Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Analysis and Evaluation of Data Preprocessing Methods for Clustering Analyses

Leonard Labes

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Dennis Treder-Tschechlov, M.Sc.
Commenced:	November 13, 2023
Completed:	May 22, 2024

Abstract

Data is often used to extract knowledge from it and guide decisions in several areas. The knowledge extraction process is usually done with a machine-learning model like clustering. However, frequently, this data contains data imperfections such as missing values, outliers, or the data has skewed distributions. These imperfections need to be addressed to extract knowledge from the data because otherwise, machine-learning models can not be applied or achieve poor results. This process is called preprocessing. Many different preprocessing methods and corresponding hyperparameters exist. Therefore, finding a good selection of methods and hyperparameters to improve the machine-learning model result is challenging, especially if the data imperfections are unknown. In addition, more than one preprocessing method is often used, which increases the search space and brings additional challenges as the order in which they are performed and the interaction between the preprocessing methods is unclear. This may result in a lengthy trial and error process, especially for inexperienced users, where different preprocessing pipelines are evaluated until an acceptable pipeline is found.

Some recent advances have been made in the area of AutoML that cover the automation of this selection as a small part of their approach. However, these are limited in the area of clustering, and most approaches propose only a single preprocessing method or consider only a limited number of methods in their configuration space. One reason for these limitations is that they focus on an end-to-end approach where, besides an appropriate preprocessing pipeline, a clustering algorithm and its parameters are suggested as well.

The AutoML approaches consider preprocessing to be a small part at best and focus primarily on model building. In contrast, this work focuses mainly on addressing the challenges named in the first paragraph and improving the results with appropriate preprocessing pipelines while using a model, in this case clustering, to evaluate the pipelines.

To mitigate the challenges described in the first section, the overall goal is to make accurate suggestions for preprocessing pipelines that improve the clustering result. In order to achieve this, it is important to identify data imperfections and to understand the relationship between data imperfections and preprocessing pipelines. This thesis contributes a first step in that direction with the concept of how a knowledge base can be created that accurately measures the effects of preprocessing pipelines on data with imperfections and can identify data imperfections of datasets. In order to achieve the latter, meta-features are evaluated because they are a good way to describe unseen datasets and their imperfections. Such a knowledge base needs to contain information from many different datasets to be able to generalize. Because of that, synthetic data is used and further manipulated with data imperfections to have an almost unlimited dataset available and to make a precise evaluation of what data imperfection is handled by which pipeline well. These manipulations skew the data distribution, remove parts of the data to create missing values, and add outliers to the data. Because of the named challenges of section one, it is impossible to apply all combinations of preprocessing methods and their hyperparameters to a dataset, even if the selection of preprocessing methods is small. Instead, pipelines are generated and refined during an optimization process. As optimization, Genetic optimization is used because it provides high flexibility and can be well customized to address the named challenges.

The evaluation shows that for most datasets, a preprocessing pipeline is found by the optimizer, which leads to a significant improvement in the clustering results compared to the results without preprocessing. The improvement is most significant for skewed distribution data, while datasets that have not been manipulated show the slightest improvement. Additionally, the evaluation of the optimization process shows that a well-performing pipeline is found relatively quickly, while the improvement afterward exists but is comparatively small. It is shown that the missing value imputation works best with the KNN-Imputer compared to other imputation techniques. Other data imperfections do not produce a preferred method or pipeline. Concerning the order of the preprocessing methods within a pipeline, it could not be shown that there is a significant difference. Additionally, it is demonstrated that meta-features correlate with data imperfections. Therefore, it suggests that it is possible to determine the need for preprocessing and the identification of data imperfections with the thesis used meta-features.

Kurzfassung

Daten werden häufig dazu verwendet, um Wissen aus ihnen zu extrahieren und Entscheidungen in verschiedenen Bereichen zu treffen. Der Prozess der Wissensextraktion erfolgt in der Regel mit einem maschinellen Lernmodell wie Clustering. Häufig enthalten diese Daten jedoch Fehler wie fehlende Werte, Ausreißer oder eine schiefe Verteilung der Daten. Diese Fehler müssen beseitigt werden, um Wissen aus den Daten zu extrahieren, da sonst die Modelle des maschinellen Lernens nicht angewandt werden können oder schlechten Ergebnisse erzielen. Dieser Prozess wird als Vorverarbeitung bezeichnet. Es gibt viele verschiedene Vorverarbeitungsmethoden und entsprechende Hyperparameter. Daher ist es eine Herausforderung, eine gute Auswahl an Methoden und Hyperparametern zu finden, um das Ergebnis des maschinellen Lernens zu verbessern, insbesondere wenn die Fehler der Daten unbekannt sind. Darüber hinaus wird häufig mehr als eine Vorverarbeitungsmethode verwendet, was den Suchraum vergrößert und zusätzliche Herausforderungen mit sich bringt, da die Reihenfolge, in der sie durchgeführt werden, und die Interaktion zwischen den Vorverarbeitungsmethoden unklar sind. Dies kann zu einem langwierigen Versuch und Irrtum Prozess führen, bei dem verschiedene Vorverarbeitungspipelines bewertet werden, bis eine akzeptable Pipeline gefunden ist. Das ist insbesondere für unerfahrene Benutzer häufig der Fall.

In letzter Zeit wurden einige Fortschritte auf dem Gebiet AutoML erzielt, die die Automatisierung dieser Auswahl als einen kleinen Teil ihres Ansatzes abdecken. Diese sind jedoch auf den Bereich des Clustering begrenzt vorhanden, und die meisten Ansätze schlagen nur eine einzige Vorverarbeitungsmethode vor oder berücksichtigen nur eine begrenzte Anzahl von Methoden in ihrem Konfigurationsraum. Ein Grund für diese Einschränkungen ist, dass sie sich auf einen Ende zu Ende Ansatz konzentrieren, bei dem neben einer geeigneten Vorverarbeitungspipeline auch ein Clustering-Algorithmus und dessen Parameter vorgeschlagen werden.

Die AutoML-Ansätze betrachten die Vorverarbeitung bestenfalls als einen kleinen Teil und konzentrieren sich hauptsächlich auf die Modellerstellung. Im Gegensatz dazu konzentriert sich diese Arbeit hauptsächlich auf die Bewältigung der im ersten Absatz genannten Herausforderungen und die Verbesserung der Ergebnisse durch geeignete Vorverarbeitungspipelines, wobei ein Modell, in diesem Fall Clustering, zur Bewertung der Pipelines verwendet wird.

Um die im ersten Abschnitt beschriebenen Herausforderungen zu entschärfen, besteht das allgemeine Ziel darin, genaue Vorschläge für Vorverarbeitungspipelines zu machen, die das Clustering Ergebnis verbessern. Um dies zu erreichen, ist es wichtig, Fehler in den Daten zu identifizieren und die Beziehung von deisen Fehlern zu Vorverarbeitungspipelines zu verstehen. Diese Arbeit leistet einen ersten Schritt in diese Richtung mit einem Konzept, wie eine Wissensbasis geschaffen werden kann, die die Auswirkungen von Vorverarbeitungspipelines auf Daten mit Fehlern genau untersucht und Fehler von Datensätzen identifizieren kann. Um Letzteres zu erreichen, werden Meta-Merkmale evaluiert, da sie eine gute Möglichkeit darstellen, ungesehene Datensätze und deren Fehler zu beschreiben. Eine solche Wissensbasis muss Informationen aus vielen verschiedenen Datensätzen enthalten, um verallgemeinern zu können. Aus diesem Grund werden synthetische Daten verwendet und mit Fehlern versehen, um einen nahezu unbegrenzten Datensatz zur Verfügung zu haben und eine genaue Bewertung vornehmen zu können, welche Fehler von welcher Pipeline gut verarbeitet werden. Durch diese Manipulationen werden die Daten verzerrt, Teile der Daten entfernt, um fehlende Werte zu erzeugen, und Ausreißer zu den Daten hinzugefügt. Aufgrund der im ersten Abschnitt genannten Herausforderungen ist es unmöglich, alle Kombinationen von

Vorverarbeitungsmethoden und ihren Hyperparametern auf einen Datensatz anzuwenden, selbst wenn die Auswahl an Vorverarbeitungsmethoden klein ist. Stattdessen werden Pipelines generiert und während eines Optimierungsprozesses verbessert. Als Optimierung wird eine genetische Optimierung wird verwendet, weil sie eine hohe Flexibilität bietet und gut an die genannten Herausforderungen angepasst werden kann.

Die Auswertung zeigt, dass der Optimierer für die meisten Datensätze eine Vorverarbeitungspipeline findet, die zu einer deutlichen Verbesserung der Clustering Ergebnisse im Vergleich zu den Ergebnissen ohne Vorverarbeitung führt. Die Verbesserung ist am signifikantesten für Daten mit schiefer Verteilung, während Datensätze, die nicht manipuliert wurden, die geringste Verbesserung aufweisen. Darüber hinaus zeigt die Auswertung des Optimierungsprozesses, dass eine gut funktionierende Pipeline relativ schnell gefunden wird, während die anschließende Verbesserung zwar vorhanden, aber vergleichsweise gering ist. Es wird gezeigt, dass das Ersetzen von fehlenden Werten mit dem KNN-Imputer am besten funktioniert. Andere Fehler in den Daten führen nicht zu einer bevorzugten Methode oder Pipeline. Was die Reihenfolge der Vorverarbeitungsmethoden innerhalb einer Pipeline betrifft, konnte kein signifikanter Unterschied nachgewiesen werden. Darüber hinaus wird gezeigt, dass Meta-Merkmale mit den Fehlern in den Daten korrelieren. Dies legt den Schluss nahe, dass es möglich ist, die Notwendigkeit einer Vorverarbeitung und die Identifizierung von Datenmängeln anhand der in dieser Arbeit verwendeten Meta-Merkmale zu bestimmen.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AMI Adjusted Mutual Info Score. 28

ARI Adjusted Rand Index. 23

CHI Calinski-Harabasz Index. 28

CJI Coggins-Jain Index. 29

CME Clustering-oriented Meta Feature Extraction. 28

CRISP-DM Cross Industry Standard Process for Data Mining. 17

CVI Cluster Validity Index. 22

DBCV density-based clustering validation. 29

DBI Davies-Bouldin Index. 28

DI Dunn Index. 29

FCPS Fundamental Clustering Problem Suite. 28

GMM Gaussian Mixture Model. 29

KDD Knowledge Discovery in Databases. 17

KNN K-Nearest Neighbor. 21

LR Linear Regression. 31

MAE Mean Absolut Error. 31

MLE Maximum Likelihood Estimation. 40

MSE Mean Squared Error. 31

NMI Normalized Mutal Information Score. 51

PCA Principal Component Analysis. 18

RI Rand Index. 23

RL Reinforcement Learning. 33

RMSD Root Mean Squared Error. 31

SC Silhouette Coefficient. 28

SVM Support Vector Machines. 31

TPOT Tree-based Pipeline Optimization. 33

WEKA Waikato Environment for Knowledge Analysis. 36

1 Introduction

Data has become increasingly important in almost every aspect of today's lives. In healthcare, for instance, data can be used to improve patient outcomes or predict personalized treatments [RR14]. To leverage these benefits, it is crucial to first collect data and then extract knowledge from it. The knowledge extraction can be accomplished with well-known processes like Cross Industry Standard Process for Data Mining (CRISP-DM) [CCK+00] or Knowledge Discovery in Databases (KDD) [FPS96]. The latter is depicted in Figure 1.1. Fayyad et al. [FPS96] divide the path from raw data to knowledge into five steps (cf. Figure 1.1). The first step is the selection of data. Within this step, the goal of the application is defined, and the target data is created by selecting a relevant subset out of the overall available data [FPS96]. Afterward, the data is preprocessed and transformed. The main objective here is to prepare it for the data mining algorithm. The exact implementation of the sub-steps highly depends on the data and the target of the application. Usually, it includes normalization, outlier removal, missing value imputation, feature selection, and transformation [FPS96]. This part is followed by the selection and execution of a data mining algorithm(s) to find patterns or relationships within the data. A variety of data mining algorithms are available, and the selection is use-case-dependent. One example is clustering, which is used to find patterns in the data. The last steps include interpreting the results acquired in the previous step. It is possible and common in practice to go back to earlier steps and refine the decisions to improve the overall results and insights gained [FPS96].

The preprocessing and transformation step is particularly crucial. Kirchner et al. [KZD16] conducted a study where scientific researchers and users of preprocessing algorithms were asked about how important they consider preprocessing in combination with clustering, and the results showed an average rating of 4.5 (where one means not necessary and five highly important). Secondly, 85% state that even when the clustering algorithm can handle some problems within the data, they would instead use preprocessing in combination with clustering because they think it achieves better results [KZD16].

While the KDD provides a good general overview and instruction on what to do from a high-level perspective, the detailed implementation is still challenging. The main reason is the massive amount of preprocessing methods available and the different options to use them [KZD16]. This variety ranges from very particular methods designed for a single use case [ZB20] to well-established and

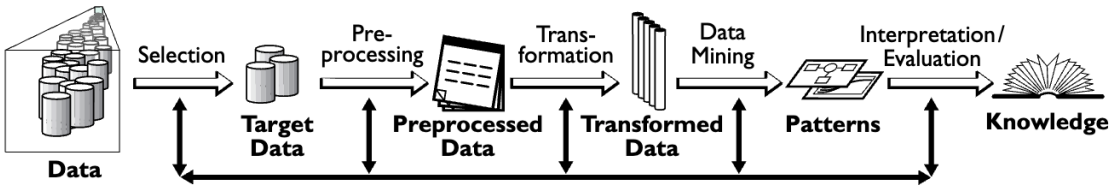


Figure 1.1: Steps of the KDD Process (image extracted from [FPS96])

widely used methods like Principal Component Analysis (PCA) (initially introduced by Pearson [Pea01]). Additionally, these methods often include parameters that must be selected and may significantly affect the result. An example is the parameter specifying how many dimensions PCA should reduce a dataset to. Choosing this parameter too high may result in a similar dataset to the not-preprocessed one, making the preprocessing step needless. On the other hand, picking it too low may result in too much information loss. Therefore, the wrong preprocessing method or parameter choice can affect the clustering result significantly. An additional complicating factor is that certain cases require several preprocessing methods. An example of that is a dataset that contains outliers and missing values, resulting in a preprocessing pipeline instead of a preprocessing method. Within these pipelines, it is unclear what combination of methods is the correct one and in which order they should be executed. To summarize these challenges, it is hard to select suitable preprocessing methods and parameters during the KDD process, especially if multiple preprocessing methods are involved.

The users' experience can partially compensate for the challenges. However, they remain, especially for inexperienced users, and there is a risk of poor results or that finding good preprocessing pipelines will take a long time. This motivates the idea of an automated concept to find suitable preprocessing methods, parameters, and the correct order of execution. Such automation has been developed and is currently being researched for the data mining step of the KDD process and is called AutoML. However, AutoML focuses on finding a good model with its parameters and a good evaluation score [THHL13], and only a few approaches deal with preprocessing. Despite the fact that this thesis does not provide an AutoML approach but focuses on finding good preprocessing method pipelines for clustering algorithms in an automated way, AutoML has some concepts that can be transferred or borrowed to this process. These are the optimization of algorithms and parameters, the use of meta-features, and the concept of a knowledge base.

This thesis does not provide an AutoML approach, including preprocessing, but is inspired by the mentioned high-level concepts. However, some AutoML approaches [ES22; FEF+22; OM16; Ree23] include preprocessing as a small part and are currently the possible solution to solve the described problems in the search for well-performing preprocessing pipelines in an automated way. Since their main focus is the data mining step of the KDD, the following weaknesses of AutoML systems exist in the context of preprocessing: They all are designed to find an end-to-end approach from a dataset to a full machine learning pipeline. This means the effect of preprocessing in this process is not well-researched. Additionally, most are designed for supervised learning or include only one preprocessor per pipeline, or the overall search space of preprocessors and parameters is very limited. Additionally, they often use real-world datasets from sources like the UCI Machine Learning Repository[1] or Kaggle[2]. These datasets are often created or collected for classification tasks instead of clustering, and the class labels, usually considered as ground truth, do not have to correspond with the found clusters [ZZY19]. Another point is that it is primarily unknown what imperfections these datasets have, and therefore, it is hard to validate the effect of specific preprocessing methods.

To mitigate the challenges described above and fill the gaps in the research, the overall goal is to make accurate suggestions for preprocessing pipelines that improve the clustering result. In order to achieve this, it is important to identify data imperfections and to understand the relationship

[1] https://archive.ics.uci.edu/
[2] https://www.kaggle.com/datasets

between data imperfections and preprocessing pipelines. This thesis contributes a first step in that direction with the concept of how a knowledge base can be created that accurately measures the effects of preprocessing pipelines on data with imperfections and can identify data imperfections of datasets. In order to achieve the latter, meta-features are evaluated because they are a good way to describe unseen datasets and their imperfections. Such a knowledge base needs to contain information from many different datasets to be able to generalize. Because of that, synthetic data is used and further manipulated with data imperfections to have an almost unlimited dataset available and to make a precise evaluation of what data imperfection is handled by which pipeline well. These manipulations skew the data distribution, remove parts of the data to create missing values, and add outliers to the data. Because of the named challenges of section one, it is impossible to apply all combinations of preprocessing methods and their hyperparameters to a dataset, even if the selection of preprocessing methods is small. Instead, pipelines are generated and refined during an optimization process. As optimization, Genetic optimization is used because it provides high flexibility and can be well customized to address the named challenges. Additionally, a single clustering algorithm is used to evaluate the preprocessing pipelines, and the parameters of this clustering algorithm are kept constant. The reason for that is that the effect of preprocessing can be better evaluated with a single algorithm and constant parameters.

With the presented concept, the following questions are answered:

Q1: Is there a general effect on the data?

Q2: Are there any patterns of preprocessing methods or hyperparameters?

Q3: Is there a difference in terms of the order in which the preprocessing methods are executed?

Q4: What meta-features are needed to reflect the data characteristics and imperfections

The overall contribution of this thesis can be summarized to (addressed questions provided in brackets):

1. Concept of generating synthetic data and manipulating them have a ground truth with known imperfections in the data
2. Define a large selection of preprocessing methods to assess their effect on the data (Q1)
3. Define a set of meta-features that are likely to reflect the imperfections and changes in the created data (Q4)
4. Propose and optimization approach that focuses solely on preprocessing pipelines
5. Creation of a Knowledgebase that captures the results
6. Evaluate the approach concerning the general effect of preprocessing methods on the data and meta-features (Q1, Q2, Q3, Q4)

The remaining work is structured as follows: Chapter 2 introduces general knowledge about clustering, preprocessing, and meta-features. This is followed by an overview of the related work in chapter 3. After that, the challenges of preprocessing and concept are described in chapter 4, and the general concept of the knowledge base creation and the methodology are provided in chapter 5. Chapter 6 evaluates the approach presented in chapter 5, followed by the conclusion and an outlook for potential future research in chapter 7.

2 Background

In this chapter, helpful background knowledge is provided. It consists of an introduction to the relevant parts of preprocessing (cf. Section 2.1), an explanation of the clustering algorithm used in this thesis (cf. Section 2.2), and an overview of the optimization technique (cf. Section 2.3).

2.1 Preprocessing

Datasets often contain imperfections like missing values or outliers. Without handling these, the result of the model can be significantly worse since low-quality data usually leads to low-quality model results [HPT22, p. 23]. Within this section the preprocessing techniques used in this thesis are explained.

2.1.1 Missing Value Impuation

Missing values are values of features that are not present in the data. A missing value is usually reflected by a blank space or the value *NaN* (Not a number). The reasons why they are missing are dependent on the domain and data collection process. An example is the manual data entry procedures where some values are forgotten [GLH15, p. 59].

Different approaches exist to impute missing values. The most basic approach is to discard all rows of a dataset that contain missing values or to fill them manually [HPT22, p. 57]. The first approach can lead to significant data loss, while the second option may be time-consuming [HPT22, p. 57]. These options should only be considered if no others are available. Other options are divided by García et al. [GLH15] into methods that analyze the relationship between features and ones that do not. For this work, only the ones that do not explore the deeper relationships between features are relevant. Such methods can be relatively simple, like taking the mean of the feature where the missing value is missing, or a bit more complex considering multiple features like the K-Nearest Neighbor (KNN)-imputation technique [GLH15, p. 64].

2.1.2 Normalization

Data can often appear at different scales, and this can influence the result if it depends on distance calculations. Features that have smaller scales tend to have a wide range of values and, therefore, might have a greater weight for certain clustering algorithms [HPT22, p. 113]. To overcome this problem, the data is often normalized to a fixed range (usually $[-1, 1]$) to give all attributes an equal weight [HPT22, p. 113].

2.1.3 Outlier Removal

Outliers are data points that are significantly different compared to the rest of the dataset [HPT22, p. 544]. They can be further divided into three groups: global outliers, conditional outliers, and collective outliers. In this work, only global outliers are relevant for a deeper explanation of the other groups refer to Han et al. [HPT22]. Global outliers are outliers that are significantly different compared to the majority of the dataset. One complex challenge is to define the border of this significant difference [HPT22, p. 545]. Several approaches exist to remove outliers. Since this work is restricted to unsupervised learning, only unsupervised outlier detection methods are applicable. These unsupervised methods assume that the normal points of the data are somewhat distributed such that the outliers stick out [HPT22, p. 550].

2.2 Clustering

Clustering is an unsupervised machine-learning concept that divides the data into subgroups (clusters) based on their similarity [HPT22, p. 444]. There exist several different approaches that measure the similarity in various ways, like distance-based, hierarchically based, or density-based methods [HPT22, p. 448]. This work only uses K-Means [Mac+67], a distanced-based clustering algorithm. In the following, the notation and description of Han et al. [HPT22] are used to describe how K-Means works.

The algorithm takes a dataset $\mathcal{D}$ and the number of clusters k as input, and the output is a set of labels indicating which row of the dataset corresponds to which cluster [HPT22, p. 452]. The following steps are executed to obtain the output:

1. chose random k points of $\mathcal{D}$ and set them as initial cluster centers
2. Assign each element of $\mathcal{D}$ to the cluster where the center has the smallest distance to the element
3. Update the center of the clusters based on the newly formed clusters
4. repeat steps two and three until there is no further change.

2.2.1 Adjusted Rand Index

To evaluate the clustering result, a score (Cluster Validity Index (CVI)) is needed. This score states how good or bad a clustering result is. Since clustering is an unsupervised technique, class labels are not available in the real world. Therefore, such scores often evaluate characteristics of the clusters, such as how dense they are. However, in this work, ground truth labels are available due to how the data is generated and, therefore, can be used to provide an accurate score. The score that is used in this work is the Adjusted Rand Index introduced by Hubert and Arabie [HA85]. It is an extension of Rand Index that eliminates the weakness that the random clusterings sometimes agree by chance [HA85]. The calculation takes two clustering results, U and V, as input. This can be a result obtained by K-Means and the ground truth clusters.

