime. Sampling optimizes and selects configurations on only a subset of the data. This is similarly a trade-off, as while it improves runtime it worsens accuracy.

# 5 Prototypical Implementation

This chapter covers the prototypical addition of preprocessing methods to the package AM4C dubbed Automatic Machine Learning for Preprocessing and Clustering (AM4PC). This includes relevant Python libraries, an overview of the extended package structure, and additional added functionality.

## 5.1 External Python Packages

The core functionality of AM4PC and the additional preprocessing functionality relies on three packages: scikit-learn, ConfigSpace and SMAC3. Further packages and version numbers can be found in the GitLab repository[1].

Scikit-learn[2] [PVG+11] is a Python package that offers a wide range of modern ML algorithms for both supervised and unsupervised learning. These are provided through high-level interfaces that focus on ease of use, API consistency, and detailed documentation. This consistency is useful since it allows a modular implementation of both clustering algorithms and preprocessing methods. Scikit-learn provides the majority of preprocessing methods, clustering algorithms, and CVMs (Cluster Validation Measures).

The clustering algorithms and preprocessing methods with their respective hyperparameters are combined in a search space with ConfigSpace[3] [LEF+19].
This package supports all the standard types of hyperparameters such as categorical, ordinal, integer-valued, and continuous. Further, these hyperparameters can be combined with conditional constraints, so that only relevant hyperparameters are used depending on the selected clustering algorithm. For example, the number of cluster centers is only considered if the chosen algorithm is K-Means and not when it is DB-SCAN.

Similarly to auto-sklearn, SMAC3[4] [LEF+22] is used to determine an optimal configuration from the search space via HPO. Configurations specify a preprocessing method, clustering algorithm, and their relevant hyperparameters. This package provides a widely used implementation of BO (see Section 2.1.1). SMAC3 implements multiple BO approaches in the form of facades with a simple API that hides complexity. Especially the SMAC4HPO facade is relevant, as it is specifically tailored to solve the CASH problem. When compared to other libraries it performs well quality-wise. Therefore, this package enables the efficient search of the search space defined with ConfigSpace.

[1] https://gitlab-as.informatik.uni-stuttgart.de/reedcr/AM4PC/-/blob/Dev/Package/requirements.txt
[2] https://scikit-learn.org/stable/
[3] https://automl.github.io/ConfigSpace/main/
[4] https://automl.github.io/SMAC3/v1.4.0/index.html

## 5.2 General Overview of the Prototype

The prototypical package dubbed Automatic Machine Learning for Preprocessing and Clustering (AM4PC) contains five important sub-packages. These are listed below and shown in Figure 5.1 before additional detail is provided in the following sections.

- **Benchmarks**: This sub-package contains the framework for managing and loading benchmark datasets. The benchmark datasets are described in Section 6.1.1.
- **Cluster Validation Measures**: This sub-package is responsible for handling CVMs and contains seven internal and six external measures.
- **Optimizer**: This sub-package wraps SMAC's BO which includes multiple generic optimizers that can be used interchangeably. These are further adapted to the preprocessing and clustering optimization problem. The main optimizer that is used is SMAC4HPO.
- **Preprocessing**: This sub-package contains two classes. A *Dataformater* and a *DataPreprocessor*. The *DataFormater* contains preprocessing methods and steps that are always applied when relevant.
  The *DataPreprocessor* contains eight feature transformation and six feature extraction methods, of which the optimizer can only select one at a time, with their relevant hyperparameters.
- **Clustering**: This sub-package contains nine clustering algorithms that can be applied on numerical data and two that can be applied on categorical data. For all of the algorithms, their relevant hyperparameters are implemented as well. Similarly to preprocessing methods only one can be applied per configuration.

Everything is combined in the *AM4PC* class, which offers a simple and concise API.

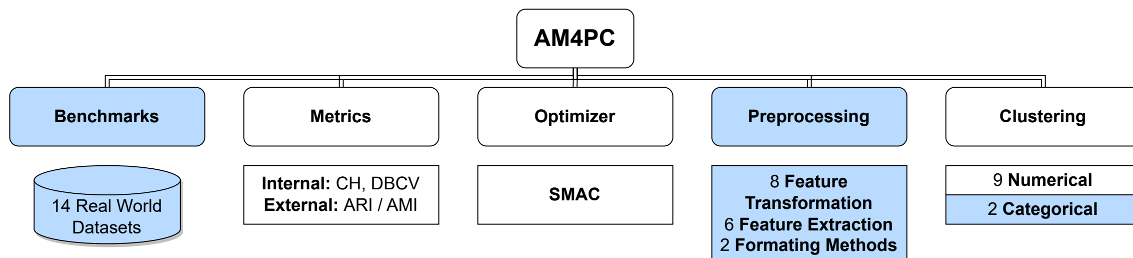


**Figure 5.1:** AM4PC package overview where blue markings indicate additions compared to AM4C.

### 5.2.1 Benchmarks

The *Benchmarks* class coupled with the *BenchmarksMetadata.json*[5] file provides two main functionalities. The first is *get_list_of_benchmarks* which simply returns an overview of all available benchmark datasets names that are further described in Section 6.1.1. These names can be used in

[5]https://gitlab-as.informatik.uni-stuttgart.de/reedcr/am4cp/-/blob/Dev/Benchmarks/Real/BenchmarksMetadata.json

combination with the second method *load* which returns the feature data and if present, the label data. The feature data includes all features (variables) that describe a datapoint while the label data contains the ground truth cluster assignment if available.

### 5.2.2 Cluster Validation Measures

The main functionality of this sub-package is contained in two classes.

The *Metric* class wraps the CVMs to provide a consistent way of scoring each CVM as well as handling common errors.

The *MetricCollection* class groups all CVMs and defines them as *Metrics* with a name, an internal or external flag, and if they are minimized or maximized. Out of the 13 implemented validation measures four are especially promising. The two external validation measures (see Section 2.3), ARI and AMI, are implemented using scikit-learn. Whereas the internal validation measure (see Section 2.3) CH is implemented using scikit-learn while DBCV is implemented using hdbscan[6]. Internal measures (metrics) are always evaluated on the standard scaled feature data due to results described in Section 6.2.4.

### 5.2.3 Optimizer

The *SMACOptimizer* class contains the core functionality of this sub-package. This includes defining the target function to be optimized, the optimization scenario, and providing methods to extract the run results which contain tried configurations and their score.

The target function takes the configuration to execute and a measure as input. The configuration is executed by first applying the preprocessing method, if present in the configuration, to the data with the defined hyperparameters. Then the data is clustered using the defined clustering algorithm and respective hyperparameters before the labels are returned. The resulting labels are then evaluated depending on the chosen CVM (metric). If an external measure is chosen the returned labels are compared to the ground truth labels and scored appropriately. Else the clusters are evaluated upon their internal structure upon criteria such as compactness and separation between clusters (see Section 2.3).

Additionally, a scenario is defined that specifies optimizer settings such as:

- The run objective, which can either be the value returned by the target function (here accuracy) or runtime.
- The search space which includes the methods, algorithms, and hyperparameters shown in Table 5.1 and Table 5.2 from which configurations can be sampled.
- The number of optimizer loops to execute before returning the best configuration with respect to the run objective.

[6]https://hdbscan.readthedocs.io/en/latest/index.html

Lastly, methods to parse run logs into both pandas[7] dataframes and Python dictionaries for further analysis are implemented.

### 5.2.4 Preprocessing

The preprocessing sub-package contains two classes, *DataFormater* and *DataPreprocessor*.

The *DataFormater* is responsible for transforming the data to a consistent format so that further operations on the data can be performed uniformly. The main functionality of this class lies in the *format* method.
When called, the data types such as float or string are automatically detected and set. Then ground truth label and feature sets are created as Dataframes[8] by selecting the appropriate columns. For each Dataframe their respective categorical columns are either One-Hot-encoded or label encoded. One-Hot encoding is preferred as it is a more natural way of depicting categorical data in space, but label encoding is sometimes also necessary to avoid creating too many new features or because of subsequent categorical clustering. One-Hot encoding is therefore always chosen when the number of unique values in a categorical column is less than 10 unless categorical clustering is selected. Lastly columns with too many missing values, per default 40%, are dropped while the remaining missing values are imputed with KNN (see Section 4.2.1).

In the *DataPreprocessor* class the 14 preprocessing methods (see Section 4.2.2) are consistently defined as functions that wrap scikit-learn methods. These take the relevant hyperparameters as inputs before applying *fit_transform* and returning the result as seen in Listing 5.1.

```
1  def fast_ica(self, n_components: int = None, fun : str = "logcosh") -> None:
2      return FastICA(n_components=n_components, fun=fun, tol = 0.01).fit_transform(self.X.copy())
```

**Listing 5.1:** Example of a preprocessing method definition in Python.

The aforementioned relevant hyperparameters are defined using the ConfigSpace library and include categorical, integer, and float variables. These are defined with a name, a range of values or choices, and a default value as seen in Listing 5.2.

```
1  fun = CategoricalHyperparameter(name = "fun", choices = ["logcosh", "exp", "cube"], default_value = "
logcosh")
```

**Listing 5.2:** Example of categorical hyperparamter definition in Python.

Depending on the dataset some hyperparameter ranges are adjusted, since for example *n_components* must be smaller than the number of features.

Lastly, the search space is built by calling *build_config_space* which contains options that allow the user to specify which preprocessing methods should be considered. The chosen preprocessing options as well as their relevant hyperparameters are added to the search space and combined with conditions so that the hyperparameters are only sampled when the particular preprocessing method is chosen as seen in Listing 5.3.

[7] https://pandas.pydata.org/
[8] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

```python
updated_cs = ConfigurationSpace()

preprocessing_options = CategoricalHyperparameter(name = "preprocessing_options", choices = ["pca", "fast_ica", ...])

updated_cs.add_hyperparameter(preprocessing_options)
updated_cs.add_hyperparameter(fun)

updated_cs.add_condition(InCondition(fun, preprocessing_options, ["fast_ica"]))
```

**Listing 5.3:** ConfigSpace conditions in Python.

When all preprocessing methods are chosen, which is the usual case, the whole search space could look like Table 5.1.

| Preprocessing Methods | Relevant Hyperparameters | Hyperparameters | Type | Options / Ranges |
|---|---|---|---|---|
| Standard Scaler | with_std | with_std | Categorical | [True, False] |
| MinMax Scaler | - | cutoff | Float | [0.01, 0.45] |
| MaxAbs Scaler | - | ratio_quantiles | Float | [0.01, 1.0] |
| Robust Scaler | cutoff, with_std | output_distribution | Categorical | [uniform, normal] |
| Log Transformer | - | n_components | Integer | [1, 30] |
| Power Transformer | - | fun | Categorical | [logcosh, exp, cube] |
| Binarizer | - | learning_decay | Float | [0.5, 1.0] |
| Quantile Transformer | ratio_quantiles, output_distribution | learning_offset | Integer | [1, 100] |
| PCA | n_components | degree | Integer | [2,3] |
| Fast-ICA | n_components, fun | | | |
| Factor-Analysis | n_components | | | |
| LDA | n_components, learning_decay, learning_offset | | | |
| Truncated-SVD | n_components | | | |
| Poly Features | degree | | | |
| No Transformation | - | | | |

**Table 5.1:** Example CASH search space for preprocessing methods.

### 5.2.5 Clustering

The core clustering functionality lies in the *ClusteringAlgorithms* class. This class is structurally equivalent to the *DataPreprocessor* class. Its 11 clustering algorithms are defined as modular functions and mainly rely on scikit-learn with the exception of K-Modes and K-Prototypes. These originate from the kmodes package[9]. The functions take the relevant hyperparameters as inputs before applying *fit_predict* and returning the result as seen in Listing 5.4.

```python
def kmeans(self, n_clusters : int = 10) -> np.array:
        return KMeans(n_clusters = n_clusters, n_init=self.n_init, max_iter = self.max_iter).fit_predict(self.X)
```