To calculate the Adjusted Rand Index (ARI), first, the Rand Index (RI) is calculated. In order to do this, four types of results are defined among all distinct pairs $\binom{n}{2}$ [HA85], these are: (definition and description taken from Hubert and Arabie [HA85]):

1. Elements of the pair are in the same class in U and in the same class in V
2. Elements of the pair are different classes in U and in different classes in V
3. Elements of the pair are different classes in U and in the same classes in V
4. Elements of the pair are same classes in U and in different classes in V

Types one and two are interpreted as agreements (A) between the two results U and V, while types three and four are considered disagreements (D) [HA85]. The resulting RI is now

$$\text{RI} = \frac{A}{\binom{n}{2}}$$

Where an RI of 1 would state a complete similarity between the results U and V, and a score of 0 would state complete disagreement.

Thus, the RI can be high for a random clustering by chance [HA85]. To overcome this, Hubert and Arabie [HA85] provides an extension of the RI by including the expected RI. The expected RI is calculated with the help of a contingency table of U and V and is the mean RI if the clustering is assigned randomly (for a detailed mathematical explanation, please refer to Hubert and Arabie [HA85]). This results in the following formula for the ARI [HA85]:

$$\text{RI} = \frac{\text{RI} - \text{Expected RI}}{\text{Max RI} - \text{Expected RI}}$$

The ARI has the same upper bound as the RI (1) but is zero for completely random clustering and can be below zero for clustering that is worse than random [HA85].

2.3 Genetic Optimization

Genetic optimization is a framework that is inspired by biology and applies the theory of evolution by natural selection of Darwin to problems in computer science problems [Koz94]. Because of that, the terms of elements and functions within this concept are often borrowed from the field of biology. The terms are introduced on a high level since what they actually are is dependent on the problem. For a mapping from these general terms to the problem solved in this thesis, refer to the section 5.5.

The first term is called individual (sometimes chromosome) and represents a single possible solution to a problem and consists of at least one gene [Mit98]. A gene is a subpart of the individual that can be manipulated during the optimization process. All individuals together result in the population. This population represents the current state of all individuals and, therefore, of the optimization problem [Gre86]. The optimization happens through changes in the population during multiple iterations. These iterations are called generations of the population. [Gre86]. A change can either be a mutation or a crossover [Col99]. A mutation changes at least one gene of an individual,

while a crossover combines two individuals and forms a new one [Col99]. How these changes are implemented and how often they happen during a generation is problem-dependent. After that, the new population is built for the next generation. This is called selection [Col99]. Several different selection approaches exist, and they can be custom-implemented as well. However, this thesis uses an out-of-the-box approach called tournament selection [MG+95]. This selection strategy is described in section 2.3.1. To be able to select good individuals, a quality measure is necessary, called the fitness of an individual. Again, how the fitness is evaluated depends on the problem. In this work, the ARI is chosen as fitness.

An optimization process is finished after the maximum number of generations is reached or one individual achieves the desired fitness.

2.3.1 Tournament Selection

The tournament selection performs multiple tournaments until a fixed size of individuals is selected [CL18]. This fixed size is usually the population size. A tournament selects random k individuals of the current population and chooses the one with the best fitness to be part of the new generation [CL18]. k has to be defined by the user. This is done until the fixed size of individuals, e.g., the population size, is reached [CL18]. It should be noted that an individual can appear in multiple tournaments, and therefore, it might be more than once present in the new generations.

2.4 Meta-Learning

Meta-learning is usually described with the terms *"learning how to learn"* [Van19] or *"learning from the past"* [TFSM23]. Like many solutions in machine learning, this idea is borrowed from human learning, where we generalize from only a few sample experiences to accomplish new tasks [TP98, p. 3]. It is typically in AutoML for Clustering to use meta-learning to generalize from seen datasets to unseen [PDK24]. A meta-learning part consists of two essential parts. The meta-features that are used to characterize a dataset and at least one evaluation measure to state the performance of different solutions for this task [PDK24]. With these parts, a predictive Meta-learner is trained to find suitable solutions for new datasets [PDK24]. This description implies two phases of meta-learning: the offline and online (or application) phases. During the offline phase, datasets are evaluated with different algorithms, hyperparameters, and meta-features are extracted, and well-performing combinations are found [PDK24]. Then, a meta-learner is trained and can provide potentially good configurations for new datasets during the online phase [PDK24].

2.4.1 Meta-Features

Algorithms such as preprocessing methods make certain assumptions about the data [RGS+18]. For example, certain missing value imputation methods assume relationships between missing values [GLH15, pp. 61ff]. Meta-features are used to differentiate between different datasets and their characteristics. Rivolli et al. [RGS+18] state that meta-features are widely used and are often separated into groups, but there is a lack of consistency throughout the literature. To overcome this,

the authors introduce six groups and these groups are used in this work to differentiate between meta-feature types. These groups are (groups, descriptions, and examples strongly inspired or extracted of Rivolli et al. [RGS+18]):

1. **Simple/General:** Meta-features that are simple to understand and can be extracted without high computational costs. Examples are the number of dimensions or the number of missing values
2. **Statistical:** These are meta-features that provide statistics about the data. An example is the standard deviation.
3. **Information-theoretic:** These are mostly for categorical features and classification problems [RGS+18], and show how much *information* is in a dataset. One example would be the entropy of the target class labels.
4. **Model-based:** A model is built on the training data for this meta-feature set. The model's characteristics are then considered meta-features and can provide assumptions about how complex a certain dataset might be. These features are focused on supervised learning as well. An example would be the number of leaves of a decision tree.
5. **Landmarking:** As the two before, landmarking features are made for classification. Instead of taking the model's characteristics as features, the performance of a model is considered. An example would be the best, a random, and the worst node of a decision tree. These may indicate the boundary of classes.
6. **Others:** The authors put all remaining meta-features that appear in the literature in this group. An example is the connectivity that captures nearest neighbor violations of a dataset.

3 Related Work

The following chapter provides an overview of the current research. It is divided into works that apply AutoML or Meta-Learning to clustering only, works that apply AutoML or Meta-Learning to supervised and unsupervised methods but consider preprocessing, and popular libraries in AutoML/MetaLearning.

3.1 AutoML and Metalearning for clustering

Poulakis et al. [PDK24] provides an up-to-date survey about current advances in AutoML for clustering. Based on the focus of the work, the different approaches are categorized into three groups by looking at the main features and differences of the papers, and some are further analyzed to show the current state in this area.

3.1.1 AutoML4Clust

Tschechlov et al. [TFS+21] propose a generic approach for automating clustering analysis.

The generic framework is shown in Figure 3.1 and needs a dataset $\mathcal{D}$, an internal metric $\mathcal{M}$ and a budget l as input. The dataset $\mathcal{D}$ only contains the data and no class labels. Instead of using class labels, the internal metric $\mathcal{M}$ is used to evaluate the clustering result. Budget l determines how long an optimization may take or how many optimization loops are allowed.

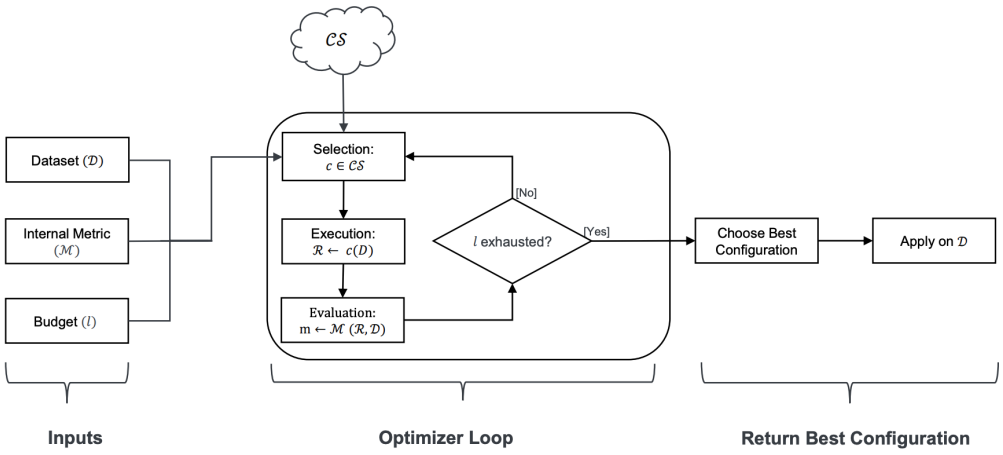


Figure 3.1: Workflow of AutoML4Clust (Figure created by [TFS+21])

With these inputs, a config c containing an algorithm and hyperparameter of the config space $\mathcal{CS}$ is selected, executed, and evaluated until the budget is exhausted (cf. optimizer loop in Figure 3.1). Afterward, the best config is chosen and applied to $\mathcal{D}$.

Tschechlov et al. [TFS+21] evaluate four different optimizers as optimization techniques (cf. optimizer loop Figure 3.1), namely Random [BB12], Bayes [THHL13], Hyperband [LJD+18], and BHOB [FKH18]. This evaluation is done with 24 synthetically generated datasets and five real-world datasets [TFS+21]. The real-world datasets are a selection from the UCI Machine Learning repository[1]. Due to the popularity and good runtime behavior, only k-centered clustering algorithms are used [TFS+21]. These are K-Means [Mac+67], MiniBatch-K-Means [Scu10], K-Medoids [RK87], and GMM [Bis06]. While optimizing with internal CVIs, they use the Adjusted Mutual Info Score (AMI) to validate the results.

Their results show that no better results are achieved after 60 optimization loops, but during the 60 loops, a noticeable improvement is achieved with an AMI of over 90%. In addition, they achieve up to 437 times better runtime for synthetic data sets and 276 times better runtime for real-world data sets compared to exhaustive searches.

3.1.2 AutoCluster

Liu et al. [LLT21] propose an approach where they use Clustering-oriented Meta Feature Extraction (CME) for meta-learning, in combination with multiple CVIs, hyperparameter optimization, and an ensemble clustering to find the best algorithm and corresponding hyperparameters for a dataset.

The meta-features are a combination of five new meta-features proposed by Liu et al. [LLT21] and 19 existing ones by Li et al. [LWWT19]. While the 19 features of [LWWT19] are statistical meta-features (cf. Section 2.4.1), the five new proposed ones focus primarily on cluster structure and support the algorithm choice. To measure the quality, they use three internal CVIs, namely the Calinski-Harabasz Index (CHI), Davies-Bouldin Index (DBI), and Silhouette Coefficient (SC). The data is split into 150 *prior* datasets and 33 *new* datasets. The *prior* datasets are used to suggest better algorithms and CVIs. They use ground truth labels and the external CVI ARI to measure the performance and ensure correct selection. With all the measures, meta-features, and knowledge from prior datasets combined, a suggestion with promising algorithms is made, and the hyperparameters of these algorithms are further tuned with a grid search. These results are then taken to construct an ensemble model through Majority Voting [LLT21].

The datasets for evaluation of the described approach are taken from different places. The 150 *prior* datasets are a selection of OpenML [VVBT14], with maximal 5000 samples and 50 features. The 33 benchmark datasets are an assembly of different clustering benchmark datasets from clustering basic benchmark [FS18][2], the Fundamental Clustering Problem Suite (FCPS) [Ult05], and clustering benchmarks from a widely used GithHub repository[3]. One out of six possible clustering algorithms is suggested. The possibilities are: K-Means [Mac+67], Affinity Propagation [FD07], Mean Shift [FH75], DBSCAN [EKSX+96], Agglomerative clustering, and BIRCH [ZRL96]. The scikit-learn [PVG+11] library is used to implement the algorithms mentioned. Technically, they apply

[1] https://archive.ics.uci.edu/datasets

[2] http://cs.uef.fi/sipu/datasets/

[3] https://github.com/deric/clustering-benchmark.

preprocessing before evaluating the datasets, but all preprocessing steps are applied to every dataset and are not further discussed. Since the methods are removing missing values, one-hot encoding, and z-score standardization, the main focus is to make every dataset runnable by the algorithms.

The evaluation compares the performance of the named algorithms with default parameters and for K-Means with different numbers of clusters (ranging from 2 to 20). AutoCluster performs best at 15 of the 33 datasets and is often close to the best method while achieving an overall average ARI of 0.776 [LLT21].

3.1.3 ML2DAC

ML2DAC is an approach that uses meta-learning to select a CVI, a clustering algorithm, and suitable hyperparameters [TFSM23].

Treder-Tschechlov et al. [TFSM23] divide the learning phase into five steps and require the availability of ground truth labels. The first step calculates the meta-features. These have a wide variety ranging from simple ones like the number of features, statistical meta-features, information theory-based, and landmarking ones. The second step evaluates the given datasets. This is done with a Bayesian optimizer. To decide which CVI is the best to evaluate similar datasets during the application phase, the ground truth labels are used to calculate the ARI. Next, the best CVI is selected by comparing it with the ARI and choosing the one that correlates most with it. The correlation is calculated with the Spearman rank correlation coefficient. After that, they train a classifier to predict suitable CVIs for unknown datasets. This is done by treating the meta-features as input set and the CVIs as class labels, therefore a multi-class classification is done. [TFSM23]. Lastly, the gained knowledge is stored in a knowledge base. It consists of the meta-features, the evaluations with their scores (ARI and CVIs), and the classification model described in step four.

In the application phase, a new dataset is given, and the goal is to provide a good configuration consisting of an algorithm and hyperparameters. This process can be divided into five steps as well. First, a CVI is selected with the trained model from the learning phase. Secondly, warmstart configurations are chosen based on the knowledge base. The third step reduces the available algorithms to the ones of the warmstart configurations. Multiple optimizer loops are performed to get the best configuration based on the performance of selected CVI.

In their evaluation, they generate 76 synthetic datasets to build the knowledge base. This is achieved by using different generation methods of the scikit-learn [PVG+11] library that generates Gaussian distributed data, moon-shaped data, and circle-shaped data. They evaluate with samples sizes $n \in \{1000, 5000, 10000\}$, feature sizes $f \in \{10, 30, 50\}$, and cluster $k \in \{10, 30, 50\}$. The moon- and circle-shaped data have, per definition, only two features, but they apply additional noise to them $r \in \{0, 0.01, 0.05, 0.1\}$. Additionally, 22 real-world datasets from a clustering benchmark[4] and the UCI repository[5] are used in the application phase. The selected CVIs are the DBI, SC, CHI, Dunn Index (DI), Coggins-Jain Index (CJI), density-based clustering validation (DBCV), and COP Index. They use nine clustering algorithms with different characteristics, for example, K-Means [Mac+67], Gaussian Mixture Model (GMM) [Ras99], and DBSCAN [EKSX+96]. They

[4] https://github.com/deric/clustering-benchmark.
[5] https://archive.ics.uci.edu/datasets

evaluate ML2DAC against several other solutions, namely AutoCluster [LLT21], AutoML4Clust [TFS+21], and AutoClust [PDK20]. ML2DAC achieves better and more consistent accuracy while needing fewer optimizer loops on most synthetic datasets compared to the named baselines. The same results are achieved on the 22 real-world datasets.

3.2 AutoML and Meta-Learning considering Preprocessing

This section discusses different approaches that use preprocessing as part of their AutoML approach. Since not much research is generally published in this area, in particular, there is only one work that combines AutoML for clustering with preprocessing, and there is no work that focuses exclusively on AutoML for preprocessing. Supervised methods are included as well as unsupervised methods. The weight of the preprocessing part varies significantly across the papers.

3.2.1 Preprocessor Selection for Machine Learning Pipelines

Schoenfeld et al. [SGP+18] conduct an empirical study on a range of preprocessing methods combined with supervised learning classifiers to evaluate if preprocessing improves the accuracy of the ML Pipeline.

They select eight preprocessing algorithms: Min-Max-Scaler, Standard-Scaler, Select Percentile, Principle Component Analysis, Fast Independent Component Analysis, Feature Agglomeration, Polynomial Features, and Radial Basis Function. As classification algorithms, they test Random Forest Classifier, Logistic Regression, K-nearest Neighbor Classifier, Perceptron, Support Vector Machines, and Gaussian Naive Bayes. The hyperparameters are kept to their default values for all algorithms. As pipeline construction approaches, they choose three-stage pipelines as a baseline and four-stage pipelines as an experiment. The baseline consists of a missing value imputation algorithm, one-hot encoding if needed, and a classification algorithm. The experiment pipelines are built similarly but contain an additional preprocessing step before applying the classifier. All combinations are tested, resulting in 1152 baseline pipelines of length three and 9216 pipelines of length four, including the named preprocessing methods [SGP+18]. For the benchmark dataset, 192 datasets from the OpenML [VVBT14] are used. The results show that 79.1% of all benchmark pipelines have a shorter runtime during training when preprocessing is applied, and 80.2% have a shorter testing runtime [SGP+18]. The causes of this are the feature and dimensionality reduction preprocessors. Accuracy-wise, they observe a reduction when applying the preprocessing overall, with 69.4% of the pipelines performing worse. Still, if only pipelines with the best accuracy are considered, 91.1% of them contain a preprocessor.

Concerning the meta-learning, Schoenfeld et al. [SGP+18] extract 18 simple, eight statistical, one information-theoretic, and 14 landmarking meta-features from all datasets. With these meta-features, a random forest classifier is trained to predict if a given preprocessor improves the accuracy. They achieve an accuracy of 62.6% with this approach.

3.2.2 Auto-prep

Bilal et al. [BAI+22] provide an interactive and data-driven tool to help data scientists decide what preprocessing method is effective for a given transformation task within a preprocessing pipeline. Their approach covers specific areas of preprocessing, namely missing value imputation, qualitative data encoding, feature selection, feature scaling, and feature extraction. Instead of generating full pipelines, they choose a step-by-step approach where every preprocessing area is handled individually and in a fixed order. They approach each area with three steps: show the user what is wrong with the data (e.g., missing value percentage), evaluate specific methods (e.g., mean imputation), and then show all results to the user while suggesting the best one. The user can overrule the choice and can select the final method per area but can not control the methods evaluated or the order in which they are evaluated. These methods are mean, median, most frequent, KNN, and Multiple Imputation for missing values. Qualitative data encoding uses label encoding and one hot encoding. For feature selection, they use the backward elimination procedure; for scaling, the user can choose between normalize, standardize, or nothing, and feature extraction wise, the user can choose if they want to use PCA or not.

Bilal et al. [BAI+22] verify their approach on the most used datasets from different collections like UCI[6], OpenML [VVBT14][7], and Kaggle [8], this results in a total of six classification and five regression datasets with different characteristics. A detailed list can be found in the original paper [BAI+22]. As machine learning models, Linear Regression (LR), Support Vector Machines (SVM), and decision trees are evaluated. They always compare their approach to fully manual preprocessed data. The results of these algorithms are then measured with a confusion matrix and an f1 score for the classification algorithms and R Squared, Mean Absolut Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSD) for the regression algorithms.

In most cases, they achieve higher scores with the Auto-prep approach than with manual preprocessing, sometimes similar scores, and on rare occasions, they perform worse. Additionally, they claim that they make the whole process less tedious, especially for novelist data analysts [BAI+22].

3.2.3 TPE-AutoClust

ElShawi and Sakr [ES22] propose a framework for automated clustering. Their approach consists of three phases: meta-learning, optimization, and clustering ensemble construction, and tries to find the optimal solution considering different CVIs, clustering algorithms, and preprocessing methods [ES22]. The hyperparameters of the clustering and preprocessing methods are optimized as well.

Figure 3.2 shows the general concept and flow of TPE-AutoClust. They are starting with a meta-feature extraction to find similar datasets, going over to do a warmstart genetic optimization, and ending with a clustering ensemble that evaluates the solution with three internal CVIs.

A knowledge base is needed to exploit meta-features and build warmstart configurations. ElShawi and Sakr [ES22] build them by extracting 28 different meta-features. These are from the areas of statistics, information theory, and landmarking. For data collection, they choose a combination of

[6] https://archive.ics.uci.edu/datasets
[7] openml.org
[8] https://www.kaggle.com/datasets

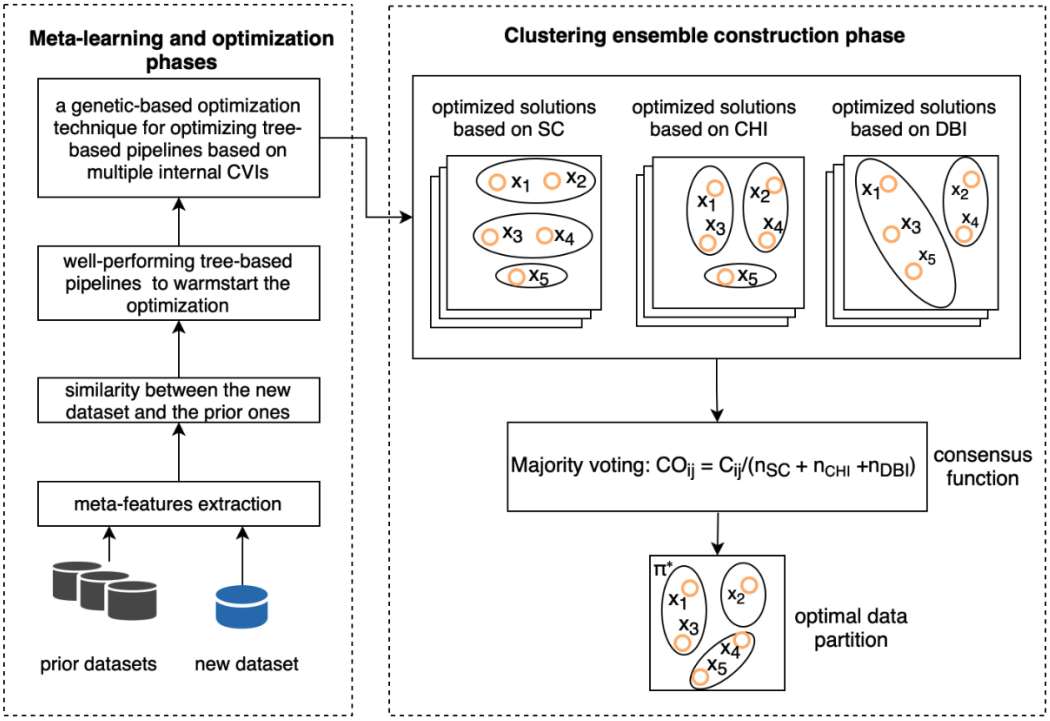


Figure 3.2: TPE-AutoClust Framework (Figure created by [PDK24])

synthetic and real-world datasets, and their knowledge base contains information from 118 datasets. To find out which configurations are best, each of the 118 data sets is optimized with a genetic optimizer, and the results are compared with the external CVI ARI. This is done until the time budget of 15 hours is exhausted [ES22].

For a new dataset, the 50 most similar datasets of the knowledge base are selected and then further optimized with a genetic tree-based optimization technique [ES22]. This is done with the Python package DEAP [FDG+12]. The optimizer runs for every CVI (CHI, DBI, and SC) separately and minimizes the CVI and the number of operators in the pipeline at the same time. After that, they select the best five pipelines per CVI and apply a majority voting to obtain the best clustering results (cf. Figure 3.2).