**Listing 5.4:** Example of a clustering method definition in Python.

[9]https://pypi.org/project/kmodes/

The relevant hyperparameters are defined in the exact same manner as in the *DataPreprocessor* class. The creation of the search space is similarly identical, though algorithms and hyperparameters differ as shown in Table 5.2.

| Clustering Algorithms | Relevant Hyperparameters | Hyperparameters | Type | Options / Ranges |
|---|---|---|---|---|
| K-Means | n_clusters | n_clusters | Integer | [2, 30] |
| Mini-Batch-K-Means | n_clusters | eps | Float | [0.1, 1.0] |
| Agglomerative Clustering | n_clusters | min_samples | Integer | [2, 2110] |
| Birch | n_clusters | quantile | Float | [0.1, 0.9] |
| Spectral | n_clusters | damping | Float | [0.6, 0.9] |
| GMM | n_clusters | | | |
| DB-SCAN | eps, min_samples | | | |
| Mean-Shift | quantile | | | |
| Affinity Propagation | damping | | | |
| K-Modes | n_clusters | | | |
| K-Prototypes | n_clusters | | | |

**Table 5.2:** Example cash SearchSpace for clustering algorithms

### 5.2.6 AM4PC

This class combines all previous sub-packages to provide a simple high-level API.
To initialize the class, the feature data and if present the label data are passed. The main functionality of the class lies in the *cluster* method. This method has many parameters including ones that specify preprocessing methods and clustering algorithms that should be applied, which CVM to use, and the budget for optimizer loops.
The *cluster* function begins by combining the search space of both preprocessing methods and clustering algorithms provided by the *DataPreprocessor* and *ClusteringAlgorithms* classes respectively. Then the optimizer is defined via the *SMACOptimizer* class and run. It returns results containing the best configuration.
A minimal use-case is shown in Listing 5.5 while the full workflow is shown in Figure 5.2.

```
from sklearn.datasets import make_blobs
from AM4PC import AM4PC


# Create a synthetic data set
X, y = make_blobs(n_samples=1000, n_features=2, centers= 10)
X, y = _Data_Formater(X,y).format()

AM4PC = AM4PC(X=X,y=y)
AM4PC.cluster(
    preprocessing_algorithms = "SCALING",
    clustering_algorithms = "PARTITIONAL",
    metric_str = "CH", n_loops  = 20)

# print run history for further analysis
history = AM4PC.Optimizer.get_run_history()
print(history)
```

**Listing 5.5:** Minimal AM4PC Python example.

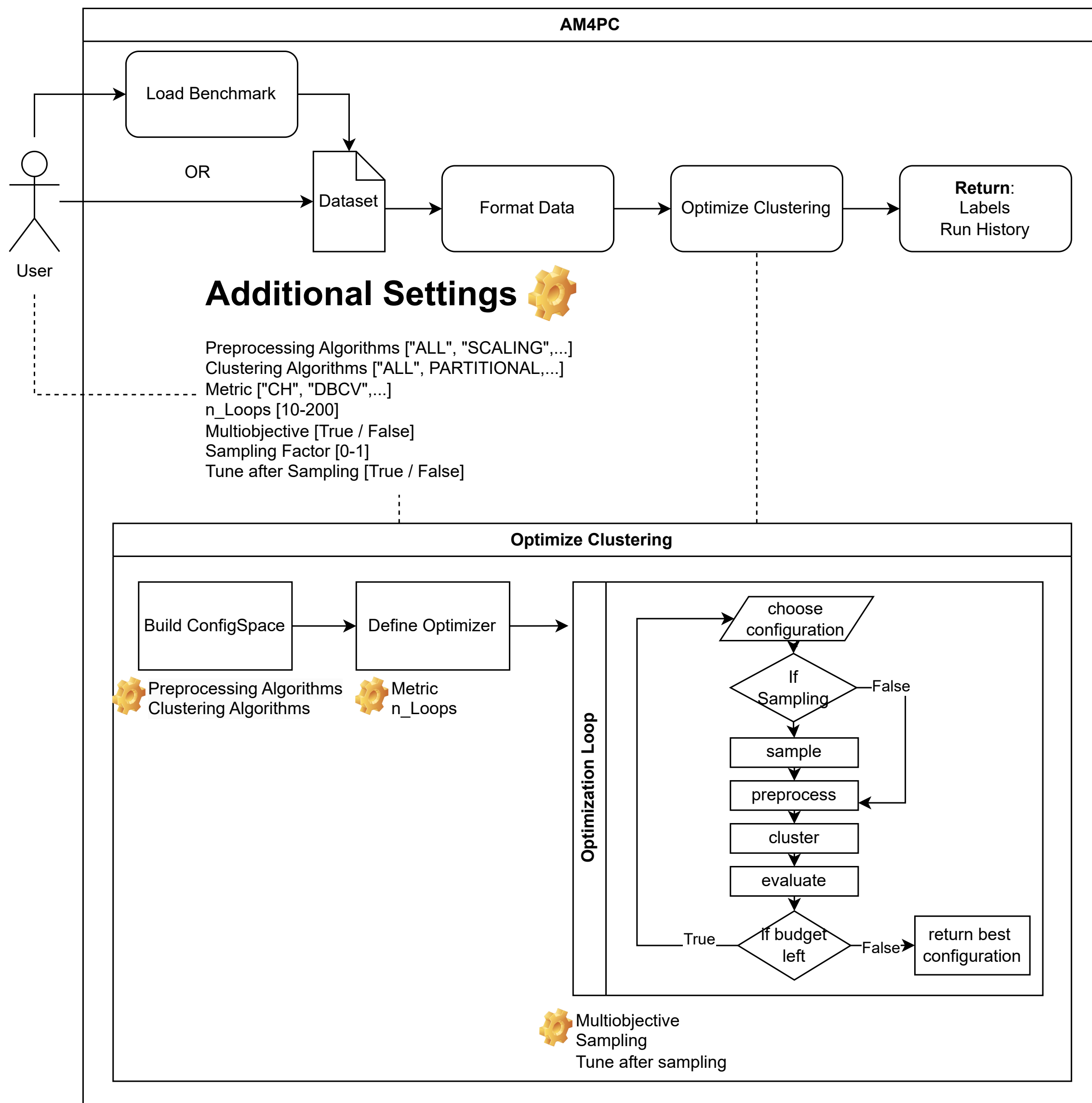


**Figure 5.2:** AM4PC full flowchart with additional settings.

## 5.3 Additional Functionality

A major design factor while implementing new functionality in AM4C is keeping it extensible and modular. This is achieved as benchmarks, preprocessing methods, clustering algorithms, metrics, etc. are all easily extensible. To add a new benchmark one only needs to create an additional entry in *BenchmarksMetadata.json* and add the .csv file to the appropriate folder. Preprocessing methods and clustering algorithms need to be defined as a function and their respective hyperparameters added to the search space. Metrics (CVMs) can be extended by defining a new *Metric* object with the correct parameters.

This extensibility is used to add three major new features. One that aims to improve accuracy by adding categorical clustering algorithms. Two that aim to improve runtime through multiobjective optimization and sampling.

### 5.3.1 Categorical clustering algorithms

Most datasets include a mix of numerical and categorical features. As the majority of clustering algorithms can only handle numerical data, these are usually transformed with One-Hot encoding as described in Section 4.2.2. While better than label encoding, this method still does not completely capture categorical features as numerical features. Additionally, it leads to a massive increase in features and therefore runtime.
As categorical clustering algorithms were developed to solve this problem, two clustering algorithms K-Modes and K-Prototypes are added to AM4PC. When these algorithms are executed, categorical columns are label encoded instead of One-Hot-encoded. Further no preprocessing methods are applied, as these would distort the categories. These categorical clustering algorithms are only added to the search space if at least one categorical column is present in the feature dataset.

### 5.3.2 Multiobjective Optimization

As the name suggests, multiobjective optimizations allows the optimizer to pursue multiple different goals by reaching a trade-off. Since the addition of preprocessing increases runtimes, this feature is used to reach a trade-off between the CVM and runtime. Multiobjective optimization is inherently offered by SMAC3[10] and uses ParEGO [CK15] to combine the objectives. This is achieved by ParEGO normalizing each objective to a range of $[0, 1]$ by using an estimated upper and lower bound. These bounds are mainly based on values from prior runs [Kno06]. Finally, a single score is calculated by taking the mean of the normalized objective values and used for optimization.

This only requires a few additional lines when defining the optimizer as seen in Listing 5.6:

```
scenario_dict = {
        ...
        "multi_objectives": ["metric", "runtime"],
        ...
        }

self.smac_algo(
        ...
        multi_objective_algorithm=ParEGO,
        multi_objective_kwargs={"rho": 0.05},
        ...
        )
```

**Listing 5.6:** SMAC3 multiobjective Python example.

[10]https://automl.github.io/SMAC3/v1.4.0/details/multi_objective.html

### 5.3.3 Sampling

Data reduction can be a necessary step when dealing with large datasets and high runtimes as described in Section 4.1.2. Therefore the option to optimize on a subset of the data, selected by random sampling, is implemented.
The results on the sample do not always transfer to the whole dataset. Therefore an approach inspired by Hyperband (see Section 2.1.1) is implemented. Here the hyperparameters of the best prior preprocessing method and clustering algorithm are fine-tuned on the whole dataset. This fine-tuning happens during the last 10% of the optimizer loops.

# 6 Evaluation

This chapter evaluates the prototype AM4PC described in Chapter 5. First, the evaluation setup is explained in Section 6.1. Then results in regard to both accuracy and runtime stemming from the new additions to AM4C are evaluated in Section 6.2 and Section 6.3. Finally, a detailed analysis of preprocessing and clustering algorithms is presented and discussed in Section 6.4.

## 6.1 Evaluation Setup

This section describes the conditions under which the prototype (see Chapter 5) is evaluated. It consists of the choice of benchmarks, hardware, and evaluation scenarios.

### 6.1.1 Benchmarks

It is difficult to find established benchmarking datasets for clustering. Literature usually uses a mix of synthetic and real-world datasets [LW21; TFS21].

Synthetic benchmarks have the advantage of configurability, ease-of-use, and ground truth labels. Scikit-learn provides multiple methods[1] such as *make_blobs*, *make_circles*, and *make_moons* to create synthetic datasets as seen in Figure 6.1

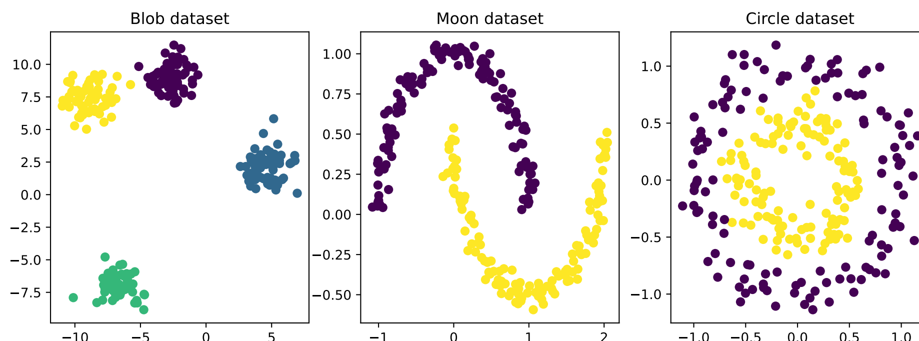


**Figure 6.1:** Synthetic datasets generated with scitkit-learn.

Similarly to [TFS21], these are used to create a synthetic benchmark suite with different attributes. The specified attributes are the number of datapoints $n$ which ranges from 100 to 1000, the number of features $f$ which ranges from 2 to 50, the number of clusters $k$ which ranges from 2 to 50, and the

[1] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets

standard deviation *std* which ranges from 0 to 2.
Synthetic datasets can not completely mirror real-world datasets in all aspects such as multi-class imbalance and heterogeneous groups [TRSM23]. These are the areas where preprocessing could have the most impact. This theory is proven by the poor results on synthetic datasets discussed in Section 6.2.1. Therefore, mainly real-world datasets are used to evaluate the potential benefit of preprocessing.