ElShawi and Sakr [ES22] evaluate this approach on 12 real-world datasets and 15 synthetic datasets obtained from OpenML [VVBT14][9] and the well-known clustering benchmark set[10]. As clustering algorithm, they use K-Means [Mac+67], Affinity Propagation [FD07], Agglomerative clustering, DBSCAN [EKSX+96], Mean Shift [FH75], and Spectral clustering, and as preprocessing methods the use StandardScaler, SimpleImputer, KNNImputer and PCA. They compare their approach to an approach where they only optimize one CVI and keep the default parameters of the methods,

[9] openml.org
[10] https://github.com/deric/clustering-benchmark.

to an exhaustive search approach using grid search, and to other AutoML Frameworks namely cSmartML by [ELS21] and AutoML4Clust [TFS+21], the latter one is presented in Section 3.1.1. TPE-AutoClust and the other frameworks were evaluated with a 10- and 30-minute budget.

Out of the 12 synthetic datasets, TPE-AutoClust performs best on 10, while the cSmartML performs best on two. On real-world datasets, the result is a bit more mixed. For one dataset, several algorithms achieve an ARI of 1.0 (perfect score). Of the remaining 14, TPE-AutoClust achieves ten times the best core, two times the exhaustive search is best, and the other two are best handled by cSmartML [ES22].

3.2.4 Auto-FP

Qi et al. [QPHW24] conduct an experimental study on automated feature preprocessing for tabular data in the context of supervised learning [QPHW24]. Their main contributions are the indication of the importance of feature preprocessing, the categorization of optimization algorithms, and different experiments that focus on the selection of optimization techniques in varying settings. Finally, they compare their work to existing AutoML approaches.

They choose seven preprocessing methods from the Scikit-learn library [PVG+11], where all the chosen methods can be added to the group of scaling methods. Namely, these are StandardScaler, MaxAbsScaler, MinMaxScaler, Normalizer, PowerTransformer, QuantileTransformer, and Binarizer. To explore whether preprocessing is important in general, they generate 2800 pipelines with a length of up to four, evaluate them on four datasets with a linear regression model, and compare them to non-processed results. No information has been given on how the pipelines are constructed and how the parameters are chosen. They show that there always exists a pipeline that outperforms the case with no preprocessing, but several preprocessing pipelines perform similarly or even worse. Secondly, they compare their best-performing pipelines against the AutoML tool Tree-based Pipeline Optimization (TPOT) [OM16] and can show that they perform better on every dataset as well while producing longer pipelines, which is an indication that longer pipelines improve the accuracy [QPHW24].

The second experiment tries to clarify if any meta-features indicate the need for preprocessing. To evaluate that, they extract 40 meta-features that are used in Auto-Sklearn [FKE+15b] and are from the areas of basic, statistical, information-theoretic, and landmarking [QPHW24]. The binary label, which indicates that preprocessing is promising, is created by first evaluating the dataset without preprocessing (by the authors defined as approach A) and then with 200 randomly selected pipelines (by the authors defined as approach B). If $B - A > 1.5\%$ the label is 1 (preprocessing helps) and if $B - A < -1.5\%$ the label is 0 (preprocessing does not help) [BAI+22]. With these labels gathered, they train a decision and try to obtain characteristics or rules that may imply preprocessing or certain preprocessing methods. However, they could not find any reliable meta-features to predict the effectiveness of preprocessing [QPHW24].

The evaluation of the optimization techniques (by Qi et al. [QPHW24] called search algorithms) first groups all algorithms into five groups. These are: *traditional algorithms* like random search, *surrogate-model-based algorithms* like SMAC [HHL11], *evolution-based algorithms* like tournament evolution [RAHL19], *Reinforcement Learning (RL)-based algorithms* like REINFORCE [Wil92], and *bandit based algorithms* like BHOB [FKH18], this results in 15 optimizers that are

evaluated. They are all evaluated with 45 datasets from an AutoML challenge site[11], Kaggle [12], and an AutoML Benchmark collection [GLT+19]. As classifiers, they chose Logistic Regression, XGBoost, and Multi-layer Perceptron. Additionally, different time budgets are defined: 60, 400, 600, 1200, 1800, and 3600.

The first evaluation keeps all default parameters of the preprocessing methods and solely focuses on which optimization technique works best. For this, all 45 datasets are evaluated with all 15 optimizers. The results show that evaluation-based algorithms generally perform better than others, while others struggle because of complexity or bad initialization [QPHW24]. Additionally, no frequent pattern of preprocessors has been found. The other experiments perform hyperparameter optimization. They differentiate between a relatively small and big configuration space and two approaches to initializing the pipelines. The first approach considers every preprocessor with different parameters as whole different preprocessor and initializes every possible combination, these are then evaualuted (called one-step by Qi et al. [QPHW24]. The second variant (called two-step by Qi et al. [QPHW24]) initializes every preprocessor with a random parameter, evaluates them with a short time limit (60s), and then repeats these two steps until the budget is exhausted [QPHW24]. The results show that the one-step approach is better for a small configuration space because it does more exploration. For a larger configuration space, the two-step approach is more successful [QPHW24]. They also note that the pipeline initialization needs more investigation. Lastly, they compare their approach with the existing TPOT [OM16] tool but turn off every optimization in TPOT that is not preprocessing related. The results are close, but Auto-FP is often better by a small margin.

3.2.5 Meta-Learning Based Approach for Automated Preprocessing for clustering

Tanvir [Tan22] extends in their master-thesis the existing cSmartML [ELS21] approach with preprocessing. Their approach consists of the two typical phases for building a knowledge base. Within the offline phase, they extract meta-features, then apply clustering and calculate a score of multiple CVIs. They use DEAP [FDG+12] to optimize the clustering algorithm and hyperparameter selection during this phase. After that, they store the three best-performing configs per dataset. The optimization step without preprocessing is done to have a baseline and validate whether the preprocessing improves the accuracy. Secondly, they build preprocessing pipelines of one and two lengths. So, each preprocessor is applied once to every dataset and every combination of two preprocessors. Afterwards, the clustering is applied again. Every result is stored in the knowledge base. In the online phase, the meta-features are extracted from a new dataset, the nearest neighbors are calculated, and the algorithm and hyperparameters are suggested.

To build the knowledge base, they use 112 datasets from a clustering benchmark collection [13]. They use seven different techniques, preprocessing method-wise: KBin, L1, L2, MinMax, Z-score, mean imputation, and median imputation.

[11] https://automl.chalearn.org/data
[12] https://www.kaggle.com/datasets
[13] https://github.com/deric/clustering-benchmark

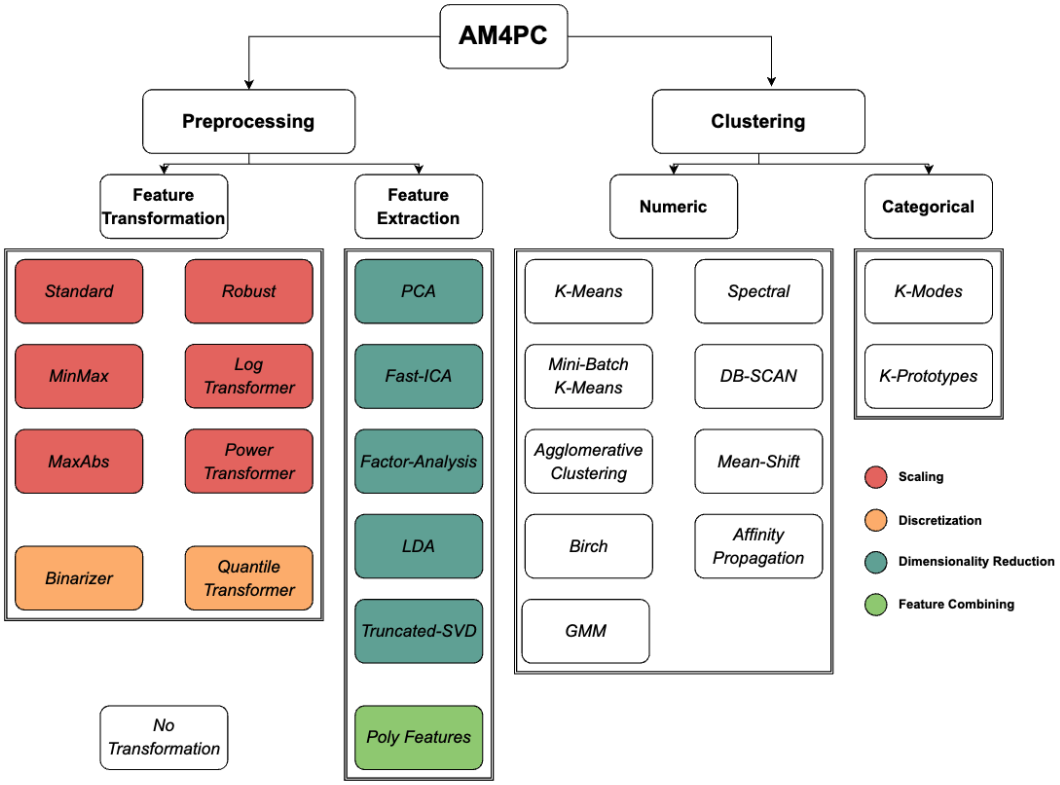


Figure 3.3: Methods used by Reed [Ree23] (Figure created by [Ree23])

The result of the offline phase is that only one out of the 112 datasets performs better without preprocessing being applied. Secondly, they compare their approach directly to cSmartML [ELS21] and show that the described approach performs better for 10 out of 16 datasets.

3.2.6 Analysis and Integration of Data Preprocessing Steps in AutoML for clustering

The AutoML4Clust [TFS+21] (see Section 3.1.1) is extended with preprocessing by the Bachelor Thesis of Reed [Ree23]. They extend the work by adding a benchmarking framework, data formatting steps, preprocessing methods, a categorical clustering algorithm, and an option for multi-objective optimization and sampling Reed [Ree23].

They use 14 preprocessing and eleven clustering methods, as depicted in Figure 3.3. These methods are evaluated on synthetic and real-world datasets. To create the synthetic datasets, they use Scikit-learn [PVG+11] methods *make_blobs*, *make_circles*, and *make_moons*, with a sample size between 100 and 1000, a feature size from 2 to 50, a standard deviation between 0 and 2, and the

number of clusters are between 2 and 50. Additionally, 14 datasets of the UCI repository [14] are evaluated. Every dataset is cleaned before being handed over to the framework. This includes imputation of missing values and one-hot encoding or label encoding.

The pipelines are built with the restriction that no more than one preprocessing step can be involved, and the budget per optimization is limited to 100 loops. The external measure AMI is calculated to ensure the evaluation reflects the ground truth.

Applied to the synthetic datasets, the approach performs 2.05% worse than without preprocessing (just AutoML4Clust). On the other hand, the real-world dataset benefits significantly from preprocessing, and the AMI improves by 24.6% on average [Ree23]. They also show that the preprocessing methods are relatively evenly distributed apart from the *binarizer*. That suggests an extensive selection of preprocessing methods is reasonable [Ree23].

3.3 Libraries

Besides the named published research, libraries exist to automate machine learning and sometimes preprocessing. They often have a paper as their origin but are grouped in this section because they have matured into widely used libraries. This section introduces them while mentioning their shortcomings for the described tasks.

3.3.1 Auto-Weka

Auto-Weka is based on Waikato Environment for Knowledge Analysis (WEKA) [HFH+09] and provides an optimization framework that helps find good algorithms and corresponding hyperparameters. It only works for classification and regression and provides no preprocessing option. Internally, Bayesian Optimization is used to find a good configuration for a given dataset. While WEKA and Auto-Weka have been written in Java, there exists a Python wrapper [15]; however, it does not seem well maintained.

3.3.2 Auto-SKlearn

Auto-SKlearn is based on Scikit-learn Pedregosa et al. [PVG+11]. The library was initially presented by Feurer et al. [FKE+15a], later Feurer et al. [FEF+22] then presented extensions for version 2.0. They support classification and regression algorithms and preprocessors. They support more than 20 preprocessors, but not all are applicable for every type of data since some solely focus on text or categorical data. A complete list can be found in the GitHub Repository [16]. All preprocessors can be divided into two groups: *data preprocessors* and *feature preprocessors*. A *data preprocessor* can be missing value imputation or one-hot encoding, while a *feature preprocessor* could PCA. Again, a complete list is provided here [17]. It should be noted that although there can be several *data*

[14] https://archive.ics.uci.edu/datasets
[15] https://github.com/automl/pyautoweka
[16] https://github.com/automl/auto-sklearn/tree/master/autosklearn/pipeline/components
[17] https://github.com/automl/auto-sklearn/tree/master/autosklearn/pipeline/components

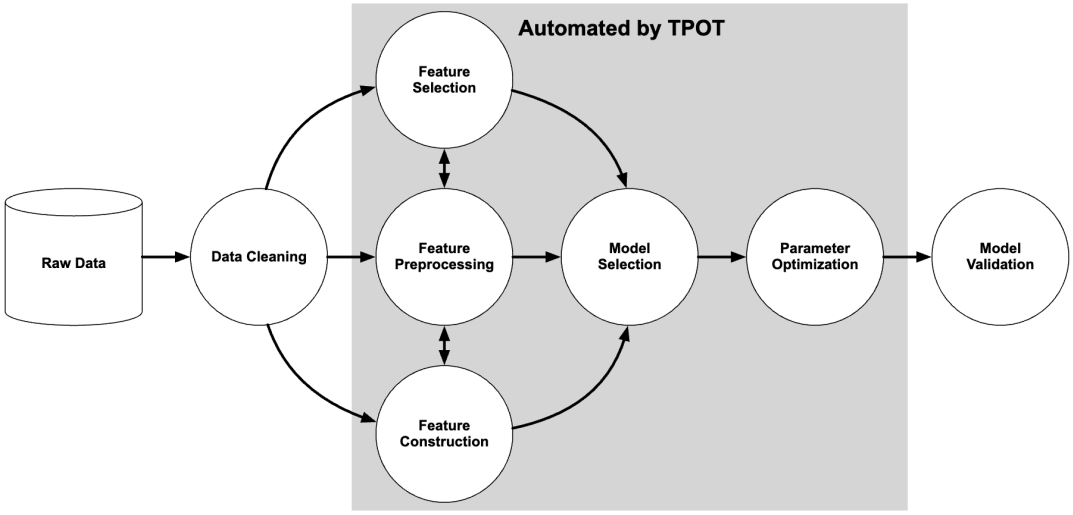


Figure 3.4: TPOT Overview (Figure created by [OBUM16])

preprocessors per pipeline, only one *feature preprocessor* per pipeline is supported. The order in which the preprocessors are executed can not be changed. Besides the preprocessor, these pipelines contain a single classifier or a regression algorithm and are optimized with SMAC [LEF+22].

3.3.3 TPOT

TPOT [OBUM16] is a tree-based genetic optimization approach that supports multiple preprocessors in combination with classification or regression tasks.

As depicted in Figure 3.4, the automated part can consist of several different preprocessing kinds, like feature selection, feature preprocessing, and feature construction. After that, they are evaluated with a model like SVM.

Under the hood, they use the DEAP [FDG+12] library to implement their optimization process. The result is a highly customizable library that can be altered to specific needs. The most important decisions that still need to be made are the choices of hyperparameters for the genetic optimizer.

3.4 Summary

No research focuses solely on optimizing preprocessing pipelines for supervised or unsupervised learning. Existing work contributes to optimizing a model with its corresponding hyperparameters or building a combination of preprocessing and model optimization. Due to the even larger search space, related work often contains a limited selection of preprocessing methods and limits the number of preprocessing steps per pipeline. In addition, there is usually no focus on the order of the preprocessing steps.

4 Challenges of Preprocessing and Knowledge Base Creation

Finding an exemplary configuration of preprocessing methods and hyperparameters for a given dataset is hard. Additionally, creating a knowledge base that reflects the effect of preprocessing includes difficulties. This chapter introduces the challenges that arise during these processes.

4.1 Challenges of Preprocessing

This section describes the challenges that arise during preprocessing in general.

4.1.1 Selection of Preprocessing Methods

There is a wide variety of different preprocessing methods ranging from particular techniques that are only applied to single-use cases to very popular ones like PCA for dimensionality reduction [MR93]. Finding the correct preprocessing methods for a given dataset may be difficult.

4.1.2 Selection of Hyperparameters

Almost every preprocessing method has hyperparameters that change its behavior. The differences in impact on the method and options available vary a lot. An example of the effects of hyperparameters is the parameter `n_components` of PCA. Choosing this parameter too big may result in an almost unaffected dataset. On the other hand, defining it as too small may result in too much information loss and worsen the result. The different options include parameters with infinite values like the tolerance of the internal solver of PCA or only a limited selection, as with PCA, the number of components to which a data set is reduced. Selecting the best or even well-functioning parameters is a difficult task, especially for inexperienced users.

4.1.3 Defining the Order of Preprocessing Methods

It is unclear what effects the preprocessing methods have on each other. Therefore, the order of preprocessing methods is another part that extends the search space. An example is that scaling and dimensionality reduction might affect each other, but there is no clear evidence of what to do first. Additionally, there might be some restrictions because some preprocessing methods have to be executed before others work. An example would be the missing value imputation. Several other preprocessing methods can not handle missing values, and therefore, this has to be executed first. This adds another layer of complexity.

4.2 Challenges of Knowledge Base Creation

The challenges 4.1.1, 4.1.2, and 4.1.3 also apply to the knowledge base creation at a larger scale. Instead of finding the correct set of preprocessing methods with their hyperparameters for a given dataset, this has to be done for various datasets with different characteristics. Finding a suitable restriction for a good set of preprocessing methods, hyperparameters, and hyperparameter values is difficult. Additionally, the following challenges arise:

4.2.1 Risk of Invalid Pipelines

The named problems of challenge 4.1.2 introduce the additional challenge of producing invalid pipelines in the offline phase of knowledge base creation. The more hyperparameters and hyperparameter values are added, the higher the potential to create an invalid pipeline. The first reason for that is the implementation of preprocessing methods from Scikit-learn [PVG+11]. An example is the `n_components` parameter of PCA. It can be set to `mle` (Maximum Likelihood Estimation (MLE)) to use an automatic way presented by Minka [Min00] to guess this parameter. If this is the case, the parameter *svd_solver* is restricted [PVG+11]. Such restrictions need to be defined in the logic of the creation process to prevent invalid pipelines. The second reason for potentially invalid pipelines is the value range of hyperparameters. If, for example, an outlier removal method is defined with parameters that are too aggressive, the preprocessed datasets may result in too few samples to cluster.

4.2.2 Challenge of Data Selection

The offline phase must include a variety of datasets to create a knowledge base that provides good suggestions for preprocessing. It is not clear how different these datasets should be or how many should be included to achieve a good generalization.

4.2.3 Large Search Space

Challenge 4.1.1, 4.1.2, and 4.1.3, combined results in a huge search space. Considering the selection of preprocessors, the different hyperparameters, and the order, even a small selection of, for instance, three different preprocessors, two different hyperparameters with five different values per hyperparameter results in:

$$5^2 = 25$$

possible configurations per preprocessor and considering three unique preprocessors:

$$(3x25)^n = 75^n$$

for a pipeline with a max length of n. If the max length of the pipeline is 3 and the minimal length is 1, the number of all possible pipelines is:

$$\sum_{n=1}^{3} 75^n = 427.575$$

In reality, there are more preprocessors with more hyperparameters, making an exhaustive search impossible. Even with more sophisticated optimization algorithms, finding an optimal pipeline is very hard to accomplish.

4.2.4 Meta-Feature Selection

Different meta-feature groups and within these groups, different meta-features exist (cf. Section 2.4.1). There is no clear indication of which meta-features should be included or excluded to find the right balance between a good description of the datasets and preprocessing methods and too many meta-features that extend the calculation process and may blur the resulting representation of a dataset.

5 Knowledge Base Creation for Preprocessing Pipelines

This chapter presents the general concept, and its sub-parts while addressing the challenges of chapter 4.

5.1 General Concept

A general concept is needed to research how preprocessing affects the clustering result, what meta-features could be used to capture these effects, and how a knowledge base could be created. This conception is introduced in this section, and the single steps are described in detail in the following sections.

The whole process is divided into five steps to achieve the named objectives while handling the presented challenges (cf. Chapter 4). These steps are depicted in Figure 5.1. The process starts with meta-feature selection to form the meta-feature space. This space consists of a selection of meta-features potentially helpful in achieving the goals (Step 1 Figure 5.1). This step has to come first because the meta-features are needed in the following steps. The second step is the generation of data. Afterward, the data is modified to introduce data imperfections in a controlled way (Step 3 Figure 5.1) such that it can be clearly evaluated what meta-features indicate what data imperfections and which preprocessing pipelines may help reduce these imperfections. Fourthly, the configuration space is defined. It includes all available preprocessors with their hyperparameters and the possible values of these hyperparameters. The most significant step is step five. Here are preprocessing pipelines created and refined during optimization cycles to collect well-functioning pipelines.

As depicted in Figure 5.1, the knowledge base is created in such a way that it can reflect all changes during the mentioned steps. This creation includes general characteristics of the dataset, such as a unique ID, the sample size, the number of clusters, the number of features, the standard deviation of the clusters and what imperfections exist in the data. Additionally, for steps two to five, the change in the data, the meta-features, and a score is saved, and after step five, the best-performing pipeline.

A clustering algorithm is needed to calculate a score. This work focuses on preprocessing, and a single clustering algorithm is used. Due to its high popularity and wide use, the decision for the clustering algorithm is K-Means [Mac+67].

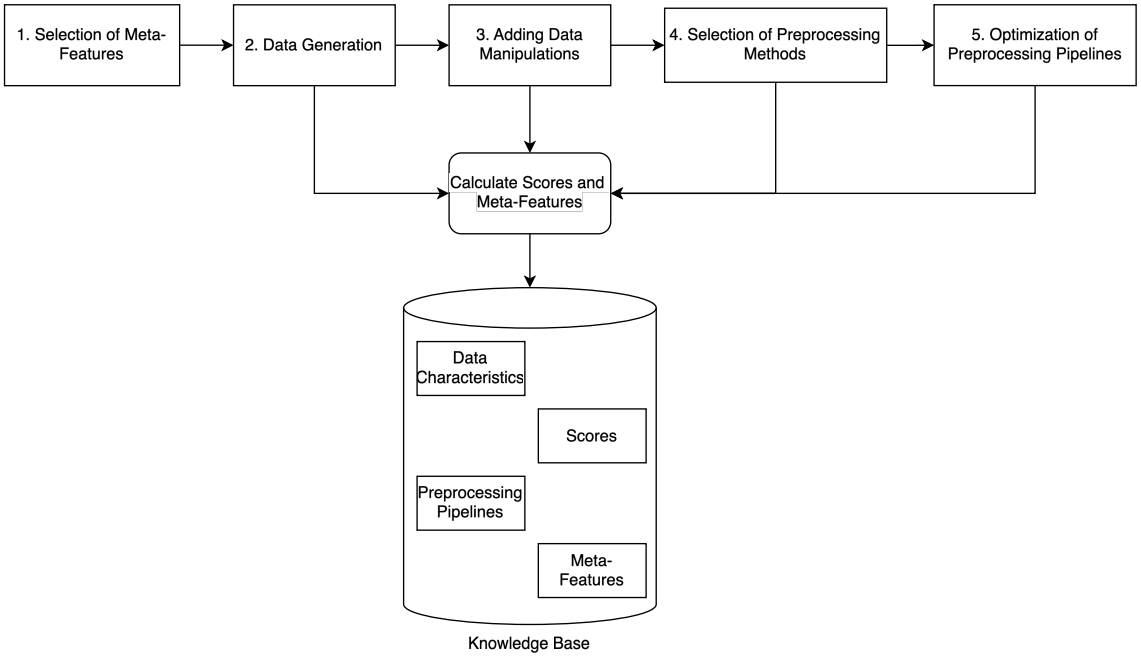


Figure 5.1: General Approach

5.2 Selection of Meta-Features

This section covers the first step of the general concept (cf. Figure 5.1), the meta-feature selection. As described in challenge 4.2.4 (cf. Section 4), many different meta-features exist. However, not all are applicable for clustering analysis. The main exclusion criterion is if the meta-features require class labels. Additionally, they should be widely used and are already implemented.