Previous works on AutoML for clustering [ELS21; TFS21] use the UCI machine learning repository[2] for benchmarking data, as it offers a wide range of datasets. They choose datasets with ground truth labels so that external CVMs can be used. This work proceeds similarly. In total 14 datasets are chosen, with the goals of variety in number of datapoints $n$, number of features $f$, number of categorical features $c$, number of clusters $k$, and the average $std$ across all features, as shown in Table 6.1.

| **Dataset** | $n$ | $f$ | $c$ | $k$ | average $std$ |
|---|---|---|---|---|---|
| *rna*[3] | 801 | 20531 | 477 | 5 | 1.11 |
| *frogs*[4] | 7195 | 23 | 0 | 4 | 0.71 |
| *iris*[5] | 150 | 4 | 0 | 3 | 0.94 |
| *obesity*[6] | 2111 | 31 | 8 | 7 | 1.3 |
| *glass*[7] | 214 | 9 | 0 | 6 | 0.69 |
| *ecoli*[8] | 336 | 8 | 2 | 8 | 12.25 |
| *ionosphere*[9] | 351 | 34 | 2 | 2 | 0.51 |
| *tae*[10] | 151 | 5 | 2 | 3 | 5.48 |
| *haberman*[11] | 306 | 3 | 0 | 2 | 7.07 |
| *segment*[12] | 2310 | 20 | 3 | 7 | 57.39 |
| *sonar*[13] | 208 | 60 | 0 | 2 | 0.14 |
| *car*[14] | 1728 | 21 | 6 | 4 | 0.3 |
| *contraceptive*[15] | 1473 | 9 | 7 | 3 | 1.7 |
| *diagnosis*[16] | 120 | 13 | 6 | 2 | 0.36 |
| **range** | [120, 7195] | [3, 20531] | [0, 477] | [2, 8] | [0.14, 57.39] |

**Table 6.1:** Meta-Features of real-world benchmark suite.

[2] https://archive.ics.uci.edu/ml/index.php
[3] https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq
[4] https://archive.ics.uci.edu/ml/datasets/Anuran+Calls+%28MFCCs%29#
[5] https://archive.ics.uci.edu/ml/datasets/iris
[6] https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition+
[7] https://archive.ics.uci.edu/ml/datasets/glass+identification
[8] https://archive.ics.uci.edu/ml/datasets/ecoli
[9] https://archive.ics.uci.edu/ml/datasets/ionosphere
[10] https://archive.ics.uci.edu/ml/datasets/teaching+assistant+evaluation
[11] https://archive.ics.uci.edu/ml/datasets/haberman%27s+survival
[12] https://archive.ics.uci.edu/ml/datasets/Statlog+%28Image+Segmentation%29
[13] http://archive.ics.uci.edu/ml/datasets/connectionist+bench+(sonar,+mines+vs.+rocks)
[14] https://archive.ics.uci.edu/ml/datasets/Car+Evaluation
[15] https://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice"

### 6.1.2 Hardware

The evaluation is conducted on a VM running Ubuntu 22.04.1 LTS with 8 AMD EPYC-Milan, 2450 MHz CPUs, and 16 GB of system memory.

### 6.1.3 Evaluation Scenarios

Evaluation scenarios specify the conditions under which the benchmark suite is optimized. The conditions dictate on which datasets the optimization occurs, which CVM is used and which preprocessing methods, as well as clustering algorithms, are applied. Further conditions include multiobjective optimization and sampling.

In total 15 evaluation scenarios are chosen, depicted in Table A.1, which are run at least 5 times. In total this results in a runtime of about 50.4 days per CPU. Hereby evaluation scenarios named AM4C do not include preprocessing while ones named AM4PC do. In each evaluation run each dataset of the selected benchmark suite is clustered with a budget of 100 optimization loops. The budget of 100 optimization loops is sufficient as shown in Figure 6.2, since most improvement happens in the first 60 optimizer loops as further validated in [TFS21] and after 90 optimization loops further improvements are minimal.

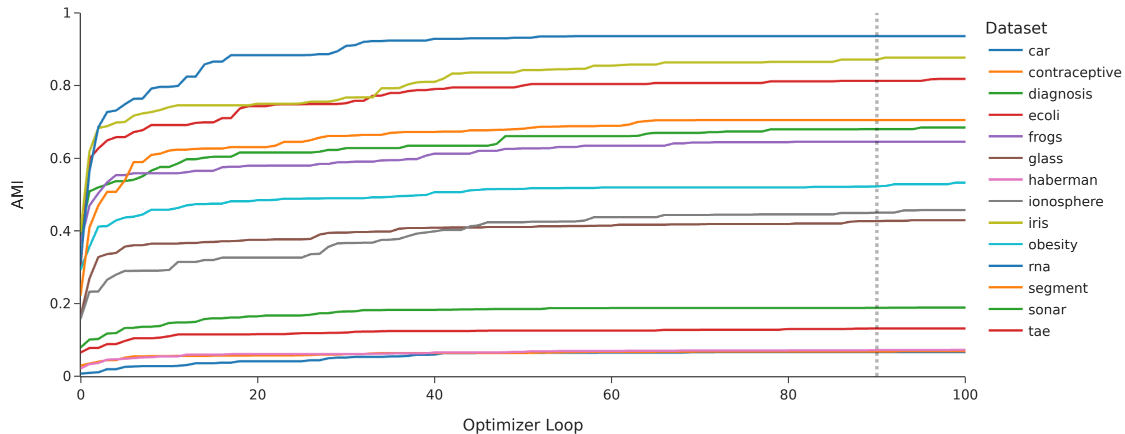


**Figure 6.2:** Best mean AMI up to optimizer loop calculated for the evaluation scenario *AM4PC real*.

Multiple evaluation runs are necessary to combat the randomness incurred through BO which can only guarantee a local maximum and not a global one. The number of evaluation scenario runs is sufficient in regards to accuracy (AMI), as the highest standard deviation for an evaluation scenario is only 0.06. Runtime is more volatile with the standard deviation ranging up to nearly 13 seconds for the evaluation scenario *AM4PC*, which is very high considering the mean runtime is only 1.9 seconds for that scenario (for runtimes see Table 6.7. These runtime issues are addressed and improved upon in Section 6.3.

[16]https://archive.ics.uci.edu/ml/datasets/Acute+Inflammations

## 6.2 Accuracy

Due to the uncertainty of internal validation measures, the external measure AMI (Adjusted Mutual Index) is used to avoid false conclusions. This method is comparable to the one used in [TFS21]. Other external CVMs such as ARI (see Section 2.3) exist, but due to a correlation of 0.912 between the two CVMs, it is safe to assume the use of AMI achieves largely comparable results. The correlation is hereby calculated over 20 evaluation runs totaling 28.000 AMI and ARI values.

In this work the Pearson correlation coefficient [BCHC09] is used to calculate correlation:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}.$$

Here $x$ and $y$ represent the variables between which the linear relationship is calculated. The Pearson correlation coefficient measures the degree to which the two variables are related to each other as well as the direction and strength of that relationship.

In the following, it is determined if preprocessing can improve accuracy on synthetic datasets in Section 6.2.1 and real-world datasets in Section 6.2.2. Then due to the significance of categorical data, additional categorical clustering algorithms are added to the search space and evaluated in Section 6.2.3. Finally, the applicability to real-world scenarios is measured by using internal CVMs in Section 6.2.4.

### 6.2.1 Synthetic Datasets

The addition of preprocessing methods to the search space (AM4PC), when evaluating the synthetic datasets described in Section 6.1.1, leads to a drop in mean AMI of 2.05% from 0.831 to 0.814. Figure 6.3 displays the mean AMI for different types of synthetic datasets calculated over all winning configurations for both AM4C and AM4PC.

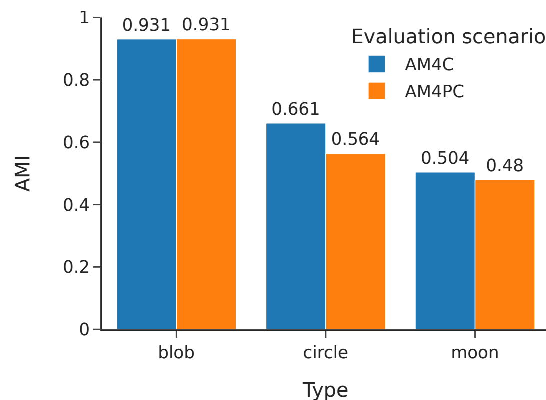


**Figure 6.3:** Best configurations mean AMI scores for different synthetic benchmark types. These are calculated for AM4C and AM4PC over five evaluation scenario runs.

The addition of preprocessing mainly affects *circle* datasets and to a lesser degree *moon* datasets. This is most likely due to the data being explicitly generated for clustering and not possessing any complex characteristics that preprocessing could improve. This means that preprocessing only unnecessarily expands the search space and consequently leads to worse accuracy. The nature of the *blob* datasets makes them easier to cluster compared to the other two types, therefore its accuracy is not significantly affected by the increased size of the search space.

### 6.2.2 Real World Datasets

The real-world datasets (see Section 6.1.1) are evaluated on both AM4C and AM4PC. The results are shown in Table 6.2.
In 12 out of the 14 datasets, the addition of preprocessing (AM4PC) leads to improved accuracy with a mean AMI increase per dataset of 24.6% . Especially the clustering result on the *car* datasets is improved, as the AMI nearly triples. These results on the *car* dataset are mainly achieved through the addition of a *Power Transformer*, *Standard Scaler*, or *Factor Analysis*. Only two out of the 14 datasets are negatively impacted by the addition of preprocessing.
Over all benchmarks a mean AMI of 0.444 with AM4C and 0.472 with AM4PC is achieved. This is an increase of 6.4%.

| Dataset | AM4C | AM4PC | Percentage increase |
|---|---|---|---|
| *car* | 0.021 | 0.067 | **213.358** |
| *contraceptive* | 0.050 | 0.069 | **39.686** |
| *diagnosis* | 0.664 | 0.684 | **3.046** |
| *ecoli* | 0.760 | 0.818 | **7.539** |
| *frogs* | 0.641 | 0.646 | **0.633** |
| *glass* | 0.434 | 0.429 | -1.031 |
| *haberman* | 0.072 | 0.073 | **0.387** |
| *ionosphere* | 0.387 | 0.457 | **18.241** |
| *iris* | 0.792 | 0.877 | **10.686** |
| *obesity* | 0.521 | 0.533 | **2.322** |
| *rna* | 0.981 | 0.935 | -4.608 |
| *segment* | 0.626 | 0.705 | **12.598** |
| *sonar* | 0.158 | 0.189 | **19.702** |
| *tae* | 0.108 | 0.132 | **21.591** |
| **MEAN** | 0.444 | 0.472 | **24.582** |

**Table 6.2:** AM4C compared to AM4PC evaluated on real-world datasets mean AMI across all evaluation runs. Improvements are marked in bold.

Therefore, the addition of preprocessing methods leads to an increased accuracy when using an external CVM and real-world datasets on most datasets. As synthetic datasets saw no improvement through preprocessing, all further evaluations are done on real-world datasets.

### 6.2.3 Adding Categorical Clustering Algorithms

Most clustering algorithms can not deal with categorical data, which is prevalent in real-world datasets (see Table 6.1). Consequently is reasonable to assume that the addition of categorical clustering algorithms to the search space could improve accuracy.

Therefore, *K-Modes* and *K-Prototypes* (see Section 2.2) are added to the search space in addition to the previous algorithms. These are mostly treated identically to a standard algorithm with two exceptions: First, when configurations that include a categorical clustering algorithm are selected by the optimizer, categorical data does not need to be One-Hot-encoded. The categories are instead mapped to numerical values with label encoding. Second, preprocessing methods are not applied, since most feature engineering methods such as scaling would not have an effect on categorical data. As most datasets are a mix of categorical and numerical features, one could analyze the effect of still preprocessing numerical features, as prior results have proven the possible benefit of preprocessing (see Section 6.2.4). This would require extensive modification of the existing system and is therefore referred to future work.

The full evaluation conditions of *AM4PC + Categorical* include preprocessing and categorical clustering algorithms (as shown in Table A.1). The addition of categorical clustering algorithms leads to a degradation in accuracy of roughly 0.6% when compared to *AM4PC*.
This can be attributed to both K-Modes and K-Prototypes performing very poorly as discussed in Section 6.4. This poor performance is in part likely due to datasets having significantly more numerical features than categorical features.
Neither clustering algorithm is present in any of the winning configurations, which include results from 5 evaluation runs over 14 datasets totaling 70 winning configuration. Consequently, they purely increase the size of the search space which leads to a decrease in accuracy. The viability of these algorithms also suffers due to their extensive runtimes as further discussed in Section 6.4.

K-Modes and K-Prototypes are a modified version of K-Means, which while performing decently is outclassed by algorithms such as GMM (see Section 6.4). Therefore a comparison between just K-Means and the categorical algorithms might be more representative. Preliminary results suggest that even just K-Means with preprocessing outperforms or matches the categorical clustering algorithms in accuracy on all benchmarking datasets.
Results therefore indicate that converting categorical columns to numerical values with One-Hot encoding is a viable alternative to clustering algorithms capable of dealing with categorical data.

### 6.2.4 Internal CVMs

To measure the transferability to real-world applications, where ground truth labels are unknown, the internal CVMs CH and DBCV are used to compare results between AM4C and AM4PC.

Preprocessing modifies the feature dataset which impacts the calculation of the internal measure. Therefore three methods are tested and evaluated, calculation on the original feature data, on the standard scaled feature data, and on the feature data after preprocessing. This is done by optimizing datasets using the different methods, while additionally calculating the external CVM AMI.
The results, as shown in Table 6.3, indicate that validation on either the original or standard scaled

feature data is preferable to the preprocessed feature data. This seems to be because the optimization on a CVM applied to the preprocessed feature data artificially favors dimensions reducing mapping due to them positively influencing CH.

| | AMI | ARI |
|---|---|---|
| **CH preprocessed feature data** | -0.11 | -0.10 |
| **CH standard scaled feature data** | 0.59 | 0.48 |
| **CH original feature data** | 0.58 | 0.47 |

**Table 6.3:** Pearson correlation between CH on differently preprocessed datasets compared and AMI / ARI.

As these preliminary results are calculated on datasets with a similar distribution in all dimensions, it is reasonable to assume standard scaling could lead to an even bigger improvement when dimensional spread differs. Therefore in AM4PC internal measures are always evaluated on the standard scaled feature data.

In Table 6.4 the mean internal CVM and AMI for four different evaluation scenarios are shown. These are *AM4C on CH*, *AM4PC on CH*, *AM4C on DBCV*, and *AM4PC on DBCV* that are fully described in Table A.1. Additionally, the mean AMI of the winning configurations over all evaluation runs is shown.

When optimizing on CH, AM4PC achieves a mean accuracy of 495.6 which is higher than AM4C at 462.1. The AMI calculated on the winning configurations remains similar when comparing AM4C and AM4PC. When using DBCV to optimize, the mean DBCV increases from 0.024 to 0.028 while the mean AMI increases from 0.247 to 0.269 which is 8.9%.

| | AM4C on CH | AM4PC on CH | AM4C on DBCV | AM4PC on DBCV |
|---|---|---|---|---|
| **Internal CVM** | 462.132 | 495.569 | 0.024 | 0.028 |
| **AMI** | 0.282 | 0.280 | 0.247 | 0.269 |

**Table 6.4:** AM4C versus AM4PC optimized on different internal CVMs.

It is important to note that the mean AMI scores 0.269 and 0.280 for AM4PC when optimizing on CH and DBCV are respectively 40.7% and 43.0% worse than when directly optimizing on AMI. This is likely due to the low correlation between the internal CVMs and the external CVMs shown in Table 6.5. Further analysis indicates that the correlation is mainly dataset dependant, as the correlation between CH and AMI on the *iris* dataset is 0.81 while the worst correlation is on the *contraceptive* dataset at $-0.54$. For DBCV the highest correlation is similarly on the *iris* dataset at 0.63 and worst on the *contraceptive* dataset at $-0.38$. Validating internal CVMs with external ones is in general problematic as described in Section 2.3.

| | AMI | ARI |
|---|---|---|
| **CH** | 0.20 | 0.13 |
| **DBCV** | 0.23 | 0.20 |

**Table 6.5:** Pearson correlation between internal and external CVMs on real-world datasets

### 6.2.5 Summary

Preprocessing did not improve accuracy on synthetic datasets generated specifically for clustering, as the data quality was already high for clustering.
On real-world data, preprocessing did improve clustering results by a mean of 24.6% per dataset and 6.4% when the AMI mean is calculated over all datasets and compared as shown in Figure 6.4.
Adding two clustering algorithms capable of dealing with categorical data worsened the accuracy slightly as shown in Table 6.2.
Lastly, the improvement won through the addition of preprocessing seems to be transferable to real-world applications, though it is important to consider which internal CVM to optimize on.

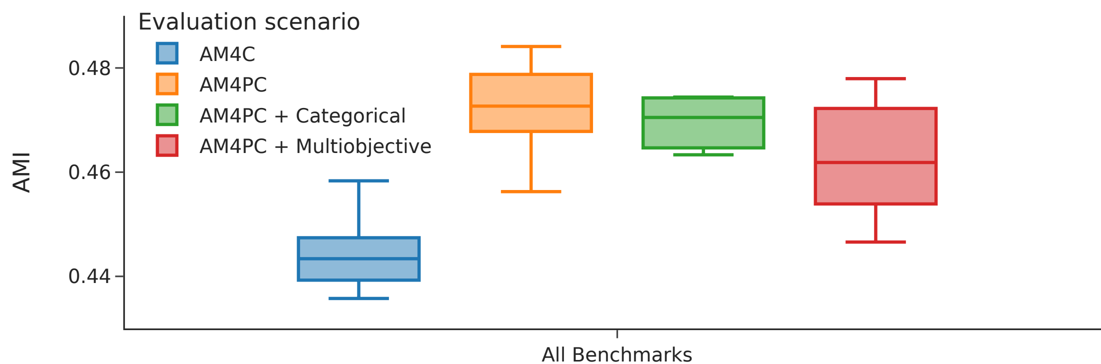


**Figure 6.4:** The four highest accuracy evaluation scenarios on real-world data with 5-10 evaluation runs each.

## 6.3 Runtime

In Table 6.6 the mean runtimes of an optimization loop per dataset in AM4C and AM4PC are shown. Although the addition of preprocessing to the search space increases accuracy, it also increases runtime.
This massive increase from 3.5 seconds per optimization loop to 295.3 seconds, or mean 2584% increase per dataset is largely due to two datasets *rna* and *frogs*. These both have a high number of features or datapoints as seen in Table 6.1. Since most preprocessing methods and clustering algorithms are not linear in runtime but rather $O(n \log n)$ or $O(n^2)$ this leads to exponential increases in runtime. Some preprocessing methods such as *Poly Features* also increase the number of features and therefore the runtime of the subsequent clustering.

| Dataset | AM4C | AM4PC | Percentage increase |
|---|---|---|---|
| *car* | 0.656 | 3.090 | 371.2 |
| *contraceptive* | 1.142 | 1.682 | 47.3 |
| *diagnosis* | 0.042 | 0.060 | 43.0 |
| *ecoli* | 0.110 | 0.291 | 164.6 |
| *frogs* | 10.506 | 3132.279 | 29715.0 |
| *glass* | 0.062 | 0.117 | 89.8 |
| *haberman* | 0.272 | 0.367 | 34.9 |
| *ionosphere* | 0.116 | 0.302 | 160.8 |
| *iris* | 0.051 | 0.226 | 339.7 |
| *obesity* | 13.997 | 10.616 | -24.2 |
| *rna* | 20.390 | 979.306 | 4703.0 |
| *segment* | 1.400 | 5.823 | 316.0 |
| *sonar* | 0.052 | 0.163 | 215.7 |
| *tae* | 0.129 | 0.132 | 2.6 |
| **MEAN** | 3.495 | 295.318 | 2584.2 |

**Table 6.6:** AM4C vs. AM4PC mean runtime per optimization loop in seconds calculated on 10 evaluation scenario runs.

When excluding the outliers, the increase in mean runtime per dataset between AM4C and AM4PC is 146.793%. Further runtime analyses will therefore disregard these two datasets, as they are too volatile and difficult to measure consistently.

To combat the increased runtime two different strategies are analyzed: Multiobjective optimization (see Section 6.3.1), which aims to reach an optimal trade-off between both the CVM and runtime. Sampling (see Section 6.3.2), which uses only part of the data during optimization.

### 6.3.1 Multiobjective Optimization

The main benefit of multiobjective optimization is how small the trade-off is in regards to accuracy, as the AMI only drops by 1.9%. The runtimes for the *AM4PC* (single objective) and *AM4PC + Multiobjective* evaluation scenarios are shown in Table 6.7. In 11 out of 12 datasets the runtime decreases. Over all datasets, runtimes decrease by a mean 34.7% with reductions of up to 58%.

| **Dataset** | **AM4PC** | **AM4PC + Multiobjective** | **Percentage decrease** |
|---|---|---|---|
| *car* | 3.090 | 1.299 | **57.949** |
| *contraceptive* | 1.682 | 0.952 | **43.388** |
| *diagnosis* | 0.060 | 0.051 | **15.426** |
| *ecoli* | 0.291 | 0.141 | **51.676** |
| *glass* | 0.117 | 0.081 | **30.778** |
| *haberman* | 0.367 | 0.183 | **49.977** |
| *ionosphere* | 0.302 | 0.188 | **37.503** |
| *iris* | 0.226 | 0.096 | **57.518** |
| *obesity* | 10.616 | 6.940 | **34.628** |
| *segment* | 5.823 | 3.323 | **42.936** |
| *sonar* | 0.163 | 0.206 | -26.436 |
| *tae* | 0.132 | 0.103 | **22.028** |
| **MEAN** | 1.906 | 1.130 | 34.781 |

**Table 6.7:** AM4PC vs. AM4PC + Multiobjective mean runtime per optimization loop calculated on 10 evaluation scenario runs. Improvements are marked in bold.

The changes in selection frequency of both preprocessing methods and clustering algorithms from *AM4PC* to *AM4PC + Multiobjective* are shown in Figure 6.5 and Figure 6.6. The mean runtime of configurations including either the preprocessing methods or clustering algorithms increases from left to right (for an overview of runtimes see Figure 6.9).

The decrease in runtime is primarily achieved by the multiobjective optimizer favoring clustering algorithms with lower runtimes as seen in Figure 6.5 but still high accuracy. This is why runtime does not exclusively dictate selection frequency.