Alcobaça et al. [ASR+20] present a paper that aims to standardize meta-feature extraction. This paper has matured into a library [1] that conforms to all requirements and provides many possible meta-features. A selection from this library is used for this thesis and is presented in Table 5.1. The names and descriptions are taken from the official documentation [2]. The selection mainly focuses on statistical meta-features and some from the general and complexity groups (cf. Section 2.4.1). Some examples of these are the number of features for the general group, the standard deviation as a statistical meta-feature, and the average number of features per dimension for the complexity group. Other groups like model-based or landmarking are excluded because they either require class labels, are expensive to compute, have additional hyperparameters, or do not fit the data. Additionally, one custom meta-feature is added, the indication if missing values exist (has_missing_values). This meta-feature is needed to identify datasets that contain missing values. The value of this meta-feature is calculated by checking if *NaN* or blank spaces occur in a dataset.

[1] https://pymfe.readthedocs.io/en/latest/
[2] https://pymfe.readthedocs.io/en/latest/auto_pages/meta_features_description.html

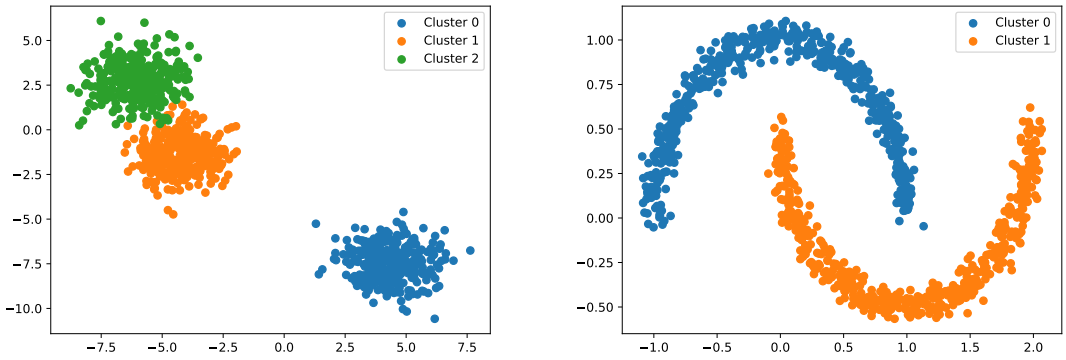
Group	Name	Description
General	nr_attr	Total number of attributes
General	nr_inst	Number of rows
Statistical	cor	Absolute value of correlations of distinct dataset column pairs
Statistical	cov	Absolute value of the covariance of distinct dataset column pairs
Statistical	eigenvalues	Eigenvalues of the covariance matrix
Statistical	iq_range	The interquartile range of each attribute
Statistical	kurtosis	The kurtosis of each attribute
Statistical	mad	The median absolute deviation
Statistical	max	Maximum value of each attribute
Statistical	mean	Mean value of each attribute
Statistical	median	Median value of each attribute
Statistical	min	Minimal absolute deviation
Statistical	nr_cor_attr	Number of distinct highly correlated pair of attributes
Statistical	nr_norm	Number of attributes normally distributed
Statistical	nr_outliers	Number of attributes with at least one outlier
Statistical	range	Range of each attribute
Statistical	sd	Standard deviation of each attribute
Statistical	skewness	The skewness of each attribute
Statistical	sparsity	The sparsity of each attribute
Statistical	t_mean	The trimmed mean of each attribute
Statistical	var	The variance of each attribute
Complexity	t2	Average number of samples per feature
Complexity	t3	Average number of PCA dimensions per points
Complexity	t4	Ratio of PCA dimensions to original dimensions
Complexity	wg_dist	The weighted distance that captures how dense or sparse is the example distribution.
Other	has_missing_values	Indication if missing values exists

Table 5.1: Selection of Meta-Features

Most meta-features can be calculated per attribute. However, calculating them per attribute would result in an inconsistent number of meta-features across different datasets. To prevent this, the mean and the standard deviation are calculated instead for these meta-features. This restriction does not apply to nr_attr, nr_inst, nr_cor_attr, nr_outliers, nr_missing_values, t2, t3, and t4 because they are calculated for the whole dataset by design.

5.3 Data Generation

This section describes the second step of Figure 5.1, the data generation, in more detail. Additionally, challenge 4.2.2 is addressed. The first decision is to only use synthetic data. There are several different reasons for this decision. There is no limit regarding data availability. Secondly, the generation is much more controllable than when using real-world datasets. To control the generation the characteristics sample size, number of features, and how the data is distributed exist. All this makes it easier to research the effect and associate certain data imperfections with data characteristics.



(a) Gaussian Distributed Data

(b) Moon Shaped Data

Figure 5.2: Different Possible Base Datasets

Another option would be to use benchmark datasets or datasets from collections like the UCI Machine Learning Repository [3] or Kaggle [4]. These datasets are often used in the works presented in Chapter 3. Still, they would introduce the significant drawback that it is usually unknown what data imperfections exist, such that the effectiveness of specific preprocessing methods is more challenging to validate. The second point is that most of these datasets are primarily made for classification, and the use of clustering analysis might not be without its effects [ZZY19].

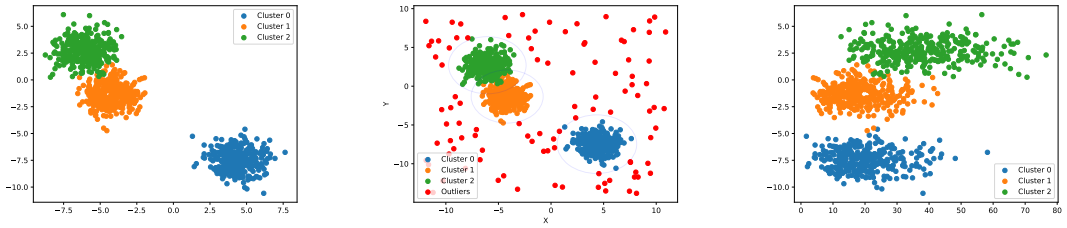
Within the area of synthetic data generation, many different approaches exist to generate differently structured data. Since K-Means uses the Euclidean distance as a disimilarity measure [HTFF09], it may work differently on certain types of these data.

Depicted in Figure 5.2 are two different types of base data, both generated with Scikit-learn [PVG+11] Figure 5.2a shows a more Gaussian-distributed dataset (created with `make_blobs`[5]), while the Figure 5.2b shows moon-shaped clusters (created with `make_moons`[6]). K-Means is expected to perform poorly on the moon-shaped datasets due to the structure of the clusters. To evade this problem, this thesis focuses only on Gaussian-distributed datasets. Within the generation of Gaussian-distributed datasets, there are still many options for creating different datasets. The difference is primarily controlled by the data characteristics sample size, number of features, number of clusters, and the standard deviation of the cluster.

5.3.1 Adding Data Manipulations

The data is manipulated to add data imperfections (cf. Figure 5.1 step 3). This is done to validate the effect of specific preprocessing methods on certain data characteristics. There are two main reasons for that. First, Reed [Ree23] shows that automated preprocessing is not that successful on

[3] https://archive.ics.uci.edu/
[4] https://www.kaggle.com/datasets
[5] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html



(a) Base Data **(b)** Data With Outliers **(c)** Skewed Data

Figure 5.3: Visual Comparison of Base Data (a), Data with Outlier (b), and Skewed Data (c)

not modified synthetic data. This is most likely because the synthetic data generation used (same for Gaussian-distributed data as in this work) is made for clustering and does not include any data characteristics that require preprocessing. Secondly, other related work mainly uses real-world datasets from various benchmark collections with the described drawbacks (cf. Section 5.3). To start with this new approach, three data manipulations are chosen: removing data to create missing values, adding outliers, and skewing parts of the data (cf. Figure 5.3).

Creating Missing Values: Missing values are created by randomly removing a percentage of the data. The only restriction is that no empty instances (rows) are made, which means eliminating all features of a data row is impossible.

Adding Outlier: The outlier generation is oriented on the approaches presented in the survey of Steinbuss and Böhm [SB21]. The generated data of step one (cf. Section 5.3) is used to approximate a hypersphere around the center of every cluster containing all points of the respective cluster. Afterward, the range for every feature is calculated and extended by 20%. Random points are generated within this extended range and the hypersphere boundary. The idea behind this is that outliers appear outside a cluster but not too far away from the real points. Figure 5.3b depicts the result, including the spheres. Additionally, ground truth labels are needed for evaluation but are unavailable for the newly generated outliers. To resolve this, a KNN-Classifier with $k = 5$ is applied on every outlier to obtain the labels based on the closest points around. For this, the K-Neighbor Classifier[7] of Scikit-learn [PVG+11] is used and all parameters besides the n_neighbors= 5 parameter are kept to their default values.

Skewed Data: The data is skewed in three ways, but always for one feature per dataset. It is only done for one feature because the change of this feature is significant, and introducing this change to many features could make the data too obscure. All data points of a feature x can be multiplied with a factor z ($x \times z$), squared x^2, or exponentially transformed e^x. A dataset with a squared feature is depicted in 5.3c.

All manipulation procedures are implemented with different options about the strength of the change in the data. These options are presented in Table 5.2

[7] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Paramater	Description	Possible Values
p_missing_values	The percentage that is removed from the dataset	{0%, 10%, 20%, 30%}
p_outlier	The percentage of outliers relative to the sample size	{0%, 10%, 20%, 30%}
skewness method	The method that is applied to a feature	factor ($z = 10$), squared ($sqaure$), exponential (exp)

Table 5.2: Values of the Data Manipulation Methods

5.4 Selection of Preprocessing Methods

This work's approach to selecting preprocessing methods is to evaluate many different well-known and widely used preprocessing methods. It is step four of the general concept (cf. Figure 5.1). The focus on this kind of method is justified by the described challenges 4.1.1 and 4.1.2, and the reason that current research about automated preprocessing for clustering is limited. Therefore, which methods may be more suitable than others is not yet well known. The process of selecting actual methods can be divided into three steps. First, a classification of the preprocessing method is introduced. The popular techniques are researched and classified, this selection is further reduced to a reasonable amount of methods, and lastly, the parameter ranges for the hyperparameter space are defined.

A classification of preprocessing methods that clearly separates them into distinguishable groups does not exist. Throughout the literature, different names exist for similar preprocessing groups, or some groups of a definitions are contained in the definition of others. An example of this is Han et al. [HPT22] include *normalization* within data transformation where García et al. [GLH15] consider them as separate groups of preprocessing methods. Oriented on García et al. [GLH15], Han et al. [HPT22], the Scikit-learn library [PVG+11], and the data manipulation methods (cf. Section 5.3.1), four groups have been identified that are relevant for this thesis. These are **missing value imputation**, **normalization** (sometimes called **scaling**), **outlier removal**, and **dimensionality reduction**. The relevance of the first three groups is derived from the created data imperfections (cf. Section 5.3.1), while the dimensionality reduction is used due to its general capability of reducing complexity and may be runtime and being able to reduce outliers in some cases [NH19].

The second decision is to focus on widely used preprocessing methods already implemented. This limits the risk of errors or inefficient implementations. Scikit-learn [PVG+11] provides an extensive collection of preprocessing methods for various use cases and is used by many people. One indication is the 58.100 stars on GitHub [8].

The preprocessing methods are then selected based on three criteria:

1. Applicable for unsupervised learning
2. Fit in one of the mentioned groups

[8] https://github.com/scikit-learn/scikit-learn

Group	Method	Parameter Options	Source besides Scikit-learn
Missing Value Imputation	Mean	None	
Missing Value Imputation	Median	None	
Missing Value Imputation	Most Frequent	None	
Missing Value Imputation	KNN-Imputer	{n_neighbors: [1, 16)}	Troyanskaya et al. [TCS+01]
Outlier Removal	LocalOutlierFactor	{p_neighbors*: {0.01, 0.05, 0.1, 0.2, 0.3, 0.5}, algorithm: {auto, ball_tree, kd_tree, brute}, p_leaf_size: {1, 5, 10, 20, 30, 50}, p: {1, 2, 5, 10}, contamination: {auto, 0.05, 0.1, 0.2, 0.3, 0.4}	Breunig et al. [BKNS00]
Outlier Removal	IsolationForest	{n_estimators: {10,50,100,200,500}, max_features: {2, 5, 10, 25, 35}, bootstrap: {True, False}, warm_start: {True, False}	Liu et al. [LTZ12]
Outlier Removal	OneClassSVM	{kernel: {linear, poly, rbf, sigmoid}, gamma: {scale, auto}, nu: {0.3, 0.5, 0.7}, shrinking: {True, False}, max_iter: {100, 500, 1000}	
Outlier Removal	EllipticEnvelope	{assume_centered: {True, False}, contamination: {0.1, 0.2, 0.3, 0.4, 0.5}	Rousseeuw and Driessen [RD99]
Outlier Removal	SGDOneClassSVM	{learning_rate: {constant, optimal, invscaling, adaptive}, warm_start: {True, False}, nu: {0.3, 0.5, 0.7}, max_iter: {100, 500, 1000}, eta0: {0.1, 0.2, 0.3}	
Scaling	PowerTransformer	{standardize: {True, False}	Yeo and Johnson [YJ00]
Scaling	RobustScaler	{with_centering: {True, False}, with_scaling: {True, False}, with_variance: {True, False}	
Scaling	StandardScaler	{with_mean: {True, False}, with_std: {True, False}	
Scaling	MinMaxScaler	{None}	
Scaling	QuantileTransformer	{p_quantiles: {0.3, 0.5, 0.7, 1}, output_distribution: {uniform, normal}	
Scaling	MaxAbsScaler	{None}	
Dimensionality Reduction	PCA	{n_components: special treatment, whiten: {True, False}, svd_solver: {auto, full, arpack, randomized}, power_iteration_normalizer: {auto,QR, LU}, max_iter: {100, 500, 1000}	Minka [Min00], Tipping and Bishop [TB99], and Halko et al. [HMT11]
Dimensionality Reduction	GaussianRandomProjection	{n_components: special treatment, eps: {0.01, 0.1, 0.3}	
Dimensionality Reduction	LocallyLinearEmbedding	{n_components: special treatment, p_neighbors*: {0.01, 0.05, 0.1}, eigensolver: {auto, arpack, dense}, max_iter: {50, 100, 500, 1000}, method: {standard, hessian, modified, ltsa}, neighbors_algorithm: {auto, brute, kd_tree, ball_tree}	Roweis and Saul [RS00], Donoho and Grimes [DG03], Zhang and Wang [ZW06], and Zhang and Zha [ZZ04]

Table 5.3: Preprocessing Methods and their Hyperparameters

3. Are promising for the type of synthetic data

This results in four missing value imputation methods, five outlier removal methods, six scaling methods, and three-dimensionality reduction methods. Most of these methods come with additional hyperparameters. The possible configurations for them are oriented on the default values. Some hyperparameters are not changed to prevent the configuration space from exploding. The unchanged parameters are expected to have a low impact or one where an `auto` value is available, which means a good parameter can be found automatically. The `auto` means an algorithm can adaptively choose a promising value for a given dataset. A detailed list of the selected methods, their parameter options, and the source is provided in Table 5.3. The source is the Scikit-learn documentation and, in some cases, a reference paper mentioned in the documentation.

All parameters marked with an * are changed from absolute to relative values. This means instead of an absolute number, a percentage is defined. This is more adaptive and fits some parameters better. An example is the LocalOutlierFactor with the parameter n_neighbors. In Table 5.3 p_neighbors is given, which would result in $1000 \cdot 0.1 = 100$ n_neighbors for a dataset with n_samples = 1000 and the value 0.1 for p_neighbors.

Secondly, all dimensionality reduction algorithms have a special treatment for the parameter n_components. This parameter is highly sensitive to the data characteristic n_features and can not be higher than it. Otherwise, the algorithms would crash. That is why this parameter is always chosen dependently on the characteristic n_features. Please refer to section 5.5.1 for a detailed definition.

5.5 Optimization of Preprocessing Pipelines

The fifth step of the concept is the optimization (cf. Figure 5.1) and addresses challenges 4.1.3, 4.2.1, and 4.2.3. A genetic optimizer is used to find well-performing pipelines. Some terms have to be mapped from the general terms of genetic programming (cf. Section 2.3) to transfer the concept of evolutionary optimization to the task of preprocessing pipeline optimization. An individual translates to a preprocessing pipeline. Such a pipeline must have at least one preprocessing method. An individual's genes are the methods, parameters, and the order in which they are executed. A population consists of multiple individuals and goes through the typical steps of an evolutionary algorithm. These are initialization, evaluation, crossover, mutation, and selection [SDB+93]. Due to the general tasks of the thesis, all steps except the selection process can not be chosen from out-of-the-box concepts, and they are designed and implemented customarily. This is a standard approach in genetic programming since, at its core, it is a framework or collection of concepts. The selection process is the tournament selection (cf. Section 2.3.1).

A particular consideration must be made concerning missing values since most preprocessing methods can not handle missing values. Therefore, these are optimized in a separate optimization loop. This means if a dataset contains missing values, a smaller optimization process is executed to impute them, and afterward, they are given to the main optimization. Both procedures contain the same steps but differ in mutation and initialization behavior. A pseudocode of the main procedure is provided with Algorithm 5.1. The optimizer itself has parameters. These are the number of generations (n_generations), the population size (population_size), the probability that an individual is selected for a mutation (MUTPB), the probability that an individual is chosen for crossover (CXPB), and the probability that an attribute is mutated within the mutation process (INDPB). Due to the already large search space (cf. Table 5.3), these values are kept unchanged and are selected by orienting on existing work and some smaller tests.

5.5.1 Initialization

The primary approach to initializing pipelines (Line 2) is based on data characteristics. Depending on these, the pipeline is created with random preprocessing methods and parameters from the groups of preprocessing methods that address the data characteristic. This means that besides the fact that a dataset that contains missing values has to be optimized with missing value imputation, a dataset that contains outliers is initialized with an outlier removal method. A dataset that contains outliers and is skewed is initialized with a pipeline of length two, containing an outlier and a normalization method. Which methods out of the groups, which parameters, and the order in which they are executed are generated randomly based on the configuration space $\mathcal{CS}$ (cf. Table 5.3) with some restrictions. Some parameters are dependent on the data characteristics. For example, the parameter

Algorithm 5.1 Optimization Process

```
1:  procedure OPTIMIZE(data, n_generations, population_size, MUTB, INDPB)
2:      population ← INIT_POPULATION(data, population_size)
3:      fitness ← EVALUATE(population)
4:      while g < n_generations and max(fitness) < 1 do
5:          for child_1, child_2 ∈ population, child_1 ≠ child_2 do
6:              if random(0, 1) < CXPB then CROSSOVER(child_1, child_2)
7:              end if
8:          end for
9:          for child ∈ population do
10:             if random(0, 1) < MUTPB then MUTATE(child)
11:             end if
12:         end for
13:         fitness ← UPDATE_FITNESS(population)
14:         population ← SELECT(population, fitness)
15:     end while
16: end procedure
```

max_features of the method IsolationForest can not be larger than the maximal features of the dataset. These parameters are chosen based on data characteristics such as the number of features to prevent such invalid pipelines.

The fitness of a population (Line 3) is evaluated by calculating the score for every pipeline, resulting in a list. As a score, the ARI is used because it is more suited to balanced clusters than other measures like the AMI or Normalized Mutal Information Score (NMI) [RVBV16]. For a detailed description of the ARI, compare section 2.2.1. ARI has a range from -0.5 to one, where one indicates perfect clustering. Thus, the general optimization process aims to maximize the fitness of a population.

5.5.2 Mutation

The mutation differs more between the two cases of missing values or no missing values. Compare the Algorithm 5.2 for the pseudocode of the main mutation.

During the missing value optimization, a pipeline can only consist of one method. Therefore, only the method or parameters can mutate, not the order. Additionally, a method cannot be removed or added to a pipeline. It can only be changed. During the main optimization process, more mutations are possible. Three key mutations are derived from the characteristics of a preprocessing pipeline. These are the mutations of the methods, the parameters, and the execution order. To avoid overcomplicating the process, this work restricts itself to one of these options per mutation step, and the probability for each option is equally distributed. Other options are available, like letting multiple mutations happen per step, but this has the risk of introducing too much change per mutation and, therefore, eliminating promising pipelines.

Algorithm 5.2 Mutation

```
1: procedure MUTATE(individual, hall_of_fame)
2:     mutation_objective ← SELECT_OBJECTIVE({method, parameters, order})
3:                                                  // Can either be, method, parameters, or order
4:     if mutation_objective == method then
5:         method_mutation_objective ← SELECT_OBJECTIVE({add, change, remove})
6:                                                  // Can either be, add, remove, or change
7:         if method_mutation_objective == add then
8:             new_method ← RANDOM_METHOD(M \ current_methods)
9:             parameters ← GET_BEST_PARAMETERS(new_method, hall_of_fame)
10:            if parameters == ∅ then
11:                parameters ← GET_RANDOM_PARAMETERS(new_method)
12:            end if
13:            individual ← new_method, paramaters
14:        end if
15:        if method_mutation_objective == change then
16:            current_method ← RANDOM_METHOD( individual)
17:            new_method ← RANDOM_METHOD( M \ current_methods)
18:            parameters ← GET_BEST_PARAMETERS(new_method, hall_of_fame)
19:            if parameters == ∅ then
20:                parameters ← GET_RANDOM_PARAMETERS(new_method)
21:            end if
22:            individual ← EXCHANGE_METHOD( new_method, curren_method)
23:        end if
24:        if method_mutation_objective == remove then
25:            if pipeline_length > 1 then
26:                current_method ← RANDOM_METHOD( individual)
27:                individual ← REMOVE_METHOD( individual, current_method)
28:            end if
29:        end if
30:    end if
31:    if mutation_objective == paramater then
32:        for paramater ∈ individual.paramaters do
33:            if random(0, 1) < INDPB then
34:                parameter ← MUTATE_PARAMETER(HP)
35:            end if
36:        end for
37:    end if
38:    if mutation_objective == order then
39:        if pipeline_length > 1 then
40:            method_1 ← RANDOM_METHOD( individual)
41:            method_2 ← RANDOM_METHOD( individual \ method_1)
42:            individual ← SWAP_METHODS( method_1, method_2)
43:        end if
44:    end if
45: end procedure
```

The method mutation (Lines 4-31) is again split into different mutation options: adding a method, removing a method, or replacing a method with another. The only constraint when adding a method is that the method is not currently present in the pipeline (Line 8). The parameters of the newly added method are taken from the top 50 pipelines (Hall of Fame, Line 18). If this method does not occur within the Hall of Fame, the parameters are initialized randomly from the hyperparameter space (Line 20). Exchanging methods (Lines 15-23) follow the same constraints and logic as adding a method. The last part is removing a method (Lines 24-28). Here is the only constraint a pipeline can mutate to an empty pipeline.

The parameter mutation iterates over every parameter of the pipelines and replaces with the probability INDPB (Lines 32-37). The replacement is randomly drawn from the corresponding part of hyperparameter (cf. Table 5.3).

Algorithm 5.3 Crossover

```
1:  procedure CROSSOVER(individual_1, individual_2)
2:      methods_in_both ← methods ∈ individual_1 ∩ individual_2
3:      for method ∈ methods_in_both do
4:          for parameter ∈ method do
5:              if random(0, 1) < CXPB then
6:                  individual_1, individual_2 ← SWAP_PARAMETERS(individual_1, individual_2, parameter)
7:              end if
8:          end for
9:      end for
10: end procedure
```

The last option is swapping two methods (Lines 38-44) of the existing pipeline to mutate the order. Again, this is not possible if the pipeline has the length 1.

5.5.3 Crossover

Due to the more complex mutation step and the many different options that can happen during the mutation, the crossover step is kept relatively simple (cf. Algorithm 5.3). The main restriction is that only parameters can be crossed, not whole methods. To prevent invalid pipelines, only parameters of the same methods are swapped. To do this, a list of methods that occur in both pipelines is first created (Line 2), and then each parameter is exchanged with the probability CXPB (Lines 3-9).

5.6 Summary

This section serves as a brief summary of the general concept and decisions.

Synthetic data is generated to evaluate the effect of preprocessing methods on clustering analysis. This synthetic data is further modified to introduce imperfections like outliers or missing values. This datasets are then preprocessed. Since it is tough to find good working pipelines, a genetic optimization approach is used to refine existing pipelines. During this process, the pipelines are evaluated by clustering with K-Means and calculating the ARI. The changes arising from introducing data imperfections and the preprocessing optimization are captured by calculating a set of meta-features. All this information, like the data characteristics, the meta-features, the scores, and the best-performing pipelines, is stored in a knowledge base to provide the possibility of further evaluation.

6 Evaluation

This chapter evaluates the presented concept of chapter 5. The goals of this evaluation are to find out what effects the preprocessing optimization has concerning the data manipulation, how the optimizer finds promising preprocessing pipelines, and if the data manipulations can be reflected with meta-features.

The following chapter is structured as follows: Section 6.1 provides an overview about the implementation and the setup of the experiments. Section 6.2 covers the effects of the data manipulation and primarily how the clustering results are changed after the application of the best-performing pipeline of the optimization. In Section 6.3, the optimization process is further analyzed in terms of pipeline evolution. Section 6.4 compares the achieved results of Section 6.2 against two baselines. In Section 6.5, it is investigated if the order of preprocessors is relevant in terms of the achieved score. Section 6.6 analyzes if the data manipulation is reflected in the meta-features and what meta-features are necessary for this type of data. After that, the runtime of the different datasets is evaluated in Section 6.7, and Section 6.8 concludes with a summary.

Throughout all sections, the preprocessors are evaluated with K-Means clustering and the ARI as an accuracy measure. Since optimizing a Clustering Algorithm is beyond the scope of this work, the true number of clusters is provided to K-Means, and all other parameters are kept to their default values (cf. Scikit-learn [PVG+11][1]). The datasets containing missing values need to be handled specially. The missing values have to be imputed before other preprocessing methods, or clustering can be applied. Therefore, the missing value imputation is dealt with in a separate small optimization called the missing value imputation optimization before the other optimization process (called the main optimization) starts. For a more detailed explanation, refer to Section 5.5.

6.1 Implementation and Setup

This section covers the used software (cf. 6.1.1), the characteristics of the hardware on which the experiments were performed (cf. 6.1.2), the data characteristics of the evaluated datasets (cf. 6.1.3), and the parameters of the optimization (cf. 6.1.4).

6.1.1 Software

The entire concept is implemented with Python [VD09][2] in version 3.11.8. Additionally, some packages are used to facilitate the implementation. The packages, the version of the package, and why they are used are listed in Table 6.1

[1] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
[2] https://www.python.org/

Library	Version	Usage
Scikit-learn [PVG+11][3]	1.2.2	Used for the data generation, preprocessing, clustering, and score calculation
SciPy [VGO+20][4]	1.11.4	Used for outlier generation
DEAP [FDG+12][5]	1.4.1	Evolutionary computation framework used for the optimization
NumPy [HMW+20][6]	1.26.1	Used for basic calculations (like the mean) and for generating random numbers
Jupyter Notebook [KRP+16] [7]	7.0.8	Used for quick testing and the evaluation of the experiments
Matplotlib [Hun07] [8]	3.8.0	Used for generating graphics
Pandas [McK10] [9]	2.1.4	Used for general data preprocessing and evaluation

Table 6.1: Libraries Used in the Implementation

6.1.2 Hardware

The evaluations are performed on different machines with the following hardware configurations and operating systems:

1. **Server 1:**
   - Processor: AMD EPYC 7763
   - Threads: 75 (more available but not used)
   - RAM: 1 Terabyte
   - Operating System: Ubuntu 20.04.6 LTS
2. **Server 2:**
   - Processor: AMD Ryzen 5 3600
   - Threads: 16 Threads
   - RAM: 64 Gigabyte
   - Operating System: Ubuntu 22.04.4 LTS
3. **Virtual Machine**

[3] https://scikit-learn.org/stable/index.html
[4] https://scipy.org/
[5] https://deap.readthedocs.io/en/master/
[6] https://numpy.org/
[7] https://jupyter.org/
[8] https://matplotlib.org/stable/
[9] https://pandas.pydata.org/

- Processor: AMD EPIC Milan
- Threads: 8 Threads
- RAM: 32 Gigabyte
- Operating System: 22.04.1 LTS.

6.1.3 Datasets

The Scitkit-learn method `make_blobs`[10] is used to generate the basis of the synthetic data. This method generates Gaussian distributed data that can be varied with parameters. These parameters are the sample size (`n_samples`), the number of features (`n_features`), the number of clusters (`centers`), and the standard deviation of the cluster (`cluster_std`). The parameter ranges for the data generation of this work are:

- **Number of Samples:** $\{500, 1000, 5000\}$
- **Number of Features:** $\{10, 30, 50\}$
- **Number of Clusters:** $\{10, 30, 50\}$
- **Cluster Std:** $\{3, 5, 10\}$

These generated datasets are further manipulated to add data imperfections (cf. Section 5.3.1). The values of these manipulations are:

- **Missing Values:** $\{0\%, 10\%, 20\%, 30\%\}$
- **Outlier:** $\{0\%, 10\%, 20\%, 30\%\}$
- **Scaling:** $\{\text{none}, \text{factor } (= 10), \text{square}, \exp\}$

With these parameter ranges defined, the datasets are created such that every combination of all parameters exists. The names for these datasets, further used in this evaluation, are derived from the data manipulations they contain. Overall, this results in the data manipulation groups described in Table 6.2.

[10] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

Name	Description	Number of Datasets
N	Contains no data imperfections	81
M	Has missing values	243
O	Contains outlier	243
S	Contains skewed data	243
S+O	Contains skewed data and outlier	729
M+O	Has missing values and contains outlier	729
M+S	Has missing values and contains skewed data	729
M+S+O	Has missing values, contains skewed data and outlier	2187
Total		**5184**

Table 6.2: Datasets of the Evaluation

6.1.4 Paramater of the Optimization

DEAP [FDG+12] is used for the implementation of the evolutionary optimization. It has parameters that can influence the general behavior (cf. 5.5). This evaluation keeps these parameters constant to not further increase the search space. Two differences must be made between the missing value and the main optimization. Fewer generations are executed since the missing value optimization has a smaller configuration space. The parameter values are as follows:

Paramater	Description	Value
Generations Missing Value Optimization	The number of generations executed per dataset	10
Generations Main Optimization	The number of generations executed per dataset	50
Population Size	The size of the population (number of initial pipelines)	20
MUTBP	The probability that an individual (a pipeline) is selected for a mutation	20%
CXPB	The probability that an individual (a pipeline) is chosen for crossover	50%
INDPB	The probability that an attribute (hyperparameter of a preprocessor method) is mutated within the mutation process	70%
Tournsize	The size of the tournament of the tournament selection process	5

Table 6.3: Parameters of the Optimization-Implementation

Additionally, a time budget of ten minutes per dataset is defined to prevent pipelines from running too long. If this time limit is reached, the current generation can finish its optimization, and afterward, the optimization is stopped.

There is no general rule of thumb or standard values for the parameters mentioned, as they depend heavily on how the mutation and crossing are implemented. However, a broad evaluation of these parameters is beyond the scope of this thesis. Therefore, different values were tested with a small part of the datasets, and the above-mentioned parameters turned out to be suitable for this approach.

6.2 Evaluation of Preprocessing Effects

This section covers the overall effect of the preprocessing pipeline optimization, including measuring if the processing improves the clustering score, if there are any differences between different data manipulations in terms of the achieved score, and if the scores are consistent. To accurately measure the impact of preprocessing, K-Means clustering is applied, and the quality of the result is measured with the ARI. This is done for every dataset once before any preprocessing is applied to achieve a result without preprocessing and during optimization to assess the fitness of the preprocessing pipelines. Afterward, the best-performing pipeline per dataset is considered to measure the effect of the preprocessing optimization. It is impossible to cluster datasets containing missing values because they can not be executed with K-Means. To still get a score, all missing values are replaced with 0.

6.2.1 Evaluation of Optimization Results

This part describes the general effect of the preprocessing optimization.

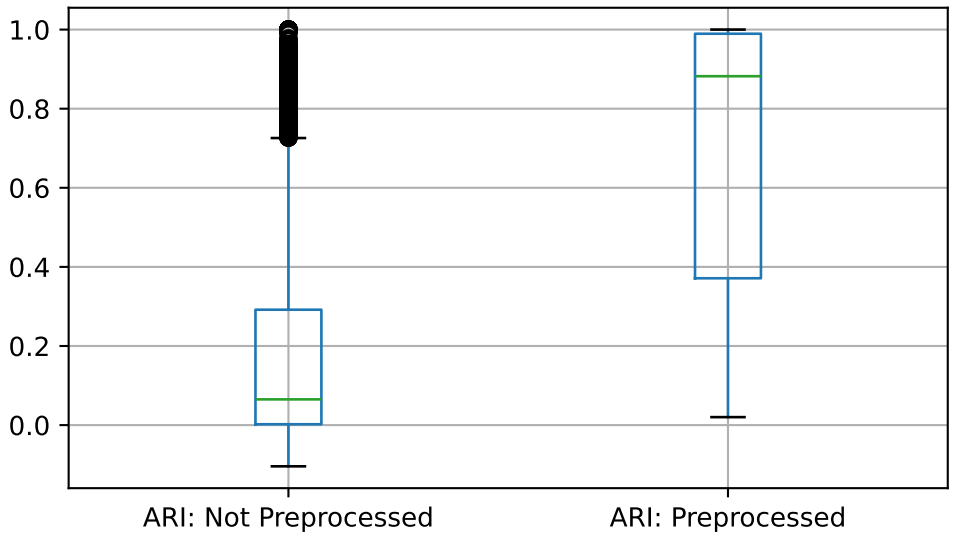


Figure 6.1: ARI Not Preprocessed (Left), and Preprocessed (right)

Figure 6.1 shows the ARI before the optimization is started (left boxplot) in comparison to the data where preprocessing pipelines are applied (right).

Key Finding: Preprocessing leads to a significant ARI increase.

Before optimization, a median ARI of 0.08 is achieved (cf. Figure 6.1). After the optimization, this improves significantly to a median of 0.89. The high increase indicates that the optimization is able to find well-performing pipelines and is able to significantly reduce the effect of the applied data manipulations.

Table 6.4 shows the median ARI per data manipulation group, where column two shows the median ARI before optimization, columns three and four the ARI after the missing value imputation optimization (MVI) and the difference to the median ARI before preprocessing, column five, six and seven the median ARI after the main optimization, the difference to the ARI after the missing value imputation optimization and the difference to median ARI before preprocessing. Only the data manipulations groups M, M+S, M+O, and M+S+O contain missing values and run through both optimizations, one for the missing value imputation and one for the rest (cf. Section 5.5). The rest is only preprocessed with the main optimization. Therefore, there are no median ARI and difference after the missing value imputation (columns three and four) and no difference compared to the missing value imputation (column 6). This is indicated by a ïn the respective cells.

Data Manipulations	**Before Preprocessing**	**After Missing Value Imputation (MVI)**		**After Preprocessing**		
Name	Median ARI	Median ARI	Difference (vs. Before Preprocessing)	Median ARI	Difference (vs. MVI)	Difference (vs. Before Preprocessing)
N	0.87	-	-	0.97	-	0.1
M	0.75	0.87	0.12	0.89	0.02	0.14
S	0.05	-	-	0.96	-	0.91
O	0.68	-	-	0.96	-	0.28
M+S	0.04	0.05	0.01	0.84	0.79	0.8
M+O	0.52	0.64	0.12	0.88	0.24	0.36
S+O	0.04	-	-	0.92	-	0.88
M+S+O	0.04	0.07	0.03	0.83	0.76	0.79

Table 6.4: ARI Change During Preprocessing per Data Manipulation Group

Key Finding: Skewed datasets achieve the lowest score without preprocessing optimization, and datasets without manipulation achieve the best.

The data manipulation group without imperfections (N) has overall the best results without preprocessing with a median ARI of 0.87, while the preprocessing for the M, O, and M+O datasets performs worse with a median ARI between 0.52 and 0.75. All data manipulation groups containing skewed data (S, M+S, S+O, M+S+O) perform significantly worse with a median ARI between 0.04 and 0.05. Within the data manipulation groups, the percentage of outliers, the percentage of missing values, and the type of skewed data influence the ARI as expected. For the missing values and outliers, the higher the value is, the worse the clustering result. For example, preprocessing for datasets with 30% outliers performs worse than for those with 10%. Additionally, preprocessing for datasets that are skewed with the parameter `exp` performs significantly worse than datasets skewed with the other methods.

Key Finding: Skewed datasets benefit the most from preprocessing, and datasets without imperfection the least.

The effects are inverse to the effects described without optimization. The group without data manipulations (N) improves only by 0.1 to a median ARI of 0.97, and the group only containing missing values (M) increases by 0.14. This is most likely because the ARI is already high before preprocessing, a constant replacement of missing values is good enough (used to calculate the ARI of not preprocessed data with missing values), or the datasets without imperfections do not benefit from the preprocessing methods. The most significant increase of the ARI can be observed by the scaling datasets (S, M+S, S+O, M+S+O) and ranges between 0.79 and 0.91. The datasets containing outliers but are not skewed (O, M+O) have a moderate increase between 0.28 and 0.36.

Key Finding: Datasets containing missing values benefit more from the main optimization compared to the missing value imputation optimization. One Exception is the M data group.

All data manipulation groups that contain missing values (M, M+S, M+O, M+S+O) run through two optimizations, one for the missing value imputation and one for the rest (cf. Section 5.5). Within these two optimizations, the missing value imputation contributes the smaller portion with an improvement between 0.01 and 0.12, while the main optimization provides the more significant part for the two groups M+O and M+S+O with an improvement between 0.24 (M+O) and 0.76 (M+S+O). The exception is dataset M, where the main optimization only contributes a small part of 0.02. The reason for the minor improvement by the missing value imputation part is that the comparison is made against a constant replacement with zero. Therefore, an imputation has already happened, even if it might not be a good one. The difficulties for the clustering algorithm arise more through skewed data and outliers than missing values (replaced by a constant). An influence for the poorer scores of O and M+O compared to S and M+S might be that the outliers are given class labels of their nearest neighbors (cf. Section 5.3.1). This might favor the clustering algorithm in such a way that they are added to the closest cluster and, if no close clusters are nearby, are clustered correctly.

Key Finding: Datasets that have high clustering results without preprocessing are from the N and M data manipulation group and have a smaller number of clusters and a smaller cluster standard deviation compared to others.

There are still datasets that can be clustered quite well with an ARI > 0.8 (cf. Figure 6.1 left). These are mainly datasets with a low number of clusters (10), a low cluster standard deviation (3-5), and either no data imperfections (N) or only missing values (M). The reasons for the good results without preprocessing are most likely that the basic data generation is designed for clustering and K-means works well as expected with these datasets, that constant replacement by zero works well enough for some missing values datasets, and that the small number of clusters and the low standard deviation compensate for the introduced data manipulations.

6.2.2 Best and Worst Scores

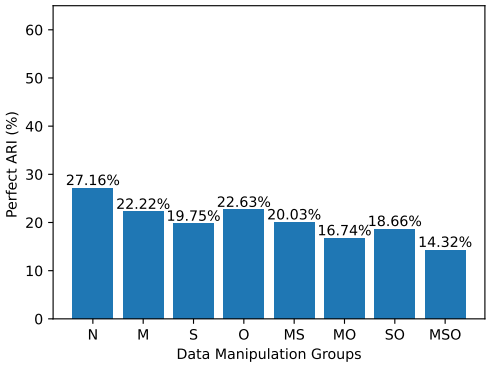
This section covers those datasets that achieve a particular high or low ARI to investigate whether there are manipulation groups that benefit particularly much or particularly little from preprocessing. To make the numbers more comparable, they are provided in percent, where 100% is the number

of all datasets of the respecting manipulation (cf. Section 6.1.3). For example, 100% of the M manipulation group are 243 datasets. If an overall comparison is made, the 100% is the sum of all manipulation groups, which is 5184.

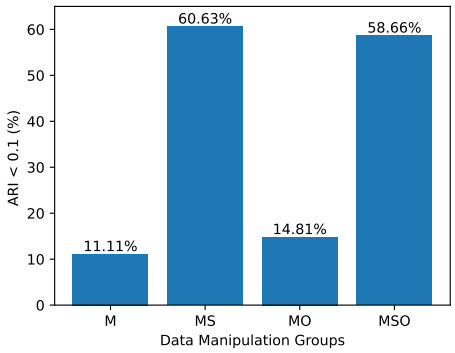
Key Finding: 17.28% of the preprocessing pipelines achieve an perfect ARI score of 1 and 7.21% a ARI if <= 0.1

Overall, a perfect score of 1.0 on the ARI is achieved 896 times, which constitutes 17.28% of all datasets (5184), and a ARI of <= 0.1 is achieved 374 times (7.21%). This shows that, in general, more datasets reach a perfect score than there are ones that achieve a very bad one. The datasets have similar characteristics to the ones that achieve good or bad clustering results without preprocessing (cf. Section 6.2.1). Therefore, the reason lies most likely within the data characteristics.

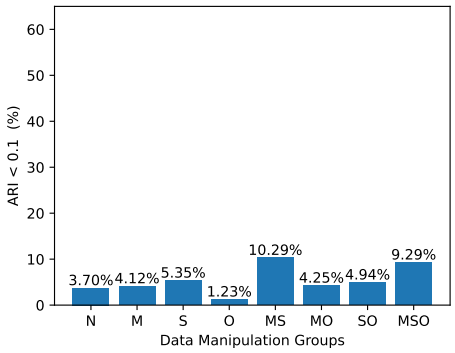
Figure 6.2 shows the percentage per group for achieving a perfect score (6.2a), achieving a score below or equal to 0.1 (6.2b) after the missing value imputation, and after the main optimization (6.2c).



(a) Percentage of Datasets that Achieve a Perfect ARI



(b) Percentage of Datasets that Have an ARI <= 0.1 After Missing Value Imputation



(c) Percentage of Datasets that Have an ARI <= 0.1 After Main Optimization

Figure 6.2: Percentage of datasets that achieve a perfect score (a), a score below 0.1 after the missing value imputation optimization (b), and after the main optimization (c)

Key Finding: Datasets with perfect results after preprocessing are mostly datasets from the data manipulation groups N and M

The datasets without data manipulations (N) achieve the highest rate of perfect results with 27.16%, followed by the outlier manipulation group (O) with 22.63%. Data manipulation groups S, M+S, M+O, S+O M+S+O achieve between 19.75% and 14.31%, where M+S+O has the smallest percentage of perfect ARI scores (cf. Figure 6.2a). The missing value imputation optimization of the groups M+S, M+O, and M+S+O do not produce any ARI of 1.0. The good scores in N are probably because these datasets do not contain any imperfections. The reasons for that are most likely similar to those mentioned before. The preprocessing for the N dataset performs well because it has no data imperfections and is easier to cluster. The M dataset is firstly well immutable and secondly compared against a constant replacement. The fact that the missing value imputation is not able to produce a perfect score if other data imperfections exist (M+S, M+O, M+S+O) indicates that every data flaw contributes to the clustering result.

There is also a difference in data characteristics: datasets with fewer clusters, a high number of features, and a small standard deviation per cluster tend to perform better than others. This is most likely because a smaller standard deviation leads to fewer overlapping clusters, and fewer clusters, in general, leave more points per cluster and make it, therefore, easier to find by K-Means.

Key Finding: Datasets with an ARI $<= 0.1$ **after preprocessing are mostly datasets that are skewed.**

The data manipulation group with the largest number of worse-performing datasets are the M+S (10.29%) and M+S+O (9.82%) groups (cf. Figure 6.2c. For the M+S dataset, the cause may be that some skewed data is not well imputed, and then the scaling afterward can further improve it. That some data from the M+S+O manipulation group do not perform well is not surprising since all data manipulations combined may produce data where it is harder to find a good preprocessing pipeline. A second reason might be that these datasets have a very high percentage of bad results after the missing value imputation (60.63% and 58.86%, cf. Figure 6.2b). That shows that the missing value imputation barely improves the clustering results, and an imperfect imputed dataset might be challenging to preprocess correctly further. The other data manipulation groups have a percentage between 4.94% and 1.23%. Again, these percentages are most likely due to unlucky initialization or data that is generally too messed up.

6.2.3 Consistency of Scores

To calculate the consistency and reproducibility of the scores, the M, O, and S data manipulation groups are run five times, and the best scores per dataset are considered. The results are presented in Table 6.5. Column one shows the name of the data group, column two the mean ARI of each run, and columns three and four the overall mean and standard deviation.

Key Finding: Overall, the results are highly similar, and the overall standard deviation is zero. This indicates a strong consistency in terms of the ARI scores achieved.

It should be noted that there are differences that vanish through the rounding, but these are insignificant.

Data Manipulations	Mean ARI of Each Run	Overall Mean ARI	Standard Deviation
M	{0.68, 0.68, 0.67, 0.68, 0.68}	0.68	0
S	{0.75, 0.75, 0.75, 0.75, 0.75}	0.75	0
O	{0.76, 0.77, 0.77, 0.77, 0.76}	0.77	0

Table 6.5: Consistency of Multiple Runs

6.3 Evaluation of the Optimization Process

This section covers the evaluation of the optimization process, including the missing value imputation optimization and main optimization. First, the performance of the generations of both optimizers is compared, and the effects of the time limit are investigated (cf. 6.3.1). Section 6.3.2 researches how the pipeline length changes during the main optimization, and Section 6.3.3 evaluates if there are any prominent preprocessing pipeline patterns that occur more often than others within the best-performing pipelines.