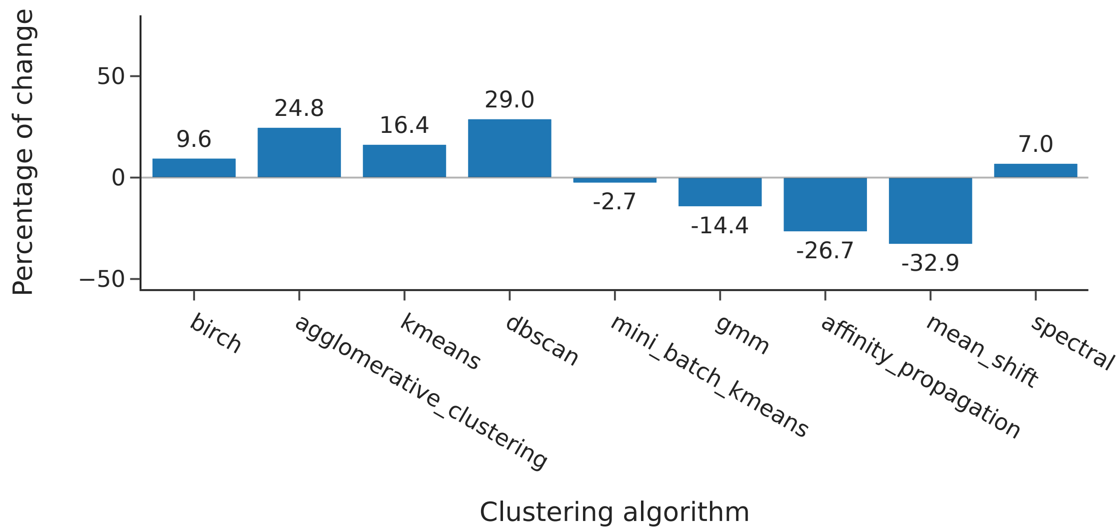


**Figure 6.5:** Change in selection frequency of clustering algorithms when comparing *AM4PC* and *AM4PC + Multiobjective*. Mean runtimes increase from left to right.

The selection of preprocessing methods is also influenced by the multiobjective optimization as seen in Figure 6.6. The selection frequency changes to a lesser extent with more outliers when compared to clustering algorithms. The most extreme outlier in this case is the *Robust Scaler*, as it is selected significantly more frequently even though it can heavily increase runtime. As selection is always a trade-off between runtime and accuracy, this hints at high accuracy.

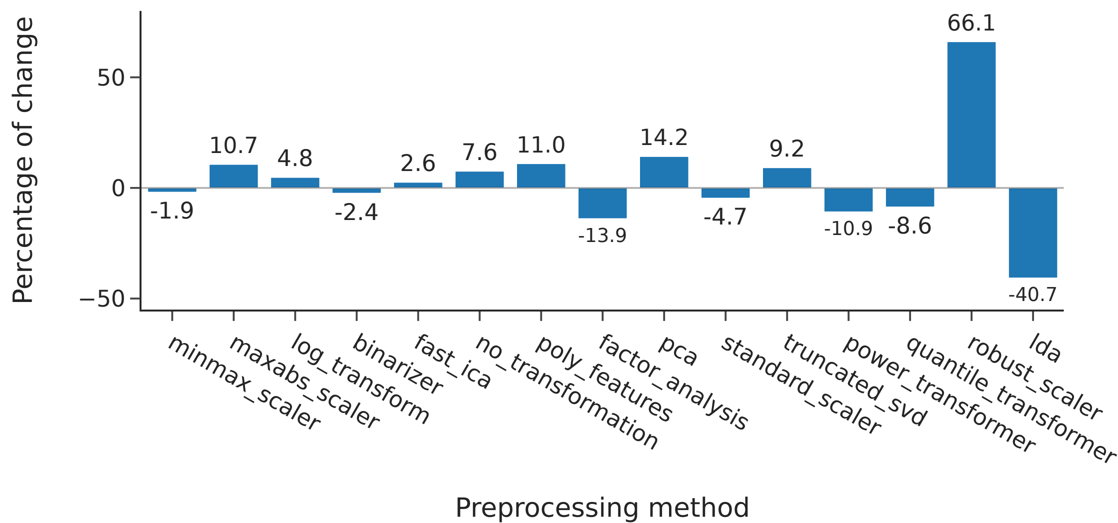


**Figure 6.6:** Change in selection frequency of preprocessing methods when comparing *AM4PC* and *AM4PC + Multiobjective*. Mean runtimes increase from left to right.

### 6.3.2 Sampling

Sampling is a common strategy to reduce runtimes by training the ML model on only a subset of the data. Therefore, 4 evaluation scenarios: *AM4PC + Sampling 0.1*, *AM4PC + Sampling 0.5*, *AM4PC + Sampling 0.1 + Tuning*, and *AM4PC + Sampling 0.5 + Tuning* (see Table A.1) are defined. Optimization occurs on the AMI calculated on the sampled data as the whole dataset is not available.
When tuning a similar approach to Hyperband is used, where the last 10 optimizer loops out of the 100 are executed on the full dataset to improve accuracy. In this final phase, the preprocessing methods and clustering algorithms are limited to the most promising one from the prior 90 optimization loops. Consequently, only the relevant hyperparameters are adjusted. This is theorized to be important, as the optimal value of parameters such as *eps* from *DB-SCAN* should heavily differ on the sampled data versus the whole dataset.
To validate, the best configurations AMI score over all the data is then calculated. This value can be used to compare results to other evaluation scenarios.

In Table 6.8 the mean AMI and runtime over the whole benchmark suite are shown for different evaluation scenarios.
When using only 10% of the data, the accuracy is 74.4% of that when using the whole dataset (AM4PC), but runtime decreases by 97.5%. The runtimes decrease of 97.5% while the data is only reduced by 90% is due to runtimes not being linearly dependent on the number of datapoints $n$.
Using 50% of the data still leads to a significant decrease in runtime by 80.7% while accuracy reaches 84.7% of that when optimized on the whole datasets.

Through tuning after sampling 10% of the data, the accuracy is increased to 80.9% of *AM4PC* while runtime is 95.0% reduced. Similarly, tuning the 50% sample on the whole dataset leads to 88.3% of the original accuracy. In this case, tuning significantly increases runtime which is only 50.2% reduced compared to training on the whole dataset.

| | **AM4PC** | **+ 10% Sampling** | **+ 50% Sampling** | **+ 10% Sampling + Tuning** | **50% Sampling + Tuning** |
|---|---|---|---|---|---|
| *Mean AMI* | 0.472 | 0.351 | 0.400 | 0.382 | 0.417 |
| *Percentage* | 100.000 | 74.364 | 84.746 | 80.932 | 88.347 |
| *Mean runtime* | 1.906 | 0.048 | 0.367 | 0.095 | 0.892 |
| *Percentage decrease* | 0.000 | 97.482 | 80.745 | 95.016 | 53.200 |

**Table 6.8:** Different sampling + tuning strategies AMI and runtime compared to optimization on the full dataset (AM4PC).

### 6.3.3 Summary

Both attempts at reducing runtime are promising. The multiobjective approach offers a decent reduction in runtime while the accuracy stays fairly similar. This is due to multiobjective optimization reaching a trade-off between both. Sampling leads to a much more significant reduction in runtime, but also accuracy. The accuracy can further be improved through a final tuning on the whole dataset at the cost of additional runtime.

In Figure 6.7 the trade-off between accuracy and runtime for different evaluation scenarios is shown.

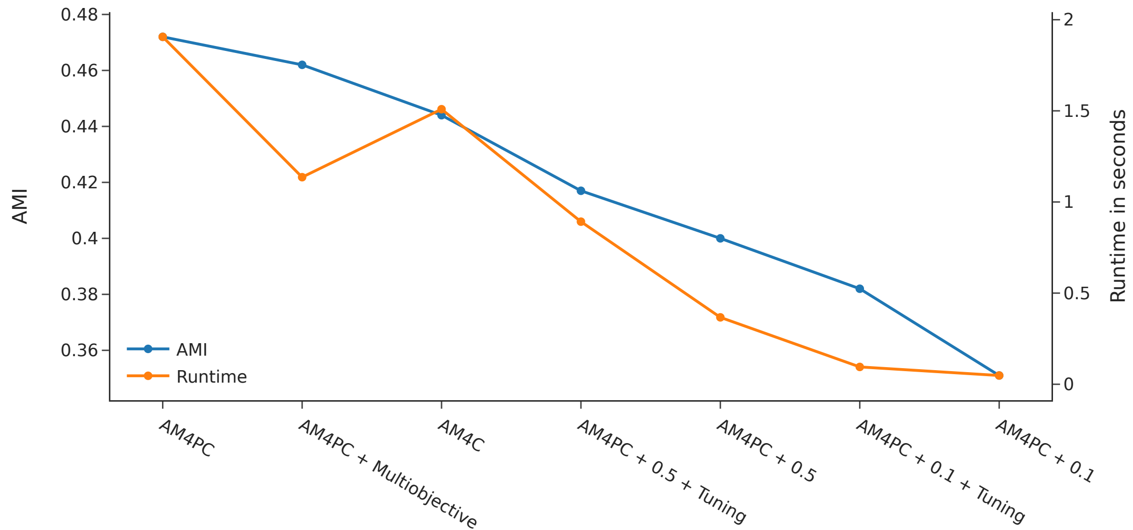


**Figure 6.7:** Trade-off between accuracy and runtime visualized for 7 evaluation scenarios.

## 6.4 Detailed Analysis of Preprocessing and Clustering Algorithms

This section covers the analysis of preprocessing and clustering algorithms made over all 85 evaluation scenario runs for a total of 176, 750 optimization loops and evaluated configurations. These include the frequency of preprocessing methods and clustering algorithms in the best configurations, the mean accuracy and runtime of the preprocessing methods and clustering algorithms, as well as the correlations between preprocessing methods and both clustering algorithms and datasets.

### 6.4.1 Selection Frequency

Figure 6.8 shows the frequency of preprocessing methods and clustering algorithms in winning configurations over all evaluation runs.
This is an excellent measure of how well a method or algorithm performs, as only the best configuration is used and therefore relevant. The top-performing algorithms, measured by selection rate in winning configurations, occur $2-3x$ more frequently than the worst-performing algorithms. Interestingly all algorithms, with the exception of categorical clustering algorithms, still occur frequently and therefore remain relevant. This indicates that including a large selection of clustering algorithms in the search space is beneficial.

Preprocessing methods behave similarly as shown in Figure 6.8. Here the selection frequency is more evenly distributed when disregarding *binarizing*, which is selected roughly 3*x* less frequently than the next best preprocessing method. All other preprocessing methods are selected $> 60$ times and therefore also indicate that a large selection of preprocessing algorithms in the search space is reasonable.

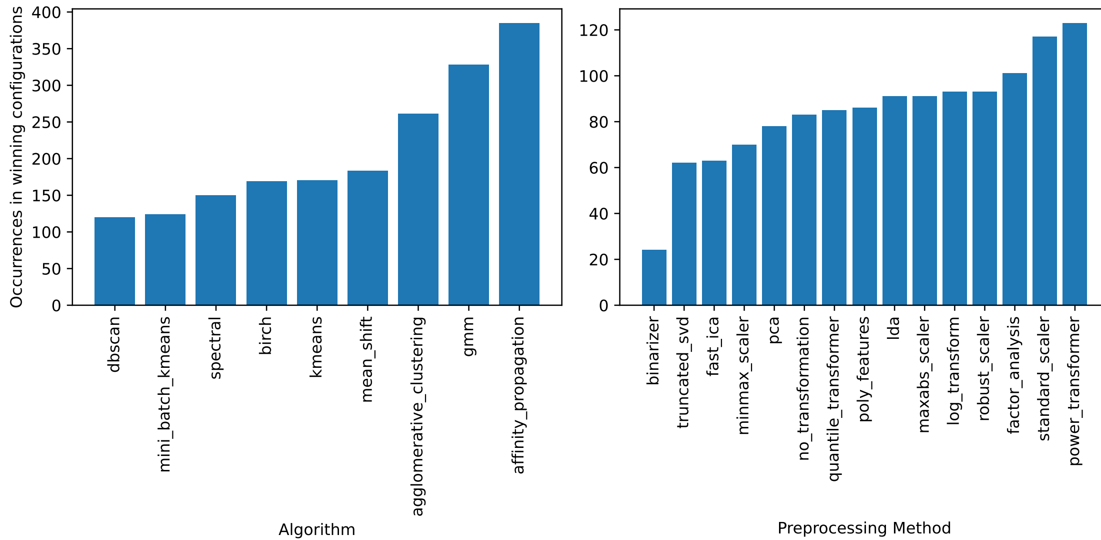


**Figure 6.8:** Number of winning configurations in which a clustering algorithm or preprocessing method occurs calculated over all 85 evaluation runs.