6.3.1 Evaluation of Generations

The goal of this part is to investigate how the mean fitness (the ARI) within the generations of both optimizers evolves. This is done to further check if the length of the optimization process is reasonable or if there is more potential with a different configuration of the optimizers.

Figure 6.3 shows the mean of the best fitness (ARI) of all datasets per generation, combining the missing value optimization and the main optimization. Since some datasets have missing values (M, M+S, M+O, and M+S+O), the missing value imputation needs to be executed before the main optimization. This takes place in the first ten generations. Afterward, the main optimization with 50 generations starts, indicated by a grey dashed line (cf. Figure 6.3). If a pipeline is stopped prematurely due to the time limit being reached, the last available score is transferred to all further generations.

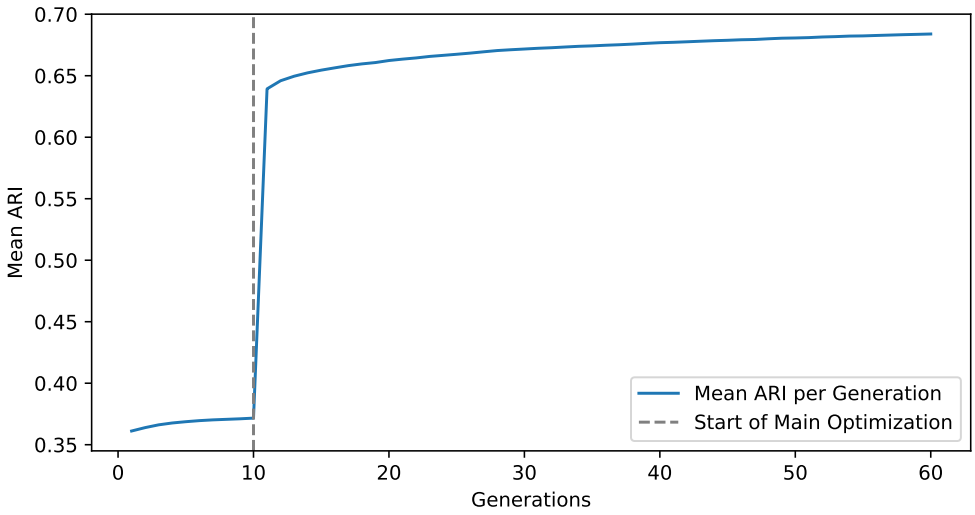


Figure 6.3: Mean ARI per Generation

Key Finding: A well-performing pipeline is found relatively quickly but improves further.

As depicted in Figure 6.3, a good-performing pipeline for the respective tasks is found within the first generations of the two optimizers. This means that the missing value imputation optimization quickly finds a good imputation, and the main optimizer finds a good pipeline in relation to the other preprocessing methods.

After that, there is still a measurable increase, but it is significantly smaller than the first generations. The reason for that is most likely the initialization phase (cf. Section 5.5.1). The pipelines are initialized depending on the data manipulations (cf. Section 5.5.1). Therefore, a good-performing pipeline is found relatively fast. Additionally, it should be noted that there are cases where the curve has a slower slope at the end but does not reach an apparent plateau. This indicates that there might still be a slight improvement if the optimizers are run for more generations, but this automatically includes longer runtime.

Effect of Time Limit

Key Finding: The time limit is reached with data records that have 5000 samples. On average, more than half of the generations are executed.

A time limit of ten minutes per dataset is applied (cf. Section 6.1.4). Combining the missing value imputation and main optimization, 9072 optimizer runs are executed. Out of these, 3888 are missing value imputation optimization runs for the datasets M, M+S, M+O, and M+S+O, and 5184 runs are the main optimization that is applied to all datasets. Out of these 9071 runs, 1085 (11.96%) have been stopped by the time limit. During the missing value, imputation 189 (4.86%) are stopped, and 896 (17.28%) are stopped during the main optimization during the main optimization. Datasets that are stopped during the missing value imputation optimization have a sample size of 5000 and have a high missing value percentage (30%). The mean generation length of all pipelines that

are stopped during the missing value imputation is seven (ten are maximal possible). This means most datasets have run over half of the optimization, and the overall impact should not be that drastic. The main optimization observes a similar trend. Most datasets have 5000 samples and are from the M+S+O data manipulation group, which contains the most data imperfections. The mean generation length of all stopped pipelines is 38.76 (50 are maximal possible). This indicates that most datasets achieved several optimization cycles as well.

This time limit has been applied to reduce the experiments' runtime and stop generations stuck with long-lasting pipelines. Considering the presented numbers and the general trend depicted in Figure 6.3, it can be assumed that an optimization without a time limit would lead to better scores in some cases. However, the effect might be small.

6.3.2 Pipeline Evolution

This part covers whether there are changes in the pipeline length and the type of method during the optimization process to investigate whether the pipelines tend to shrink or grow and, if they grow, whether there are any patterns in relation to the method groups (see section 5.4).

Pipelines are initialized depending on the data manipulation groups (cf. Section 5.5.1). This means a dataset of the O manipulation group contains only outliers and, therefore, every pipeline of the population is only initialized with a single randomly chosen outlier method.

Data Manipulations	Initial Pipeline Length	Mean Pipeline Length	Minimal Pipeline Length	Maximal Pipeline Length
N	1	1.36	1	3
M	1	1.37	1	4
O	1	1.46	1	4
S	1	1.51	1	4
M+S	2	2.55	2	4
M+O	2	2.45	2	4
S+O	2	2.27	2	5
M+S+O	3	3.39	3	6

Table 6.6: Evaluation of Pipeline Length

Key Finding: Pipelines tend to grow during the main optimization but not by much.

Table 6.6 shows the minimal, mean, and maximal pipeline length of the best performing per data manipulation group. Generally, there is always a growth of average pipeline length compared to the initialized length, but the growth is relatively small. While there exist some outliers, like the pipeline of length 6 for the M+S+O manipulation group, most of the datasets stick to their initialized length or add one method. Regarding data characteristics, all datasets with a large cluster standard deviation (10) or smaller number of features (10-30) have longer pipelines than others. Since only the best-performing pipelines are considered, there is an indication that, generally, harder-to-cluster datasets benefit from additional preprocessing methods.

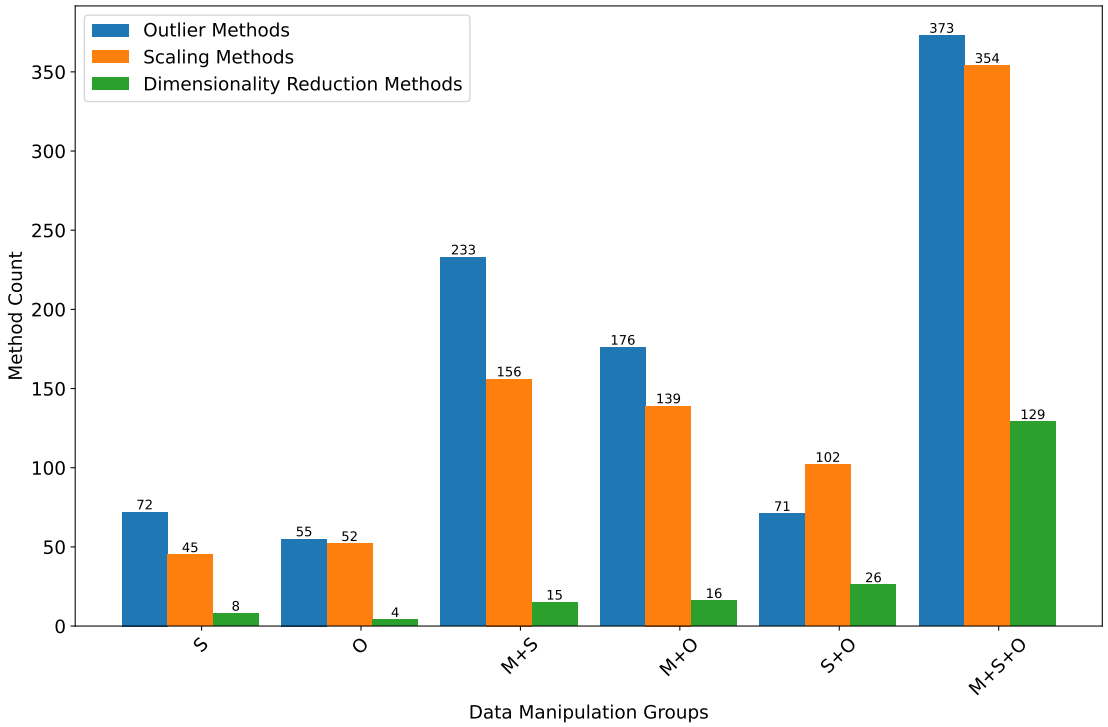


Figure 6.4: Additions of Method Groups per Data Manipulation Group

Figure 6.4 shows the overall number of methods that were added during the main optimization per method group and data manipulation group. The term "added" in this context refers to the fact that these methods were added to the pipeline after initialization during the optimization process. The figure shows the main addition is often a scaling method and, slightly less often, an outlier method. In contrast, the addition of a dimensionality reduction is extremely rare. This indicates that scaling generally can support clustering, even if the data is not additionally skewed. Adding outlier methods to the pipeline for datasets where no outliers have been added can be useful from the point of view of the optimization pipeline, especially in cases where there is a high cluster standard deviation and the points are widely scattered because these points can be considered as outliers and removal of them can improve the clustering result.

6.3.3 Patterns of Methods

This part examines whether there are any patterns of preprocessing methods that are selected particularly frequently. Its purpose is to explore whether it is possible to reduce the search space of preprocessing methods for the data used in the work.

Again, the missing value imputation optimization is differentiated from the main optimization because of the different structure. The missing value imputation part only consists of a single method per pipeline (cf. Section 5.5). Additionally, the missing value imputation is only the datasets that contain missing values (M, M+S, M+O, and M+S+O). This is a total of 3888 datasets (cf. Section 6.1.3).

Data Manipulations	KNN-Imputer	Mean-Imputer	Median-Imputer	Most-Frequent-Imputer
M	220	16	7	0
M+S	556	77	77	19
M+O	667	37	25	0
M+S+O	1628	360	80	119
Total	**3071**	**490**	**198**	**138**

Table 6.7: Occurrences of Imputation Methods per Data Manipulation Group

Key Finding: The KNN-Imputer appears significantly more often than other imputers.

Table 6.7 shows the distribution of imputation methods per data group. Out of these 3888 datasets, 3071 (78.99%) have the KNN-Imputer as the best method, 490 (12.6%) the Mean-Imputer, 189 (5.1%) the Median-Imputer, and 138 (3.55%) the Most-Frequent Imputer. The preference of the KNN-Imputer over the others is most likely caused by the fact that all other imputation techniques consider all features individually, while the KNN-Imputer considers all features of the nearby points. This results in a better approximation. The imbalance between the Mean-Imputer compared to the Median-Imputer and Most-Frequent imputer shows that the mean does approximate the missing value better than the others for this type of data. Concerning method parameters, there is no clear indication of when to use which parameter besides the fact that the KNN-Imputer tends to be selected more often with a higher number of k compared to a lower number. In addition, there is no indication concerning data characteristics as to when which imputer is selected.

The main optimization shows a more diverse picture since many more methods are available, a pipeline can contain more than one method, and the optimization process is more complex (cf. Section 5.5).

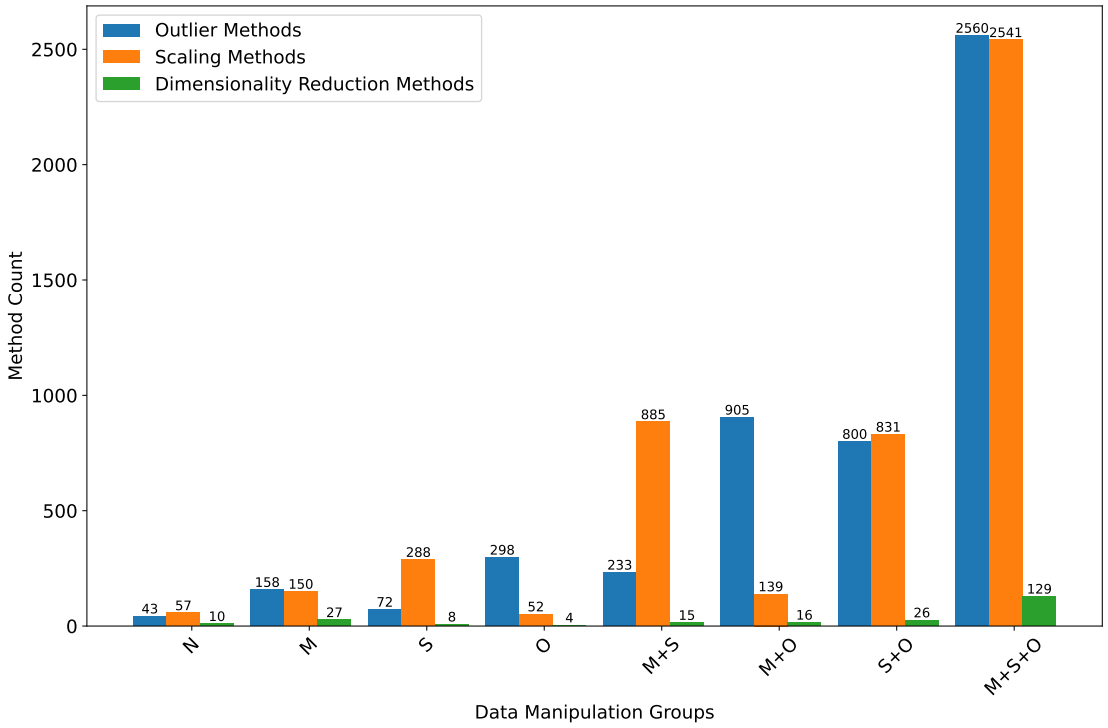


Figure 6.5: Method Groups per Data Manipulation Group

Key Finding: Dimensionality reduction methods do not occur often in pipelines.

Figure 6.5 depicts the number of methods per method and data manipulation group. One prominent pattern that can be observed is that the dimensionality reduction methods are heavily underrepresented compared to the scaling and outlier methods. The reason for that is probably the initialization phase. Since pipelines are initialized based on the dataset's data imperfections, and the dimensionality reduction methods are added as a supporting group and do not have explicit data imperfections where they are initialized, the optimizer only adds them through mutation (cf. Section 5.5.2.

Additionally, Figure 6.5 shows that there is a clear connection between the data manipulation group and the preprocessing method groups, e.g., datasets that contain outlier have mostly outlier methods in their corresponding best-performing pipeline. The reason why the number of methods is greater than the number of data records from the corresponding group is that many pipelines contain more than one method.

Method Group	Method	Number of Occurences
Outlier	Isolation Forest	978
Outlier	Local Outlier Factor	713
Outlier	One Class SVM	407
Outlier	SDG One Class SVM	88
Normalization	Power Transformer	572
Normalization	Robust Scaler	279
Normalization	Standard Scaler	382
Normalization	Min Max Scaler	475
Normalization	Quantile Transformer	624
Normalization	Max Abs Scaler	510
Dimensionality Reduction	PCA	45
Dimensionality Reduction	Gaussian Random Projection	31
Dimensionality Reduction	Locallaly Linear Embedding	30

Table 6.8: Overall Occurrences of Methods of the Main Optimization

Key Finding: Isolation Forest and Local Outlier Factor are the preferred outlier methods, and the Robust Scaler is less often selected as a normalization method than others.

Within the method groups, the selection differs on some occasions, as listed in Table 6.8. For the outlier removal, the Isolation Forest and Local Outlier Factor methods occur significantly more often than the other methods. The causes for that could be that due to how the methods work, Isolation Forest and Local Outlier Factor are better in outlier detection for these kinds of data and randomly added outliers or that the other methods contain hyperparameters that are harder to fine-tune since they drastically change the behavior and a mutation can more easily ruin a particular configuration. An example of that is the OneClassSVM method [11] that is sensitive to the selected kernel and ν parameter.

Only the robust scaler is less often chosen among the scaling methods. This is surprising since it should handle scaling with outliers better than the others and, therefore, is expected to occur more often, especially if a dataset is skewed and has outliers.

The dimensionality reduction methods are selected almost equally often.

6.4 Baselines

This part compares the presented concept against two baselines. Since there is no comparable approach or library that makes a direct comparison possible (cf. Chapter 3), two basic and relatively weak baselines are chosen. The first generates a random pipeline per dataset. In contrast, the second creates a random baseline based on the data imperfections, e.g., a dataset with a skewed dataset gets initialized with a random normalization method.

[11] https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

Key Finding: The presented concept with its optimizers outperforms the baselines significantly.

The results are depicted in Figure 6.6. For a better comparison, the not preprocessed mean ARI (left) and the mean results after the optimization (right) are provided as well. Compared to the not preprocessed mean ARI of 0.21, there is a slight increase to 0.27 for the baseline with random preprocessing and 0.37 for the baseline where pipelines are generated based on the data imperfections of the dataset. However, the optimizer outperforms both baselines significantly. Regardless of the outperformance, some datasets still achieve a high ARI with both baseline approaches. These datasets already achieved good results without preprocessing and have a small cluster standard deviation and fewer clusters (cf. Section 6.2).

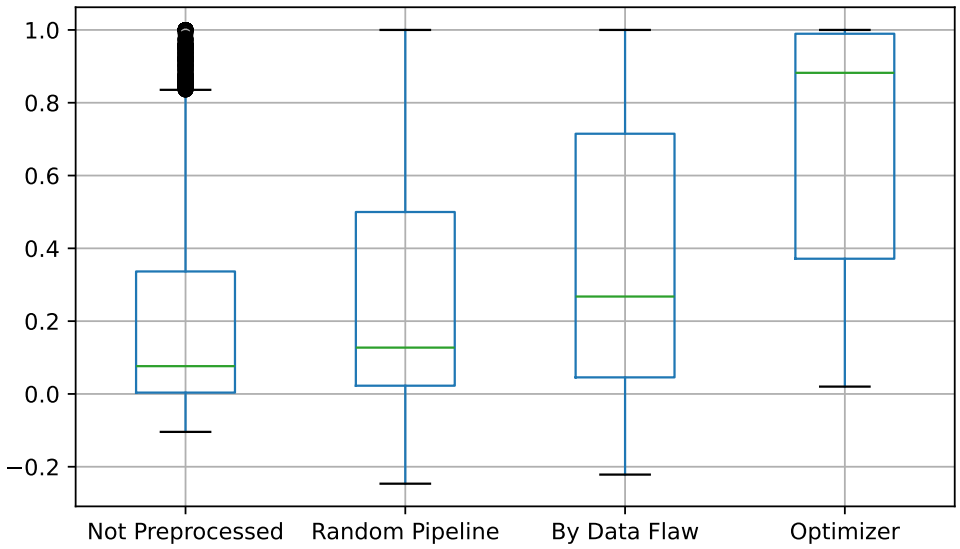


Figure 6.6: Comparison of mean ARI Between Not Preprocessed (Left), Baselines (Middle) and Optimizer Results (Right)

6.5 Order of Preprocessing Methods

The sequence of preprocessing methods is an aspect that has been particularly little studied. Therefore, this section evaluates whether there are patterns in the order preprocessing method groups (cf. Section 5.4) and secondly checks if this order is significant regarding the ARI score.

To analyze the order, all pipelines longer than two are evaluated. The missing value imputation part is also excluded because it always has to happen first. Therefore, a pipeline containing a missing value imputation, a normalization method, and an outlier removal method is considered a pipeline of length two.

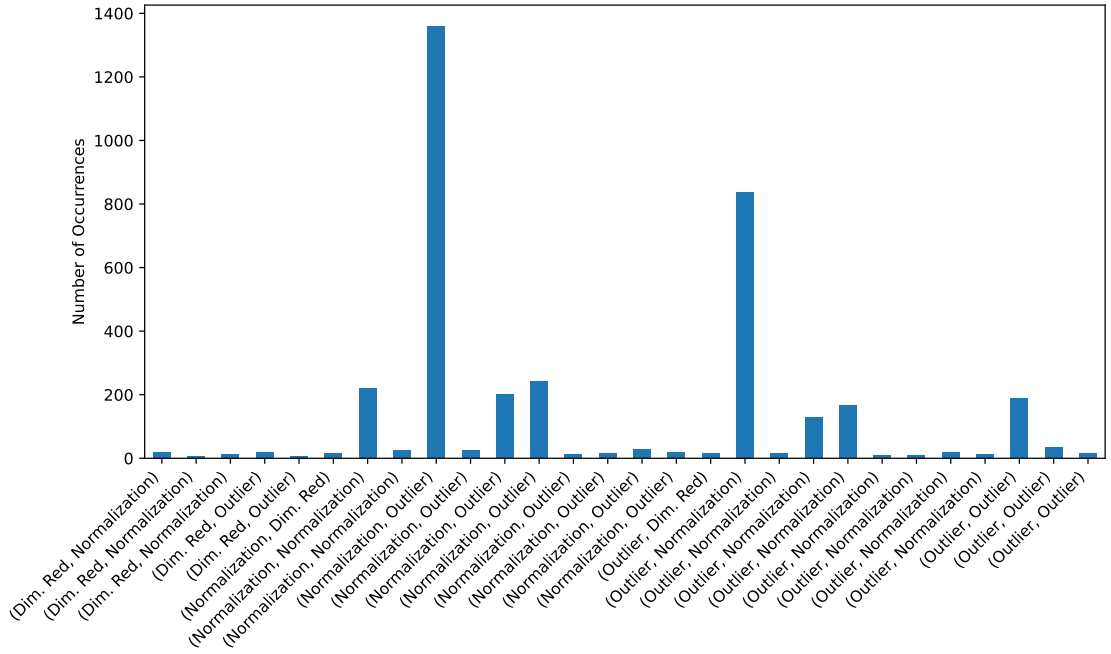


Figure 6.7: Occurrences (> 5) of Preprocessing Method Pairs

Key Finding: In the overall results, it stands out that normalization is more often applied before outlier removal than the other way around.

Figure 6.7 shows the occurrences of preprocessing method group combinations. For better readability, only pipelines of length two and combinations that occur more than five times are depicted. In addition, the dimensionality reduction group is abbreviated to Dim. Red.

The results show that 1358 pipelines of length two apply a normalization method first and then an outlier removal method, compared to 838 pipelines that do it the other way around. This result shows no clear picture of what to do first, but there is a tendency towards scaling. No difference could be found concerning data characteristics or data manipulation methods. For pipelines of length two that contain dimensionality reduction or for longer pipelines, there is no clear pattern of what method group should come first.

To evaluate if the order makes a difference, pipelines of length two or longer are further investigated. They are divided into groups according to their pipeline length, which results in five groups ranging from two to six. For all groups, 100 random samples are selected. If a group contains fewer samples, all are selected. For every one of these samples, 20 permutations are built. The high number is chosen because of the longer pipelines. If fewer permutations are possible, all possible permutations are created.