Therefore, both a large search space in regard to clustering algorithms and preprocessing methods is beneficial to maximize accuracy.

### 6.4.2 Runtime and Accuracy

When selecting which clustering algorithms and preprocessing methods to include in the search space, both runtime and accuracy are important.
To evaluate this optimally, every combination of clustering algorithm and preprocessing method would have to be executed the same amount of times on each dataset as in [WGL20]. As this is infeasible, the accuracy is calculated by taking the **best** performing optimization loop per dataset over all evaluation runs that contain the clustering algorithm or preprocessing method. Then the mean of these 14 values per clustering algorithm or preprocessing method is calculated. Only the best-performing optimization loop is used during calculation because the main focus is on how well an algorithm or method can perform and not how well it performs on average. This is especially important as hyperparameters are a large determining factor that dictate an algorithm's or method's success. For example, applying *K-Means* with a $k$ of 100 on a dataset that only contains 10 clusters will result in a bad accuracy that is not representative of the accuracy that could be achieved.
The runtime is calculated similarly, except for using the mean runtime per dataset per clustering

algorithm or preprocessing method instead of only considering the best optimizations loops. The runtime includes both the preprocessing and the clustering steps. Both results are shown in Figure 6.9 and Figure 6.10.

The results regarding clustering algorithm accuracy correlate in most cases with the selection frequency in winning configurations described in Section 6.4.1. Notable exceptions are *Spectral* clustering, which seems to do quite well on average, but is evidently outperformed regarding the maximization of the CVM and *Mini-Batch-K-Means*, which behaves similarly in regards to accuracy to a lesser extent.
All of the top performing clustering algorithms as seen in Figure 6.9, with the exception of *Spectral* clustering have comparatively low runtimes.

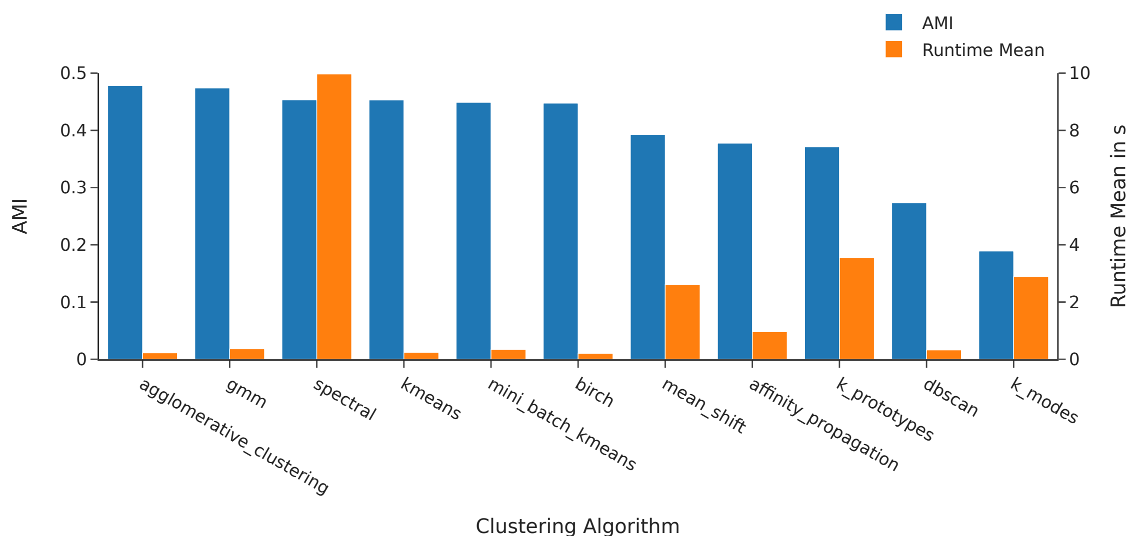


**Figure 6.9:** Clustering algorithms mean runtime compared to AMI.

The mean AMI of different preprocessing methods is fairly similar as seen in Figure 6.10 with the exception of the *binarizing* method as also seen in Section 6.4.1. Unlike AMI, runtime strongly differs with methods such as *LDA* and the *Robust Scaler* possessing especially high runtimes. There are two probable causes for increased runtime, either the execution of the method takes significant time or the method modifies the feature space in such a fashion that the subsequent clustering algorithm takes additional time. For the two most runtime intensive methods *LDA* and the *Robust Scaler* the prior seems more likely, as neither increases the number of features, with *LDA* even decreasing it.

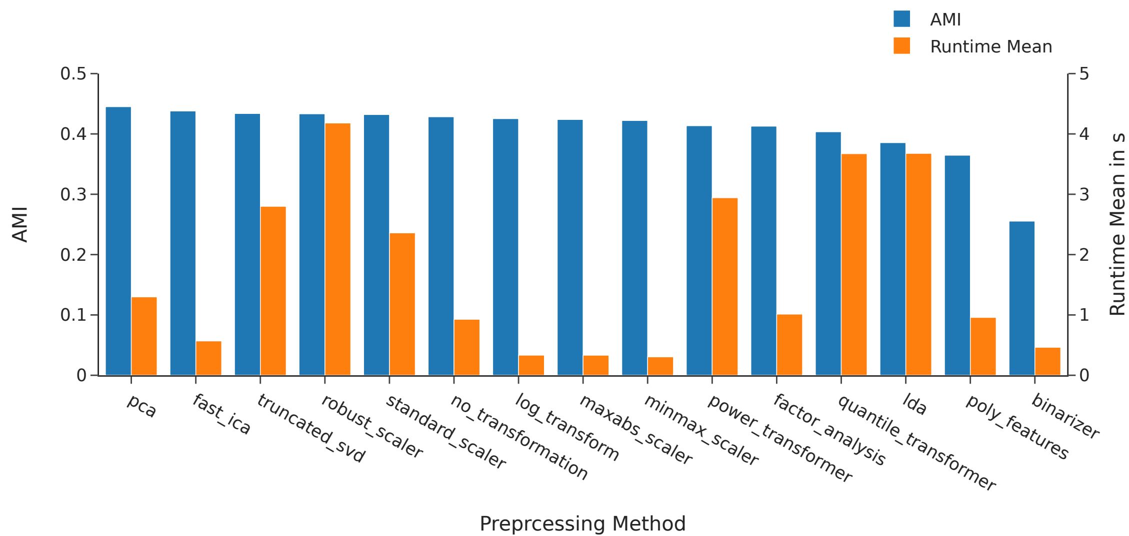


**Figure 6.10:** Preprocessing methods mean runtime compared to AMI.

### 6.4.3 Correlations

Prior work from Wang et. al. [WGL20] analyses the correlations between preprocessing methods, clustering algorithms, and datasets. They discover that the optimal preprocessing method depends on the subsequent clustering algorithm and the dataset, slightly favoring the latter.
Therefore, the correlations between the selection of preprocessing method and both the clustering algorithm and the dataset are measured over all evaluation runs. As BO selects promising combinations more frequently, the correlation (see Section 6.2) is higher between combinations that synergize. Therefore this is a valid measure of how well a preprocessing method fits a clustering algorithm or dataset.

Table 6.9 shows the highest correlating combinations between preprocessing methods and clustering algorithms or datasets. The mean of the highest individual correlations between preprocessing methods and clustering algorithms is 0.093. Combinations such as *No transformation* and *K-Means* are selected frequently and therefore correlate the strongest with a value of 0.193 .
The highest correlations between datasets and preprocessing methods are much higher with the combination of *Quantile Transformer* and the *sonar* dataset reaching 0.325 .
These correlations coincide with frequent winning combinations shown in Table A.2 where out of 55 winning configurations per dataset:

- On the *sonar* dataset 23 winning configurations apply *Quantile Transformer* and 15 *Power Transformer*.
- On the *car* dataset 23 winning configurations apply *Power Transformer*, 15 *Standard Scaler* and 9 *Factor Analysis*.
- On the *contraceptive* dataset 18 winning configurations apply *Log Transform* and 15 *Factor Analysis*.
- On the *segment* dataset 18 winning configurations apply *Factor Analysis*.

These high values indicate that the dataset is a strong influence in the selection of the preprocessing method.
The mean highest correlation is 0.178 which is nearly twice as high compared to the clustering algorithms. This further validates the findings of Wang et. al. [WGL20] that also show the dataset as a main determining factor when choosing a preprocessing method.

| **Preprocessing Methods** | **Clustering Algorithm** | **Correlation** | **Dataset** | **Correlation** |
|---|---|---|---|---|
| *Standard Scaler* | DB-SCAN | 0.054 | car | 0.136 |
| *MinMax Scaler* | Mini-Batch-K-Means | 0.039 | tae | 0.138 |
| *MaxAbs Scaler* | Affinity Propagation | 0.074 | tae | 0.112 |
| *Robust Scaler* | Agglomerative Clustering | 0.067 | obesity | 0.157 |
| *Log Transformer* | DB-SCAN | 0.045 | contraceptive | 0.222 |
| *Power Transformer* | Spectral | 0.086 | car | 0.243 |
| *Binarizer* | Birch | 0.052 | rna | 0.142 |
| *Quantile Transformer* | Agglomerative Clustering | 0.086 | sonar | 0.325 |
| *PCA* | Birch | 0.105 | glass | 0.133 |
| *Fast-ICA* | GMM | 0.145 | ionosphere | 0.180 |
| *Factor-Analysis* | Birch | 0.149 | segment | 0.214 |
| *LDA* | Spectral | 0.076 | rna | 0.187 |
| *Truncated-SVD* | Birch | 0.047 | obesity | 0.264 |
| *Poly Features* | Affinity Propagation | 0.181 | ecoli | 0.147 |
| *No Transformation* | K-Means | 0.193 | glass | 0.065 |
| **MEAN** | | 0.093 | | 0.178 |

**Table 6.9:** Highest correlation between preprocessing methods and clustering algorithms / datasets.

Many clustering systems use meta-learning as described in Chapter 3, where certain calculated features of the dataset determine which clustering algorithm is applied, have found success. Due to the high correlation of 0.178 between the dataset and preprocessing method, significantly higher than between the clustering algorithm and the dataset at 0.126, it is reasonable to assume that the same could be applied to the selection of preprocessing methods with even greater success. This is referred to future work.

# 7 Conclusion and Outlook

As the amount of data increases so does the potential gain from analysis. This analysis is commonly conducted using the KDD process and involves the complicated and time-intensive step of building, training, and tuning ML pipelines. To reduce the need for data scientists and at the same time enable domain experts to analyze their own data, AutoML solutions are developed. These automate large parts of the ML pipeline.

Most of the widespread and popular solutions such as *TPOT* and *auto-sklearn* focus on supervised learning. Only a few prototypical packages such as *AutoML4Clust* (AM4C), *AutoCluster* and *TPE-AutoClust* focus on clustering and therefore unsupervised learning. Most of these packages do not consider preprocessing methods or their hyperparameters when optimizing. The ones that do only implement a small selection with minimal analysis.
As preprocessing has been proven to increase accuracy in both supervised AutoML and clustering, it is reasonable to assume that the addition of preprocessing to AutoML for clustering could improve results.

Preprocessing methods are analyzed and selected due to precedent in existing supervised AutoML systems as well as clustering use-cases. The package *AutoML4Clust* (AM4C) is extended by adding a selection of 14 preprocessing methods with their respective hyperparameters in a similar fashion to *auto-sklearn*. These preprocessing methods are constrained to feature transformation, which includes multiple scaling and discretization methods, and feature extraction methods which mainly include dimensionality-reducing mappings. These methods are modularly added to AM4C and evaluated on both synthetic and real-world datasets as well as on internal and external CVMs.

Results indicate that preprocessing increases mean performance by 24.6% on real-world datasets. As most datasets are a mix of categorical and numerical data, clustering algorithms which are capable of dealing with categorical data are added. These perform poorly and slightly reduce the overall performance when added. When using exclusively internal CVMs, preprocessing also improves results, therefore validating the applicability to datasets with unknown ground truth labels. Preprocessing does not improve clustering on the chosen synthetic data. This is because the synthetic data is specifically generated for clustering and therefore has little need for preprocessing.

Unfortunately, while increasing accuracy the inclusion of preprocessing in the search space also increases runtime. Therefore, two methods are explored to reduce runtime. Firstly multiobjective optimization, which aims to maximize both accuracy and minimize runtime by reaching an optimal trade-off. This proves to be promising as it reduces mean runtime by 34.7%, while only slightly impacting accuracy. The second method is sampling, where only part of the data is used to optimize the ML pipeline. This leads to massive reductions of runtimes by up to 97.5%, but also significantly reduces the accuracy by up to 25.7% .

Additional observations made over all evaluations include the fact that all preprocessing methods and clustering algorithms are chosen a significant number of times. This suggests the merit of a large search space, although due to runtime constraints, it could be sensible to cut some of the lower accuracy high runtime methods. Lastly, it seems like the choice of preprocessing method, while dependent on the subsequent clustering method, mainly depends on the dataset it is applied to.

This work proves that the addition of preprocessing methods to AutoML clustering systems can lead to significant improvements in clustering accuracy.

## Outlook

This work adds preprocessing methods to the search space and applies one at a time similarly to *auto-sklearn*. This condition is restrictive, as preprocessing usually consists of multiple methods applied in sequence. For example, this would allow for datasets to be both scaled and expanded with polynomial features instead of having to choose one. This could be extended to the point where different features are preprocessed differently as with *TPOT*.
These additions would further increase an already large search space. To counteract this, meta-learning could be used to speed up the selection of preprocessing methods as done with clustering algorithms. Especially since the dependence of preprocessing method and dataset seems to be even higher than between the clustering algorithm and the dataset.
The area of feature selection, while adding additional complexity, could also be considered in future work. A large selection of unsupervised feature selection techniques exists [SCM20].

While this work attempts the implementation of categorical clustering algorithms, the implemented methods are basic and therefore do not outperform more established clustering algorithms. This could be improved upon by implementing more modern categorical clustering algorithms described in [BHW13] such as ROCK.

Most of the evaluation scenarios are optimized on external CVMs which do not transfer to real-world applications where ground truth labels are not available. In this area other work has shown, that considering multiple internal CVMs with multiobjective optimization or training a ML model that predicts an external CVM from internal CVMs is promising. These could be implemented in future work to increase the transferability of results to data with unknown ground truth labels.

While the runtime is successfully reduced with multiobjective optimization and sampling, future work could possibly perform even better by meta-optimizing the exploitation versus exploration trade-off or implementing different optimizers such as Hyperband or BOHB. Especially the latter could be effective, as the combination of sampling and tuning led to significant reductions in runtime.

# Bibliography


[Abd22] H. I. Abdalla. "A Brief Comparison of K-means and Agglomerative Hierarchical Clustering Algorithms on Small Datasets". In: *Proceeding of 2021 International Conference on Wireless Communications, Networking and Applications* (2022), pp. 623–632 (cit. on p. 22).

[ABKS99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, J. Sander. "OPTICS: Ordering points to identify the clustering structure". In: *ACM Sigmod record* 28 (2 1999), pp. 49–60 (cit. on p. 22).

[Alo13] J. B. Alonso. *K-means vs Mini Batch K-means: a comparison*. https://upcommons.upc.edu/handle/2117/23414. 2013 (cit. on p. 22).

[AYS17] B. Aleksandar, M. Yves, G. Stephan. "Robust Spectral Clustering for Noisy Data". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017) (cit. on p. 22).

[BCHC09] J. Benesty, J. Chen, Y. Huang, I. Cohen. *Pearson correlation coefficient*. 2009 (cit. on p. 52).

[BHW13] P. M. Bhagat, P. S. Halgaonkar, V. M. Wadhai. "Review of clustering algorithm for categorical data". In: *International Journal of Engineering and Advanced Technology* 3 (2 2013), pp. 341–345 (cit. on p. 68).

[CBL15] T. V. Craenendonck, H. Blockeel, K. U. Leuven. "Using Internal Validity Measures to Compare Clustering Algorithms". In: *Benelearn 2015 Poster presentations (online)* (2015), pp. 1–8 (cit. on pp. 23, 24).

[CH74] T. Caliński, J. Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics* 3 (1 1974), pp. 1–27. DOI: 10.1080/03610927408827101. URL: https://www.tandfonline.com/doi/abs/10.1080/03610927408827101 (cit. on p. 23).

[Che95] Y. Cheng. "Mean shift, mode seeking, and clustering". In: *IEEE transactions on pattern analysis and machine intelligence* 17 (8 1995), pp. 790–799 (cit. on p. 22).

[CK15] C. Cristescu, J. Knowles. "Surrogate-based multiobjective optimization: ParEGO update and test". In: *Workshop on Computational Intelligence (UKCI)* 770 (2015) (cit. on p. 46).

[DH04] C. Ding, X. He. "K-Means Clustering via Principal Component Analysis". In: *Proceedings of the Twenty-First International Conference on Machine Learning* (2004), p. 29. DOI: 10.1145/1015330.1015408. URL: https://doi.org/10.1145/1015330.1015408 (cit. on pp. 16, 35).

[EFH+13] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, K. Leyton-Brown, et al. "Towards an empirical foundation for assessing bayesian optimization of hyperparameters". In: *NIPS workshop on Bayesian Optimization in Theory and Practice* 10 (3 2013) (cit. on pp. 18, 19).

[EKSX+96] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd* 96 (34 1996), pp. 226–231 (cit. on p. 22).

[ELS21] R. ElShawi, H. Lekunze, S. Sakr. "cSmartML: A Meta Learning-Based Framework for Automated Selection and Hyperparameter Tuning for Clustering". In: *2021 IEEE International Conference on Big Data (Big Data)* (2021), pp. 1119–1126. DOI: 10.1109/BigData52589.2021.9671542 (cit. on pp. 16, 28, 50).

[ES22] R. ElShawi, S. Sakr. "TPE-AutoClust: A Tree-based Pipline Ensemble Framework for Automated Clustering". In: *2022 IEEE International Conference on Data Mining Workshops (ICDMW)* (2022), pp. 1144–1153. DOI: 10.1109/ICDMW58026.2022.00149 (cit. on pp. 16, 27, 28, 34, 35).

[FCW+21] C. Fan, M. Chen, X. Wang, J. Wang, B. Huang. "A Review on Data Preprocessing Techniques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data". In: *Frontiers in Energy Research* 9 (2021). ISSN: 2296-598X. DOI: 10.3389/fenrg.2021.652801. URL: https://www.frontiersin.org/articles/10.3389/fenrg.2021.652801 (cit. on pp. 32, 34).

[FD07] B. J. Frey, D. Dueck. "Clustering by Passing Messages Between Data Points". In: *Science* 315 (5814 2007), pp. 972–976. DOI: 10.1126/science.1136800. URL: https://www.science.org/doi/abs/10.1126/science.1136800 (cit. on p. 21).

[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* 28 (2015). Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett. URL: https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf (cit. on pp. 16, 25, 28, 34, 35).

[FPS96] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. "The KDD process for extracting useful knowledge from volumes of data". In: *Communications of the ACM* 39 (11 1996), pp. 27–34 (cit. on p. 15).

[Gia16] P. Giannis. "Master Thesis: A Study on Automated Machine Learning in the context of Clustering". 2016 (cit. on p. 32).

[GK78] K. C. Gowda, G. Krishna. "Agglomerative clustering using the concept of mutual nearest neighbourhood". In: *Pattern Recognition* 10 (2 1978), pp. 105–112. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(78)90018-3. URL: https://www.sciencedirect.com/science/article/pii/0031320378900183 (cit. on p. 21).

[GLH15] S. Garcia, J. Luengo, F. Herrera. *Data preprocessing in data mining*. Springer, 2015 (cit. on p. 31).

[GRL+16] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, F. Herrera. "Big data preprocessing: methods and prospects". In: *Big Data Analytics* 1 (1 2016), p. 9. ISSN: 2058-6345. DOI: 10.1186/s41044-016-0014-0. URL: https://doi.org/10.1186/s41044-016-0014-0 (cit. on pp. 31–34).

[HA85] L. Hubert, P. Arabie. "Comparing partitions". In: *Journal of classification* 2 (1985), pp. 193–218 (cit. on p. 24).

[HHL11] F. Hutter, H. H. Hoos, K. Leyton-Brown. “Sequential model-based optimization for general algorithm configuration”. In: *Learning and Intelligent Optimization: 5th International Conference* (2011), pp. 507–523 (cit. on p. 19).

[Hua97] Z. Huang. “Clustering large data sets with mixed numeric and categorical values”. In: *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD)* (1997), pp. 21–34 (cit. on p. 21).

[Hua98] Z. Huang. “Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values”. In: *Data Mining and Knowledge Discovery* 2 (3 1998), pp. 283–304. ISSN: 1573-756X. DOI: 10.1023/A:1009769707641. URL: https://doi.org/10.1023/A:1009769707641 (cit. on pp. 21, 22).

[HZC21] X. He, K. Zhao, X. Chu. “AutoML: A survey of the state-of-the-art”. In: *Knowledge-Based Systems* 212 (Jan. 2021), p. 106622. ISSN: 0950-7051. DOI: 10.1016/J.KNOSYS.2020.106622 (cit. on pp. 16–19, 31–33).

[IM98] P. Indyk, R. Motwani. “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality”. In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (1998), pp. 604–613. DOI: 10.1145/276698.276876. URL: https://doi.org/10.1145/276698.276876 (cit. on pp. 31, 33).

[Jai10] A. K. Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern Recognition Letters* 31 (8 June 2010), pp. 651–666. ISSN: 01678655. DOI: 10.1016/j.patrec.2009.09.011 (cit. on p. 20).

[JD88] A. K. Jain, R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988 (cit. on pp. 20, 23).

[KAH19] V. Y. Kiselev, T. S. Andrews, M. Hemberg. “Challenges in unsupervised clustering of single-cell RNA-seq data”. In: *Nature Reviews Genetics* 20 (5 2019), pp. 273–282 (cit. on p. 28).

[KCA15] A. I. Kadhim, Y. N. Cheah, N. H. Ahamed. “Text Document Preprocessing and Dimension Reduction Techniques for Text Document Clustering”. In: *Proceedings - 2014 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology, ICAIET 2014* (2015), pp. 69–73. DOI: 10.1109/ICAIET.2014.21 (cit. on p. 35).

[KKP06] S. B. Kotsiantis, D. Kanellopoulos, P. E. Pintelas. “Data preprocessing for supervised leaning”. In: *International journal of computer science* 1 (2 2006), pp. 111–117 (cit. on pp. 31–34).

[Kno06] J. Knowles. “ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems”. In: *IEEE Transactions on Evolutionary Computation* 10 (1 2006), pp. 50–66 (cit. on p. 46).

[LEF+19] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, F. Hutter. “BOAH: A tool suite for multi-fidelity bayesian optimization and analysis of hyperparameters”. In: *CoRR* (2019) (cit. on p. 39).

[LEF+22] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, F. Hutter. “SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization”. In: *Journal of Machine Learning Research* 23 (54 2022), pp. 1–9. URL: http://jmlr.org/papers/v23/21-0888.html (cit. on p. 39).

[LJRT18] L. Li, K. Jamieson, A. Rostamizadeh, A. Talwalkar. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018, pp. 1–52. URL: http://jmlr.org/papers/v18/16-558.html. (cit. on p. 19).