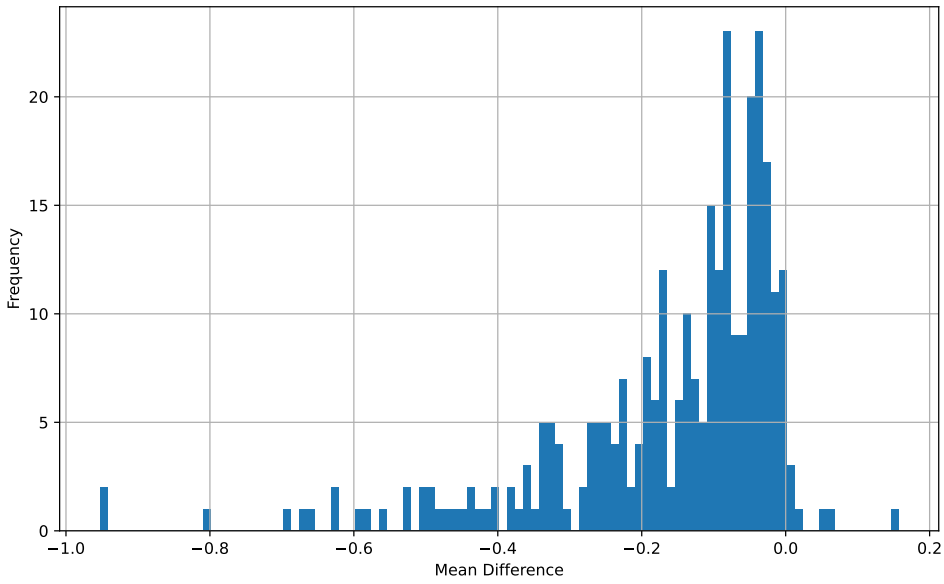


Figure 6.8: Change in ARI After Order Change

Key Finding: The order does not influence the results of the preprocessing pipeline by much.

Figure 6.8 shows mean the difference of all permutations compared to the corresponding best pipeline of the optimization result. The results make the statements from the previous paragraph seem much weaker. While the pipelines do worsen sometimes, the change of the ARI is usually around -0.1. That indicates no significant difference in terms of clustering results, and therefore, the order does not matter too much.

6.6 Meta-Features

This section covers the evaluation of meta-features, with a focus on how the data imperfections are reflected in the meta-features (cf. Section 6.6.1), whether the general meta-feature space can be reduced, and how the optimization is reflected in the meta-features.

6.6.1 Effect of Data Manipulations

To obtain a uniform number of meta-features per dataset, the mean and standard deviation are calculated across the features for most meta-features (cf. Section 5.2). Table 6.9 now shows the mean of the mean and the mean of the standard deviation per meta-feature and data manipulation group. For the groups that contain missing values (M, M + S, M + O, M + S + O), all missing values are replaced with zero because otherwise most of the meta-features could not be calculated.

Key Finding: All data manipulations correlate with parts of the meta-features

The results of Table 6.9 are shown to assess what meta-features are relevant and how the data imperfections are reflected in the meta-features. The latter is observed by taking the dataset without data imperfections (N) as a baseline and asses which meta-features have a significant difference compared to the datasets without manipulations (N). A considerable difference is assumed if the mean or standard deviation differs by 50% or more. The threshold is set relatively high at 50% to ensure that only the meta-features with the most significant change stick out. These values are marked with an underline in Table 6.9.

Some meta-features do not change significantly across all datasets. These are the number of features (nr_attr), the number of rows (nr_inst), the correlation between data pairs (cor), the number of correlations between features (nr_cor_attr), the number of features with at least one outlier (nr_outliers), and the average number of features per row (t2). Some of these are expected not to change drastically or not change at all because of how the data imperfections are added (cf. Section 5.3.1). Within this group fall nr_attr, nr_inst, and t2. The mean number of features does not change because of the data generation (cf. Section 6.1.3). The mean number of rows differs only slightly because the only occasion where rows are added is during the addition of outliers, and this addition is at max 30% of the number of rows (cf. Section 5.3), and t2 does not change because all rows contain the same number of features. The other two (cor, nr_cor_attr) are expected to correlate because of their dependence and most likely do not change that drastically because the initial data generation with SKlearn [PVG+11] most likely does not produce correlated features, and the data imperfections do not change the data to create them. The meta-features nr_outlier changes even if there are no outliers added because the data is not perfectly distributed due to the change in the cluster standard deviation, and some parts of skewed data could be considered outliers as well. It is higher for datasets that contain outliers compared to those that do not, so there is a small indication that this meta-feature might help, but others might better identify outliers. In general, most of these meta-features might help in scenarios where the data is generated differently or if it's real data, but they do not contribute to this type of data and generation method.

The evaluation of whether missing values can be detected is overshadowed by the fact that missing values are replaced with a constant value. However, this is not true for the meta-feature has_missing_values, which clearly indicates if missing values exist in the datasets. Additionally, the meta-features are often similar to the dataset without data imperfections (N), only the median and the number of features that are normally distributed (nr_norm). Both are more likely to change due to the replacement with a constant value than because of missing values.

Meta Features		N	M	S	O	M+S	M+O	S+O	M+S+O
nr_attr		30.00	30.00	30.00	30.00	30.00	30.00	30.00	30
nr_inst		2166.67	2166.67	2166.67	2600.00	2166.67	2600.00	2600.00	2600.00
has_missing_values		0	1	0	0	1	1	0	1
cor	mean	0.10	0.08	0.09	0.06	0.08	0.05	0.06	0.05
	sd	0.07	0.06	0.07	0.04	0.06	0.04	0.04	0.04
cov	mean	5.70	3.98	1.11e+14	6.46	8.14e+13	4.84	1.54e+16	1.10e+16
	sd	4.27	3.00	4.71e+14	4.87	3.82e+14	3.66	6.78e+16	4.83e+16
eigenvalues	mean	75.58	62.29	3.99e+32	138.85	3.92e+32	114.28	3.19e+35	2.72e+35
	sd	38.52	27.21	2.15e+33	45.49	2.10e+33	34.63	1.72e+36	1.47e+36
iq_range	mean	11.96	9.08	55.62	13.60	20.77	10.27	6.23e+07	1.55e+05
	sd	0.95	0.86	228.76	0.98	62.22	0.89	4.03e+08	8.99e+05
kurtosis	mean	-0.37	0.18	26.63	0.73	26.80	1.54	0.74	2.83
	sd	0.18	0.22	115.31	0.34	112.60	0.42	0.49	5.63
mad	mean	8.82	6.83	10.83	10.05	8.21	7.71	14.21	9.62
	sd	0.69	0.66	9.50	0.71	6.83	0.68	19.73	9.20
max	mean	25.20	24.84	2.80e+16	34.84	2.70e+16	34.74	3.36e+16	3.37e+16
	sd	2.36	2.38	1.41e+17	2.88	1.34e+17	2.91	1.69e+17	1.69e+17
mean	mean	-0.14	-0.11	6.49e+12	-0.16	6.24e+12	-0.13	5.01e+15	4.12e+15
	sd	1.15	0.95	3.16e+13	1.16	2.99e+13	0.96	2.52e+16	2.07e+16
median	mean	-0.19	-0.05	0.58	-0.20	0.35	-0.04	1.81	0.61
	sd	1.38	0.32	4.50	1.36	2.00	0.26	10.82	3.05
min	mean	-25.28	-24.89	-28.21	-35.02	-27.78	-34.93	-39.08	-38.44
	sd	2.56	2.59	19.81	3.15	19.53	3.16	27.60	27.24
nr_cor_attr		0.01	0.0	0.01	0.0	0.0	0.0	0.0	0.0
nr_norm		11.67	0.97	11.39	4.32	0.95	0.14	4.25	0.12
nr_outliers		19.89	27.65	20.05	27.83	27.69	29.82	27.65	29.18
range	mean	50.48	49.73	2.80e+16	69.86	2.70e+16	69.67	3.36e+16	3.37e+16
	sd	3.42	3.48	1.41e+17	4.80	1.34e+17	4.83	1.69e+17	1.69e+17
skewness	mean	0.03	0.02	0.64	0.0	0.63	-0.0	0.08	0.11
	sd	0.14	0.13	2.64	0.21	2.63	0.23	0.37	0.54
t_mean	mean	-0.18	-0.13	13.12	-0.19	3.33	-0.13	4.88e+14	4.62e+13
	sd	1.27	0.92	70.94	1.25	18.85	0.90	2.45e+15	2.33e+14
var	mean	75.58	62.29	3.99e+32	138.85	3.92e+32	114.28	3.19e+35	2.72e+35
	sd	7.08	5.99	2.15e+33	14.85	2.10e+33	12.54	1.72e+36	1.47e+36
t2		0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.02
t3		0.03	0.03	0.01	0.02	0.01	0.02	0.01	0.01
t4		0.90	0.92	0.36	0.93	0.37	0.94	0.24	0.25
wg_dist	mean	1.25	1.15	1.23	1.11	1.14	1.01	1.13	1.03
	sd	0.07	0.08	0.07	0.26	0.08	0.24	0.30	0.27

Table 6.9: Meta-Feature of the Not Preprocessed Data

All data manipulation groups with skewed data (S, M + S, S + O, M + S + O) show a high change for many meta-features. The most extreme ones are the covariance (cov), the eigenvalues, the maximum, the mean, range, and variance (var). All indicate highly skewed data, and that matches the expectation.

The meta-features changes introduced by outliers are best shown in columns O and M + O of Table 6.9 because for other data manipulation groups containing outliers, they are most likely overshadowed by the changes introduced when skewing the data. The changes are at best shown in the meta-features eigenvalues, kurtosis, variance (var), and the weighted distance (wg_dist). These meta-features indicate a higher variance or that the data has longer tails compared to a normal distribution. Therefore, they are likely to indicate the presence of outliers.

6.6.2 Correlation of Meta-Features

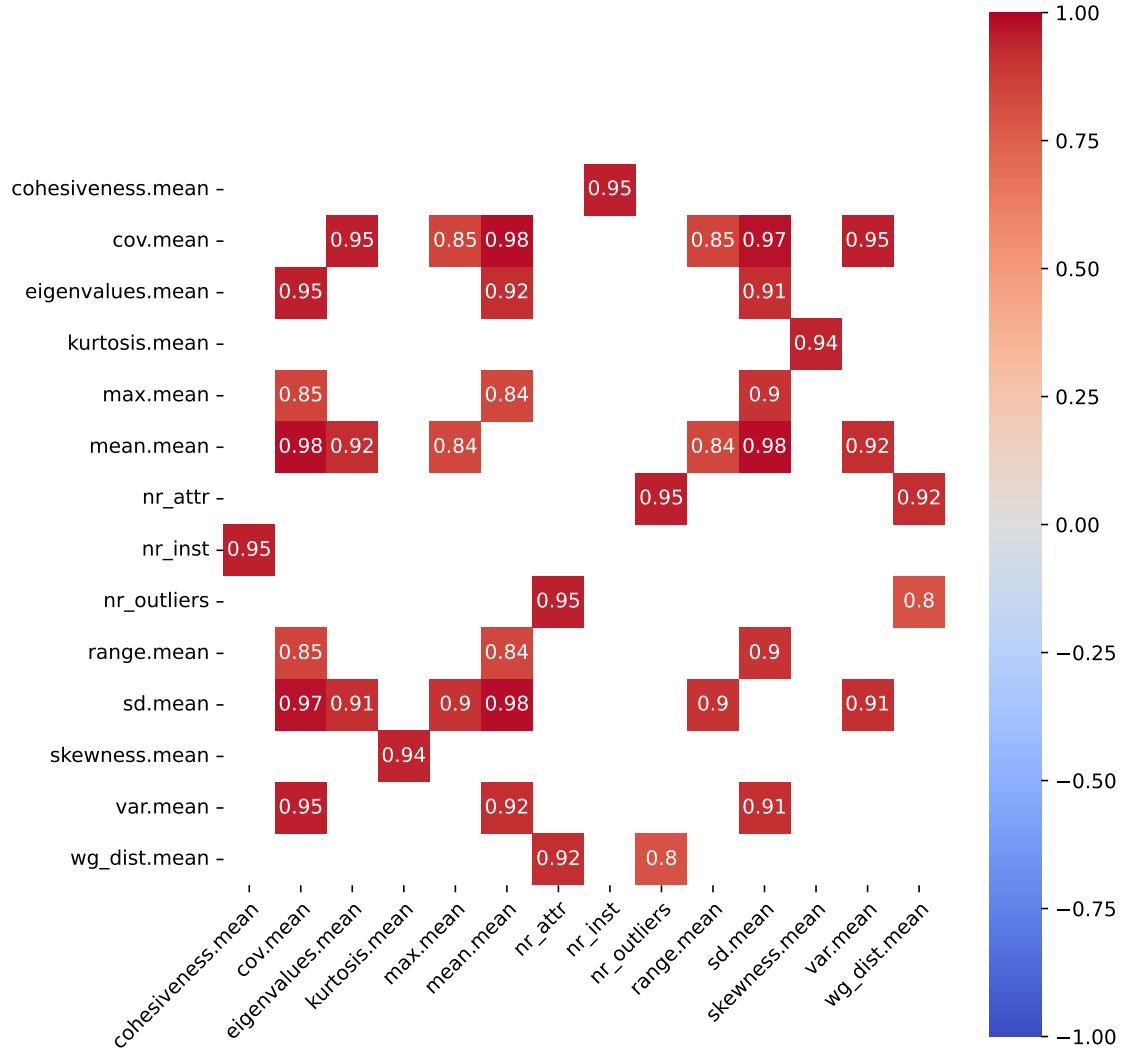


Figure 6.9: Strong Correlation of Meta-Features

Many meta-features are selected and calculated for all datasets. However, it is not clear if all are necessary for abstracting the data well. In order to evaluate this, the Person Correlation Coefficient [Pea95] is calculated between all meta-features. Because of the high number of meta-features and the fact that a mean and a standard deviation value exist for most meta-features, only a subset is presented. This subset is depicted in Figure 6.9 and contains only mean values of the meta-features and only those pairs that have a correlation coefficient of > 0.8 or < -0.8. A full correlation matrix is provided in the Appendix (cf. Figure A.1)

Key Finding: The meta-feature space could be further reduced since some meta-features strongly correlated with each other.

The correlation matrix indicates that some meta-features strongly correlate with each other. This shows that not all meta-features are necessary, and maybe some could be left out. An example of that is that the mean meta-feature is positively correlated to the max meta-features because if the overall maximum is higher, the mean is higher as well. However, since this is a reduced selection and there might still be special cases where a meta-feature or a combination of meta-features provides helpful insights, these meta-features should not be removed without further validation.

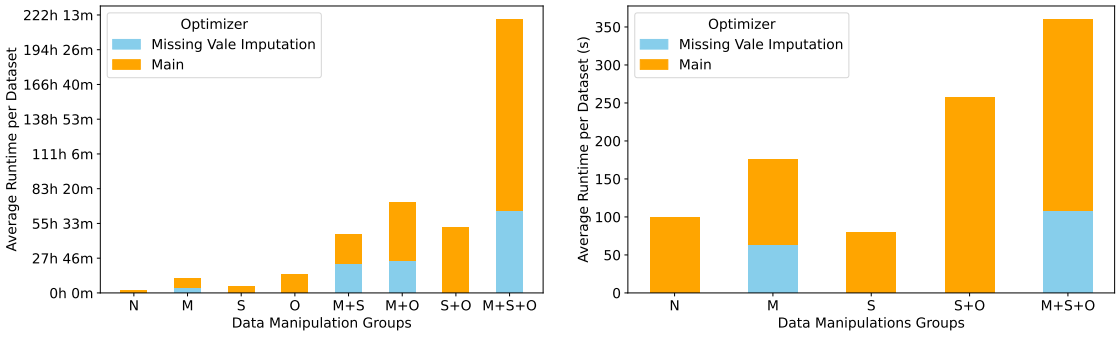
6.7 Runtime

This section covers the runtime analysis of the evaluation to show, in general, which runtime the approach requires (cf. Section 6.7.1) and how consistent the runtime is across multiple runs (cf. Section 6.7.2).

6.7.1 Runtime of Data Manipulation Groups

Key Finding: All datasets require a total of 20 days for optimization, whereby the main factor for the runtime is the sample size

The overall runtime for all data manipulation groups is around 20 days. These 20 days include only the optimization process for the datasets. Other parts, like reading and writing files, are excluded. Figure 6.10a depicts the overall runtime per dataset. The two different colors indicate the two different optimization processes where blue is the time of the missing value imputation, and orange is the time needed for the main optimization. The M + S + O manipulation group sticks the most out due to the huge amount of datasets within this group (2187).



(a) Overall Runtime per Data Manipulation Group **(b)** Average Dataset Runtime per Data Manipulation Group

Figure 6.10: Runtimes of Data Manipulation Groups

To provide a better overview, Figure 6.10b shows the average runtime per dataset. The O, M + S, M + O datasets are excluded in this comparison because they have been run on different servers, and a fair comparison is not possible due to the hardware differences (cf. Section 6.1.2). The differences in the average runtime compared across the data manipulation groups can be explained by three reasons. First, every dataset that contains missing values is optimized twice, once for the

missing value imputation and once for the other preprocessing methods. Second, the N and M data manipulation groups are randomly initialized within the main optimization because they do not contain data manipulations that would allow a tailored initialization. This means they may take longer to find a good solution and explain the difference between those two groups and the S group regarding runtime. The last reason is the length of the pipeline. The S + O and M + S + O have a longer runtime because they are initialized with two preprocessors during the main optimization and, therefore, each generation needs longer to evaluate. Besides that the most influential factor of the runtime is the sample size. Figure 6.11 showcases this exemplarily for the M+S+O manipulation group, where the datasets with 5000 samples take almost five times longer than the ones with 1000 samples. All other data characteristics, like the number of features or the standard deviation of the cluster, do not impact the runtime drastically.

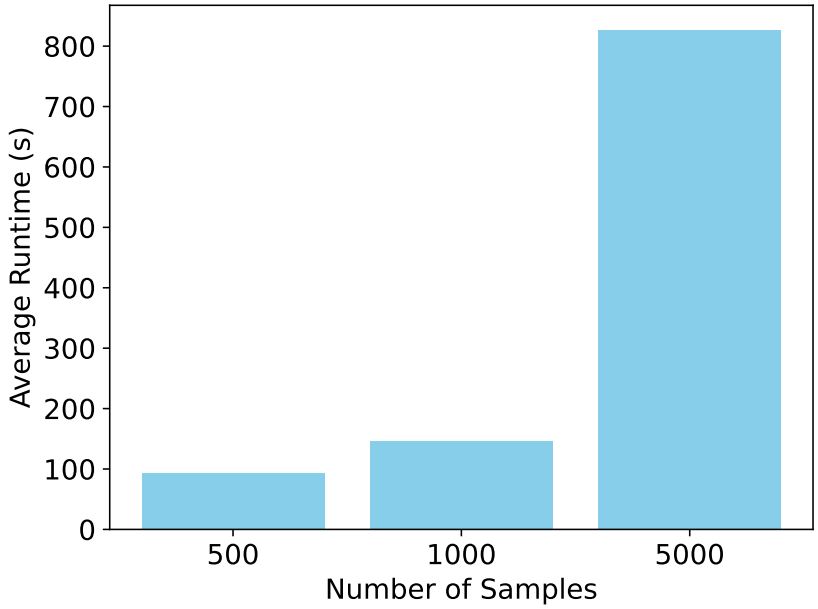


Figure 6.11: Runtime by Sample Size (M+S+O)

6.7.2 Consistency of the Runtime

Data Manipulation Group	Mean Runtime of Each Run (s)	Overall Mean	Standard Deviation
M	{65.17, 65.4, 52.1, 64.41, 66.2}	62.66	0
S	{107.83, 104.53, 106.03, 98.74, 108.04}	105.03	0
O	{233.21, 241.85, 214.18, 190.83, 237.53}	223.52	0

Table 6.10: Consistency of Multiple Runs

Key Finding: Runtimes vary a little but is overall consistent

To calculate the consistency of the runtime, a similar approach as in Section 6.2.3 is applied. The M, O, and S data manipulation groups are run five times on the same hardware. The results are presented in Table 6.10. Column one shows the name of the data manipulation group, column two is the mean runtime in seconds of each run, and columns three and four are the overall mean and standard deviation. All results are rounded to the second decimal place.

The runtimes are not as consistent as the ARI (cf. Section 6.2.3). The likely origin of that is that there are preprocessing pipelines that take longer than others, and while only the best is reflected in the final score, others have to be evaluated and, therefore, influence the runtime. However, besides an outlier in the M and O group, the runtime lies within 5% of the mean of all runtimes and, therefore, can be considered relatively consistent.

6.8 Summary

This section serves as a brief summary of the previous sections. It is overall shown that the presented concept significantly improves the clustering result from a median ARI of 0.08 to 0.89, while the datasets that are manipulated with a skew benefit the most. Those datasets that were processed with the missing value imputation optimization and the main optimization benefit more from the main optimization, with the exception of the datasets that are only manipulated with missing values. Perfect scores of ARI = 1 are achieved mainly by the manipulation groups that already have a high ARI without preprocessing. These are the groups without data manipulations and the ones that contain missing values. Datasets with a low ARI of < 0.1 are all skewed. Repeated runs for a subset of all data indicated that the scores are consistent and well reproducible.

The evaluation of the optimization process shows that well-performing pipelines are found within the first few generations of the respective optimizer. Afterward, the pipelines still improved but with a much smaller increase. A clear plateau is not reached, which indicates that with more generations a small increase might be possible. The applied time limit of ten minutes stops 11.96% of the pipelines during their optimization process. All of them have the highest here evaluated sample size of 5000. Before the pipelines were stopped, it was possible to run over half of the maximum generations (50). This means a preprocessing optimization has happened, but there still might be some improvement possible with a longer time limit. The analysis of the lengths of the pipelines shows that they do not shrink the main optimization process but rather keep their initial length or grow by a method.

Analyzing the patterns that occur in pipelines, it is shown that the KNN-Imputer is the preferred method for missing value imputation. The reason for this is that it includes all features at the same time for imputing while only considering the feature where the value is missing. Such a clear observation could not be made for the rest of the preprocessing methods groups. In general, dimensionality reduction is significantly underrepresented. This is due to the fact that there are no specific data manipulations for dimensionality reduction.

Two baselines are evaluated: one where a random pipeline is provided as a preprocessing pipeline and one where a pipeline based on the data manipulations of the dataset is initialized. Both improve the ARI result compared to no preprocessing but are significantly outperformed by the concept of this thesis.

The order in which the preprocessing methods are executed within pipelines has been specially analyzed. The results show that there are patterns if all the best pipelines of all datasets are evaluated. However, a further permutation test with pipelines could not confirm that the order makes a significant difference. The results worsen around 0.1 in terms of ARI.

The meta-feature analysis shows that every data manipulation can be clearly identified by meta-features. Additionally, correlations between meta-features are found, suggesting that the overall meta-feature space for this concept could be further reduced.

The optimization process for all datasets runs for 20 days. Additionally, consistency is evaluated for a subset of the data. Some variation is shown, but overall, the runtime is rather consistent if an optimization process is run multiple times.

7 Conclusion

Preprocessing plays a crucial role in extracting knowledge from data. However, several challenges arise during this process. A user is confronted with a large selection of preprocessing methods, and it is often unclear which one to choose because this depends on the general data characteristics and the data imperfections. Secondly, these preprocessors have parameters that can influence the behavior, and it is unclear which parameter values are suitable for a given dataset. To complicate matters further, multiple preprocessing methods can often be applied, resulting in a preprocessing pipeline. Therefore, the effect of the preprocessing methods on each other and the order of preprocessing methods within a pipeline may play a role as well. All these challenges lead to a huge search space, and it can be really challenging to find a good preprocessing pipeline for a given dataset, especially for inexperienced users.

To mitigate the challenges described above, the overall goal is to make accurate suggestions for preprocessing pipelines that improve the clustering result. In order to achieve this, it is important to identify data imperfections and to understand the relationship between data imperfections and preprocessing pipelines. This thesis contributes a first step in that direction with the concept of how a knowledge base can be created that accurately measures the effects of preprocessing pipelines on data with imperfections and can identify data imperfections of datasets. In order to achieve the latter, meta-features are evaluated because they are a good way to describe unseen datasets and their imperfections. Such a knowledge base needs to contain information from many different datasets to be able to generalize. Because of that, synthetic data is used and further manipulated with data imperfections to have an almost unlimited dataset available and to make a precise evaluation of what data imperfection is handled by which pipeline well. These manipulations skew the data distribution, remove parts of the data to create missing values, and add outliers to the data. Because of the named challenges of section one, it is impossible to apply all combinations of preprocessing methods and their hyperparameters to a dataset, even if the selection of preprocessing methods is small. Instead, pipelines are generated and refined during an optimization process. As optimization, Genetic optimization is used because it provides high flexibility and can be well customized to address the named challenges. Additionally, a single clustering algorithm is used to evaluate the preprocessing pipelines, and the parameters of this clustering algorithm are kept constant. The reason for that is that the effect of preprocessing can be better evaluated with a single algorithm and constant parameters.

The evaluation shows that for most datasets, a preprocessing pipeline is found by the optimizer, which leads to a significant improvement in the clustering results compared to the results without preprocessing. Therefore, the presented concept works for this type of dataset and data imperfections. The improvement is most significant for skewed distribution data, while datasets that have not been manipulated show the slightest improvement. Additionally, the evaluation of the optimization process shows that a well-performing pipeline is found relatively quickly, while the improvement afterward exists but is comparatively small. It is shown that the missing value imputation works best with the KNN-Imputer compared to other imputation techniques. Other data imperfections do

not produce a preferred method or pipeline. Concerning the order of the preprocessing methods within a pipeline, it could not be shown that there is a significant difference. The concept is evaluated against two baselines, one that performs a random preprocessing pipeline and one that performs a preprocessing pipeline that is initialized based on the data manipulation methods applied to this dataset. The presented concept outperforms both baselines significantly. However, the baselines achieve better scores than applying clustering without preprocessing. Additionally, it is demonstrated that meta-features correlate with data imperfections. Therefore, it suggests that it is possible to determine the need for preprocessing and the identification of data imperfections with the thesis used meta-features.

The evaluation shows that overall, a significant improvement of the ARI is obtained, and t The datasets that are skewed benefit the most from the optimization, while datasets without data manipulation the least. Further analysis of the development of the ARI during the optimization process shows that a well-performing pipeline is found relatively fast, and afterward, the ARI improves slightly.

The evaluation of the meta-features shows that meta-features can be identified for each data manipulation method, which gives a clear indication of which preprocessing is necessary and which manipulations may be present.

7.1 Outlook

This thesis presents a concept for finding good-performing preprocessing pipelines with the help of an optimizer to improve the overall clustering result for synthetically generated data with known data imperfections. While this work analyses different data characteristics, meta-features, and preprocessing methods, there are still points that could be further investigated or changed.

The main result of this work is a knowledge base that contains meta-features, preprocessing pipelines, and ARI scores. The next step could be to use this knowledge base to apply meta-learning and make suggestions for preprocessing pipelines for unseen datasets. Such a meta-learning approach could be calculating the distance between the meta-features of not preprocessed data and the ones from the unseen dataset and providing the closest pipeline as a suggestion. Another approach would be to train a model like a decision tree that predicts a preprocessing pipeline based on the meta-features.

Another approach could include different data to provide a more robust knowledge base that may generalize better. For example, different data manipulations could be introduced, like imbalanced clusters or completely different datasets like real-world and benchmark data from sources like the UCI Machine Learning Repository[1] could be included to achieve a wider variety of datasets.

The optimizer plays a crucial part in finding good-performing pipelines. This work presents one implementation for how such an optimizer could be realized. However, several changes could be further researched. First, the parameters of the optimizer can be changed to investigate the influence of those. Second, there may be other strategies related to initialization, mutation, and crossover that impact clustering or are able to find better pipelines. One example is that the crossover swaps whole

[1] https://archive.ics.uci.edu/

methods of a pipeline and not only the parameters. Apart from that, a multi-objective optimization with a second goal besides the ARI, like the pipeline length or runtime, could be realized. This could lead to a more efficient optimization process. However, it may reduce the quality of the pipeline in terms of achieved ARI. In addition, different types of optimizers, such as Bayesian-based optimizers, could also be implemented to better be able to compare the presented approach.

Bibliography

[ASR+20] E. Alcobaça, F. Siqueira, A. Rivolli, L. P. Garcia, J. T. Oliva, A. C. De Carvalho. "MFE: Towards reproducible meta-feature extraction". In: *Journal of Machine Learning Research* 21.111 (2020), pp. 1–5 (cit. on p. 44).

[BAI+22] M. Bilal, G. Ali, M. W. Iqbal, M. Anwar, M. S. A. Malik, R. A. Kadir. "Auto-prep: efficient and automated data preprocessing pipeline". In: *IEEE Access* 10 (2022), pp. 107764–107784 (cit. on pp. 31, 33).

[BB12] J. Bergstra, Y. Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012) (cit. on p. 28).

[Bis06] C. M. Bishop. "Pattern recognition and machine learning". In: *Springer google schola* 2 (2006), pp. 645–678 (cit. on p. 28).

[BKNS00] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104 (cit. on p. 49).

[CCK+00] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth, et al. "CRISP-DM 1.0: Step-by-step data mining guide". In: *SPSS inc* 9.13 (2000), pp. 1–73 (cit. on p. 17).

[CL18] D. Corne, M. A. Lones. "Evolutionary Algorithms". In: *Handbook of Heuristics*. Springer International Publishing, 2018, pp. 1–22. ISBN: 9783319071534. DOI: 10.1007/978-3-319-07153-4_27-1. URL: http://dx.doi.org/10.1007/978-3-319-07153-4_27-1 (cit. on p. 24).

[Col99] D. A. Coley. *An introduction to genetic algorithms for scientists and engineers*. World Scientific Publishing Company, 1999 (cit. on pp. 23, 24).

[DG03] D. L. Donoho, C. Grimes. "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data". In: *Proceedings of the National Academy of Sciences* 100.10 (2003), pp. 5591–5596 (cit. on p. 49).

[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*. Vol. 96. 34. 1996, pp. 226–231 (cit. on pp. 28, 29, 32).

[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)*. IEEE. 2021, pp. 1119–1126 (cit. on pp. 33–35).

[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)*. IEEE. 2022, pp. 1144–1153 (cit. on pp. 18, 31–33).

[FD07] B. J. Frey, D. Dueck. “Clustering by passing messages between data points”. In: *science* 315.5814 (2007), pp. 972–976 (cit. on pp. 28, 32).

[FDG+12] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné. “DEAP: Evolutionary Algorithms Made Easy”. In: *Journal of Machine Learning Research* 13 (July 2012), pp. 2171–2175 (cit. on pp. 32, 34, 37, 56, 58).

[FEF+22] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, F. Hutter. “Auto-sklearn 2.0: Hands-free automl via meta-learning”. In: *Journal of Machine Learning Research* 23.261 (2022), pp. 1–61 (cit. on pp. 18, 36).

[FH75] K. Fukunaga, L. Hostetler. “The estimation of the gradient of a density function, with applications in pattern recognition”. In: *IEEE Transactions on information theory* 21.1 (1975), pp. 32–40 (cit. on pp. 28, 32).

[FKE+15a] 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)*. 2015, pp. 2962–2970 (cit. on p. 36).

[FKE+15b] 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) (cit. on p. 33).

[FKH18] S. Falkner, A. Klein, F. Hutter. “BOHB: Robust and efficient hyperparameter optimization at scale”. In: *International conference on machine learning*. PMLR. 2018, pp. 1437–1446 (cit. on pp. 28, 33).

[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. 17).

[FS18] P. Fränti, S. Sieranoja. “K-means properties on six clustering benchmark datasets”. In: *Applied intelligence* 48 (2018), pp. 4743–4759 (cit. on p. 28).

[GLH15] S. García, J. Luengo, F. Herrera. *Data preprocessing in data mining*. Vol. 72. Springer, 2015 (cit. on pp. 21, 24, 48).

[GLT+19] P. Gijsbers, E. LeDell, J. Thomas, S. Poirier, B. Bischl, J. Vanschoren. “An open source AutoML benchmark”. In: *arXiv preprint arXiv:1907.00909* (2019) (cit. on p. 34).

[Gre86] J. J. Grefenstette. “Optimization of control parameters for genetic algorithms”. In: *IEEE Transactions on systems, man, and cybernetics* 16.1 (1986), pp. 122–128 (cit. on p. 23).

[HA85] L. Hubert, P. Arabie. “Comparing partitions”. In: *Journal of classification* 2 (1985), pp. 193–218 (cit. on pp. 22, 23).

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten. “The WEKA data mining software: an update”. In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18 (cit. on p. 36).

[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, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*. Springer. 2011, pp. 507–523 (cit. on p. 33).

[HMT11] N. Halko, P.-G. Martinsson, J. A. Tropp. “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions”. In: *SIAM review* 53.2 (2011), pp. 217–288 (cit. on p. 49).

[HMW+20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. on p. 56).

[HPT22] J. Han, J. Pei, H. Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022 (cit. on pp. 21, 22, 48).

[HTFF09] T. Hastie, R. Tibshirani, J. H. Friedman, J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009 (cit. on p. 46).

[Hun07] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. on p. 56).

[Koz94] J. R. Koza. “Genetic programming as a means for programming computers by natural selection”. In: *Statistics and computing* 4 (1994), pp. 87–112 (cit. on p. 23).

[KRP+16] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing. “Jupyter Notebooks – a publishing format for reproducible computational workflows”. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides, B. Schmidt. IOS Press. 2016, pp. 87–90 (cit. on p. 56).

[KZD16] K. Kirchner, J. Zec, B. Delibašić. “Facilitating data preprocessing by a generic framework: a proposal for clustering”. In: *Artificial Intelligence Review* 45 (2016), pp. 271–297 (cit. on p. 17).

[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 (cit. on p. 37).

[LJD+18] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar. “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52 (cit. on p. 28).

[LLT21] Y. Liu, S. Li, W. Tian. “Autocluster: Meta-learning based ensemble method for automated unsupervised clustering”. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2021, pp. 246–258 (cit. on pp. 28–30).

[LTZ12] F. T. Liu, K. M. Ting, Z.-H. Zhou. “Isolation-based anomaly detection”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.1 (2012), pp. 1–39 (cit. on p. 49).

[LWWT19] Y.-F. Li, H. Wang, T. Wei, W.-W. Tu. “Towards automated semi-supervised learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4237–4244 (cit. on p. 28).

[Mac+67] J. MacQueen et al. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297 (cit. on pp. 22, 28, 29, 32, 43).

[McK10] W. McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt, J. Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a (cit. on p. 56).

[MG+95] B. L. Miller, D. E. Goldberg, et al. “Genetic algorithms, tournament selection, and the effects of noise”. In: *Complex systems* 9.3 (1995), pp. 193–212 (cit. on p. 24).

[Min00] T. Minka. “Automatic choice of dimensionality for PCA”. In: *Advances in neural information processing systems* 13 (2000) (cit. on pp. 40, 49).

[Mit98] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998 (cit. on p. 23).

[MR93] A. Maćkiewicz, W. Ratajczak. “Principal components analysis (PCA)”. In: *Computers & Geosciences* 19.3 (1993), pp. 303–342 (cit. on p. 39).

[NH19] L. H. Nguyen, S. Holmes. “Ten quick tips for effective dimensionality reduction”. In: *PLoS computational biology* 15.6 (2019), e1006907 (cit. on p. 48).

[OBUM16] R. S. Olson, N. Bartley, R. J. Urbanowicz, J. H. Moore. “Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO ’16. Denver, Colorado, USA: ACM, 2016, pp. 485–492. ISBN: 978-1-4503-4206-3. DOI: 10.1145/2908812.2908918. URL: http://doi.acm.org/10.1145/2908812.2908918 (cit. on p. 37).

[OM16] R. S. Olson, J. H. Moore. “TPOT: A tree-based pipeline optimization tool for automating machine learning”. In: *Workshop on automatic machine learning*. PMLR. 2016, pp. 66–74 (cit. on pp. 18, 33, 34).

[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)*. IEEE. 2020, pp. 1220–1225 (cit. on p. 30).

[PDK24] Y. Poulakis, C. Doulkeridis, D. Kyriazis. “A Survey on AutoML Methods and Systems for Clustering”. In: *ACM Transactions on Knowledge Discovery from Data* (2024) (cit. on pp. 24, 27, 32).

[Pea01] K. Pearson. “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 (cit. on p. 18).

[Pea95] K. Pearson. “VII. Note on regression and inheritance in the case of two parents”. In: *proceedings of the royal society of London* 58.347-352 (1895), pp. 240–242 (cit. on p. 76).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 28, 29, 33, 35, 36, 40, 46–48, 55, 56, 74).

[QPHW24] D. Qi, J. Peng, Y. He, J. Wang. “Auto-FP: An Experimental Study of Automated Feature Preprocessing for Tabular Data”. In: (2024) (cit. on pp. 33, 34).

[RAHL19] E. Real, A. Aggarwal, Y. Huang, Q. V. Le. “Regularized evolution for image classifier architecture search”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4780–4789 (cit. on p. 33).

[Ras99] C. Rasmussen. “The infinite Gaussian mixture model”. In: *Advances in neural information processing systems* 12 (1999) (cit. on p. 29).

[RD99] P. J. Rousseeuw, K. V. Driessen. “A fast algorithm for the minimum covariance determinant estimator”. In: *Technometrics* 41.3 (1999), pp. 212–223 (cit. on p. 49).

[Ree23] C. Reed. “Analysis and Integration of Data Preprocessing Steps in AutoML for Clustering”. Bachelor’s Thesis. University of Stuttgart, 2023 (cit. on pp. 18, 35, 36, 46).

[RGS+18] A. Rivolli, L. P. Garcia, C. Soares, J. Vanschoren, A. C. de Carvalho. “Characterizing classification datasets: a study of meta-features for meta-learning”. In: *arXiv preprint arXiv:1808.10406* (2018) (cit. on pp. 24, 25).

[RK87] L. Rdusseeun, P. Kaufman. “Clustering by means of medoids”. In: *Proceedings of the statistical data analysis based on the L1 norm conference, neuchatel, switzerland*. Vol. 31. 1987 (cit. on p. 28).

[RR14] W. Raghupathi, V. Raghupathi. “Big data analytics in healthcare: promise and potential”. In: *Health information science and systems* 2 (2014), pp. 1–10 (cit. on p. 17).

[RS00] S. T. Roweis, L. K. Saul. “Nonlinear dimensionality reduction by locally linear embedding”. In: *science* 290.5500 (2000), pp. 2323–2326 (cit. on p. 49).

[RVBV16] S. Romano, N. X. Vinh, J. Bailey, K. Verspoor. “Adjusting for chance clustering comparison measures”. In: *Journal of Machine Learning Research* 17.134 (2016), pp. 1–32 (cit. on p. 51).

[SB21] G. Steinbuss, K. Böhm. “Generating artificial outliers in the absence of genuine ones—A survey”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15.2 (2021), pp. 1–37 (cit. on p. 47).

[Scu10] D. Sculley. “Web-scale k-means clustering”. In: *Proceedings of the 19th international conference on World wide web*. 2010, pp. 1177–1178 (cit. on p. 28).

[SDB+93] W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, H. De Garis. “An overview of evolutionary computation”. In: *European conference on machine learning*. Springer. 1993, pp. 442–459 (cit. on p. 50).

[SGP+18] B. Schoenfeld, C. Giraud-Carrier, M. Poggemann, J. Christensen, K. Seppi. “Preprocessor selection for machine learning pipelines”. In: *arXiv preprint arXiv:1810.09942* (2018) (cit. on p. 30).

[Tan22] H. M. Tanvir. “Meta-Learning Based Approach for Automated Pre-processing for Clustering”. Master’s Thesis. University of Tartu, 2022 (cit. on p. 34).

[TB99] M. E. Tipping, C. M. Bishop. “Probabilistic principal component analysis”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 61.3 (1999), pp. 611–622 (cit. on p. 49).

[TCS+01] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, R. B. Altman. "Missing value estimation methods for DNA microarrays". In: *Bioinformatics* 17.6 (2001), pp. 520–525 (cit. on p. 49).

[TFS+21] D. Tschechlov, M. Fritz, H. Schwarz, Y. Velegrakis, D. Zeinalipour-Yazti, P. Chrysanthis, F. Guerra. "AutoML4Clust: Efficient AutoML for Clustering Analyses." In: *EDBT*. 2021, pp. 343–348 (cit. on pp. 27, 28, 30, 33, 35).

[TFSM23] D. Treder-Tschechlov, M. Fritz, H. Schwarz, B. Mitschang. "ML2DAC: Meta-Learning to Democratize AutoML for Clustering Analysis". In: *Proceedings of the ACM on Management of Data* 1.2 (2023), pp. 1–26 (cit. on pp. 24, 29).

[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 (cit. on pp. 18, 28).

[TP98] S. Thrun, L. Pratt. "Learning to learn: Introduction and overview". In: *Learning to learn*. Springer, 1998, pp. 3–17 (cit. on p. 24).

[Ult05] A. Ultsch. "Clustering with som: U^* c". In: *Proc. Workshop on Self-Organizing Maps, 2005*. 2005 (cit. on p. 28).

[Van19] J. Vanschoren. "Meta-learning". In: *Automated machine learning: methods, systems, challenges* (2019), pp. 35–61 (cit. on p. 24).

[VD09] G. Van Rossum, F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697 (cit. on p. 55).

[VGO+20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on p. 56).

[VVBT14] J. Vanschoren, J. N. Van Rijn, B. Bischl, L. Torgo. "OpenML: networked science in machine learning". In: *ACM SIGKDD Explorations Newsletter* 15.2 (2014), pp. 49–60 (cit. on pp. 28, 30–32).

[Wil92] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8 (1992), pp. 229–256 (cit. on p. 33).

[YJ00] I.-K. Yeo, R. A. Johnson. "A new family of power transformations to improve normality or symmetry". In: *Biometrika* 87.4 (2000), pp. 954–959 (cit. on p. 49).

[ZB20] K. Zhang, J. S. Bloom. "deepcr: Cosmic ray rejection with deep learning". In: *The Astrophysical Journal* 889.1 (2020), p. 24 (cit. on p. 17).

[ZRL96] T. Zhang, R. Ramakrishnan, M. Livny. "BIRCH: an efficient data clustering method for very large databases". In: *ACM sigmod record* 25.2 (1996), pp. 103–114 (cit. on p. 28).

[ZW06] Z. Zhang, J. Wang. “MLLE: Modified locally linear embedding using multiple weights”. In: *Advances in neural information processing systems* 19 (2006) (cit. on p. 49).

[ZZ04] Z. Zhang, H. Zha. “Principal manifolds and nonlinear dimensionality reduction via tangent space alignment”. In: *SIAM journal on scientific computing* 26.1 (2004), pp. 313–338 (cit. on p. 49).

[ZZY19] T. Zhang, L. Zhong, B. Yuan. “A critical note on the evaluation of clustering algorithms”. In: *arXiv preprint arXiv:1908.03782* (2019) (cit. on pp. 18, 46).

All links were last followed on Mai 21, 2024.

A Evaluation

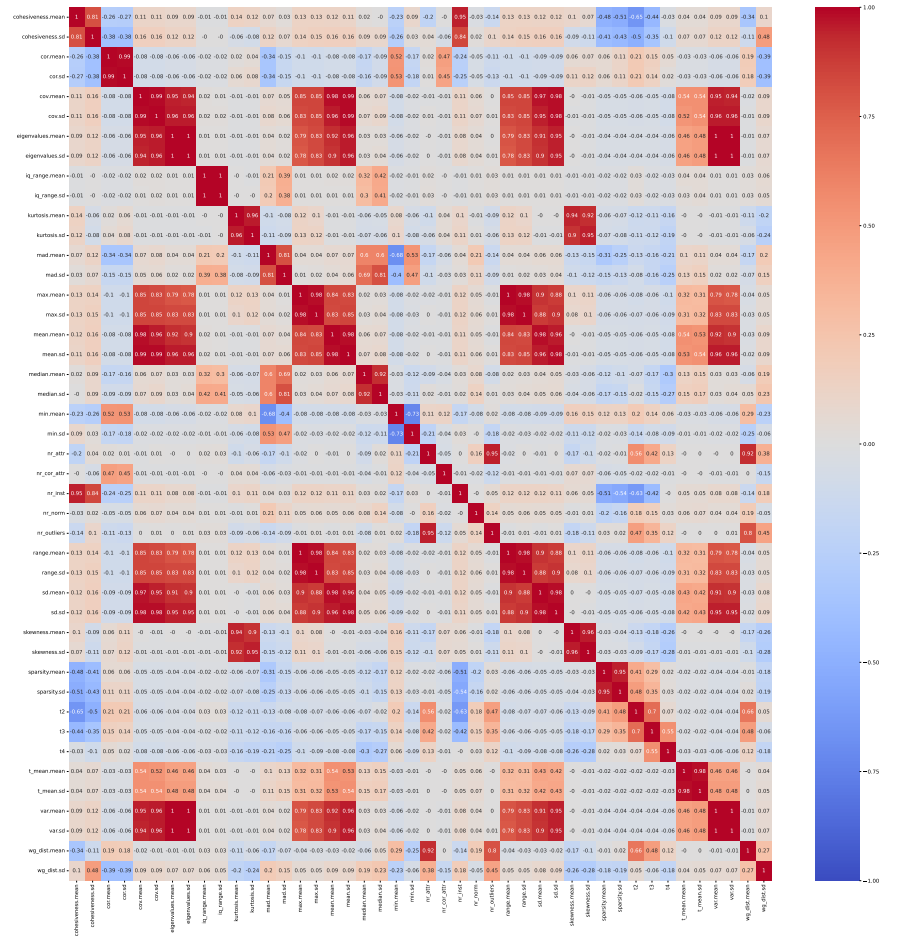


Figure A.1: Correlation of Meta-Features

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