[LLL+18] T. Lan, K. R. Lin, Z. Y. Liu, Y. H. He, C. Y. Xu, H. B. Zhang, X. H. Chen. "A Clustering Preprocessing Framework for the Subannual Calibration of a Hydrological Model Considering Climate-Land Surface Variations". In: *Water Resources Research* 54 (12 Dec. 2018), pp. 10, 034–10, 052. ISSN: 19447973. DOI: 10.1029/2018WR023160 (cit. on p. 16).

[LLX+10] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu. "Understanding of internal clustering validation measures". In: *2010 IEEE international conference on data mining* (2010), pp. 911–916 (cit. on p. 23).

[LW21] L. Y. Li, S. T. Wenjie. "AutoCluster: Meta-learning Based Ensemble Method for Automated Unsupervised Clustering". In: *Advances in Knowledge Discovery and Data Mining* (2021). Ed. by Hong, R. Naren, A. R. K., R. P. Krishna, S. Jaideep, C. T. K. Kamal, Cheng, pp. 246–258 (cit. on pp. 16, 24, 27, 28, 49).

[LXZ15] S. Li, L. D. Xu, S. Zhao. "The internet of things: a survey". In: *Information Systems Frontiers* 17 (2 2015), pp. 243–259. ISSN: 1572-9419. DOI: 10.1007/s10796-014-9492-7. URL: https://doi.org/10.1007/s10796-014-9492-7 (cit. on p. 15).

[Mad12] T. S. Madhulatha. "An Overview on Clustering Methods". In: *IOSR Journal of Engineering* 2 (4 2012), pp. 719–725. DOI: 10.48550/ARXIV.1205.1117. URL: https://arxiv.org/abs/1205.1117 (cit. on pp. 21, 22).

[MJC+14] D. Moulavi, P. A. Jaskowiak, R. J. G. B. Campello, A. Zimek, J. Sander. "Density-based clustering validation". In: *Proceedings of the 2014 SIAM international conference on data mining* (2014), pp. 839–847 (cit. on p. 23).

[OEM16] R. S. Olson, O. Edu, J. H. Moore. "TPOT: A tree-based pipeline optimization tool for automating machine learning". In: *Workshop on automatic machine learning* 64 (2016), pp. 66–74. URL: https://github.com/rhiever/tpot. (cit. on pp. 15, 16, 25, 26, 28, 35).

[PDK20] Y. Poulakis, C. Doulkeridis, D. Kyriazis. "AutoClust: A Framework for Automated Clustering Based on Cluster Validity Indices". In: *2020 IEEE International Conference on Data Mining (ICDM)* (2020), pp. 1220–1225. DOI: 10.1109/ICDM50108.2020.00153 (cit. on pp. 16, 24, 28, 31).

[Pea00] K. Pearson. "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50 (302 July 1900), pp. 157–175. DOI: 10.1080/14786440009463897. URL: https://doi.org/10.1080/14786440009463897 (cit. on p. 32).

[PF01] K. Pearson, F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2 (11 1901), pp. 559–572. DOI: 10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720 (cit. on p. 33).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830 (cit. on pp. 25–27, 39).

[RA18] L. A. Rasyid, S. Andayani. “Review on clustering algorithms based on data type: towards the method for data combined of numeric-fuzzy linguistics”. In: *Journal of Physics: Conference Series* 1097 (1 2018), p. 12082 (cit. on p. 22).

[Ras99] C. Rasmussen. “The infinite Gaussian mixture model”. In: *Advances in neural information processing systems* 12 (1999) (cit. on p. 22).

[RVBV16] S. Romano, N. X. Vinh, J. Bailey, K. Verspoor. “Adjusting for chance clustering comparison measures”. In: *The Journal of Machine Learning Research* 17 (1 2016), pp. 4635–4666 (cit. on p. 24).

[SCM20] S. Solorio-Fernández, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad. “A review of unsupervised feature selection methods”. In: *Artificial Intelligence Review* 53 (2 2020), pp. 907–948. ISSN: 1573-7462. DOI: 10.1007/s10462-019-09682-y. URL: https://doi.org/10.1007/s10462-019-09682-y (cit. on pp. 32, 68).

[SJ13] S. Singhal, M. Jena. “A study on WEKA tool for data preprocessing, classification and clustering”. In: *International Journal of Innovative technology and exploring engineering (IJItee)* 2 (6 2013), pp. 250–253 (cit. on p. 31).

[SWJ98] M. Schonlau, W. J. Welch, D. R. Jones. “Global versus local search in constrained optimization of computer models”. In: *Lecture notes-monograph series* (1998), pp. 11–25 (cit. on p. 18).

[Tal99] L. Talavera. “Feature selection as a preprocessing step for hierarchical clustering”. In: *ICML* 99 (1999), pp. 389–397 (cit. on p. 35).

[TFS21] D. Tschechlov, M. Fritz, H. Schwarz. “AutoML4Clust: Efficient AutoML for clustering analyses”. In: *Advances in Database Technology - EDBT* 2021-March (2021), pp. 343–348. ISSN: 23672005. DOI: 10.5441/002/edbt.2021.32 (cit. on pp. 16, 27, 28, 36, 49–52).

[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* (2013), pp. 847–855. DOI: 10.1145/2487575.2487629. URL: https://doi.org/10.1145/2487575.2487629 (cit. on pp. 16–18).

[TM21] O. Taratukhin, S. Muravyov. “Meta-Learning Based Feature Selection for Clustering”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 13113 LNCS (2021), pp. 548–559. ISSN: 16113349. DOI: 10.1007/978-3-030-91608-4_54 (cit. on p. 32).

[TRSM23] D. Treder-Tschechlov, P. Reimann, H. Schwarz, B. Mitschang. “Approach to synthetic data generation for imbalanced multi-class problems with heterogeneous groups”. In: *BTW 2023* (2023) (cit. on p. 50).

[VEB09] N. X. Vinh, J. Epps, J. Bailey. “Information theoretic measures for clusterings comparison: is a correction for chance necessary?” In: *Proceedings of the 26th annual international conference on machine learning* (2009), pp. 1073–1080 (cit. on p. 24).

[VOE11] R. L. Villars, C. W. Olofson, M. Eastwood. “Big data: What it is and why you should care”. In: *White paper, IDC* 14 (2011), pp. 1–14 (cit. on p. 15).

[WGL20] C. Wang, X. Gao, J. Liu. “Impact of data preprocessing on cell-type clustering based on single-cell RNA-seq data”. In: *BMC bioinformatics* 21 (1 2020), pp. 1–13 (cit. on pp. 16, 28, 35, 62, 64, 65).

[XT15] D. Xu, Y. Tian. “A comprehensive survey of clustering algorithms”. In: *Annals of Data Science* 2 (2 2015), pp. 165–193 (cit. on pp. 16, 20–22).

[YS20] L. Yang, A. Shami. “On hyperparameter optimization of machine learning algorithms: Theory and practice”. In: *Neurocomputing* 415 (2020), pp. 295–316 (cit. on pp. 18–20).

[YWC+18] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, Y. Yu. “Taking human out of learning applications: A survey on automated machine learning”. In: *CoRR* (2018) (cit. on pp. 16, 17, 31–33).

[ZRL96] T. Zhang, R. Ramakrishnan, M. Livny. “BIRCH: An Efficient Data Clustering Method for Very Large Databases”. In: *SIGMOD Rec.* 25 (2 June 1996), pp. 103–114. ISSN: 0163-5808. DOI: 10.1145/235968.233324. URL: https://doi.org/10.1145/235968.233324 (cit. on p. 21).

All links were last followed on March 21.04.2023.

# A Appendix

| Evaluation Scenario | Data Type | CVM | Preprocessing Methods | Clustering Algorithms | Multiobjective | Sampling | Tuning | Runs |
|---|---|---|---|---|---|---|---|---|
| *AM4C synthetic* | synthetic | AMI | all | all numeric | false | false | - | 5 |
| *AM4PC synthetic* | synthetic | AMI | none | all numeric | false | false | - | 5 |
| ***AM4C real*** | real | AMI | all | all numeric | false | false | - | 10 |
| *AM4PC real* | real | AMI | none | all numeric | false | false | - | 10 |
| *AM4PC + Categorical* | real | AMI | all | all | false | false | - | 5 |
| *AM4PC + Multiobjective* | real | AMI | all | all numeric | true | false | - | 10 |
| *AM4PC + Sampling 0.1* | real | AMI | all | all numeric | false | 0.1 | false | 5 |
| *AM4PC + Sampling 0.5* | real | AMI | all | all numeric | false | 0.5 | false | 5 |
| *AM4PC + Sampling 0.1 + Tuning* | real | AMI | all | all numeric | false | 0.1 | true | 5 |
| *AM4PC + Sampling 0.5 + Tuning* | real | AMI | all | all numeric | false | 0.5 | true | 5 |
| *AM4PC on CH* | real | CH | all | all numeric | false | false | - | 5 |
| *AM4C on CH* | real | CH | none | all numeric | false | false | - | 5 |
| *AM4PC on DBCV* | real | DBCV | all | all numeric | false | false | - | 5 |
| *AM4C on DBCV* | real | DBCV | none | all numeric | false | false | - | 5 |

**Table A.1:** Evaluation Scenarios, where deviations from the standard scenario *AM4PC real* are underlined.

| Dataset | no_transformation | pca | fast_ica | factor_analysis | lda | truncated_svd | standard_scaler | minmax_scaler | maxabs_scaler | robust_scaler | quantile_transformer | power_transformer | log_transform | binarizer | poly_features |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| car | 2 | 4 | 1 | 9 | 0 | 0 | 15 | 0 | 0 | 1 | 0 | 23 | 0 | 0 | 0 |
| contraceptive | 0 | 4 | 2 | 15 | 4 | 0 | 2 | 2 | 1 | 0 | 1 | 5 | 18 | 1 | 0 |
| diagnosis | 1 | 4 | 5 | 10 | 12 | 0 | 3 | 2 | 4 | 0 | 2 | 4 | 3 | 5 | 0 |
| ecoli | 1 | 3 | 3 | 2 | 0 | 2 | 5 | 9 | 7 | 8 | 2 | 0 | 0 | 0 | 13 |
| frogs | 7 | 7 | 5 | 0 | 13 | 4 | 1 | 1 | 2 | 7 | 0 | 0 | 7 | 0 | 1 |
| glass | 8 | 11 | 2 | 4 | 1 | 9 | 6 | 1 | 1 | 1 | 3 | 4 | 3 | 0 | 1 |
| haberman | 0 | 4 | 2 | 1 | 11 | 0 | 4 | 1 | 4 | 2 | 5 | 0 | 15 | 2 | 4 |
| ionosphere | 1 | 9 | 12 | 7 | 0 | 4 | 2 | 3 | 3 | 7 | 1 | 2 | 3 | 1 | 0 |
| iris | 1 | 1 | 1 | 5 | 14 | 4 | 3 | 1 | 7 | 5 | 1 | 1 | 9 | 0 | 2 |
| obesity | 2 | 3 | 0 | 0 | 1 | 16 | 3 | 1 | 0 | 14 | 1 | 4 | 0 | 2 | 8 |
| rna | 7 | 4 | 0 | 0 | 15 | 6 | 3 | 1 | 3 | 1 | 2 | 3 | 4 | 6 | 0 |
| segment | 0 | 1 | 10 | 18 | 0 | 0 | 5 | 9 | 6 | 0 | 3 | 1 | 1 | 1 | 0 |
| sonar | 0 | 0 | 1 | 3 | 1 | 0 | 4 | 3 | 3 | 0 | 23 | 15 | 2 | 0 | 0 |
| tae | 1 | 4 | 3 | 2 | 1 | 0 | 4 | 11 | 11 | 0 | 8 | 6 | 3 | 0 | 1 |

**Table A.2:** Preprocessing methods selection counts per dataset

**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