HAW HAMBURG

MASTER THESIS
Chris-Marian Forke

Responsible AI for Clustering Algorithms

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Chris-Marian Forke

Responsible AI for Clustering Algorithms

Master thesis submitted for examination in Master´s degree
in the study course *Master of Science Informatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr.-Ing. Marina Tropmann-Frick
Supervisor: Prof. Dr.-Ing. Olaf Zukunft

Submitted on: March 18, 2025

Chris-Marian Forke

Thema der Arbeit

Verantwortungsvolle KI für Clustering-Algorithmen

Stichworte

Künstliche Intelligence, Verantwortungsvolle KI, Maschinelles Lernen, Unüberwachtes Lernen, Clustering, Fairnessbewertung, Datenschutzbewertung, Sicherheitsbewertung, Erklärbarkeitsbewertung

Kurzzusammenfassung

Verantwortungsvolle AI (RAI) ist ein Thema, das immer wichtiger wird. Um RAI zu entwickeln, ist es notwendig, dass es eine allgemeine Maßeinheit gibt, um den Grad der Verantwortung zu bewerten. Diese Thesis erweitert das Bewertungs-Framework, das in [16] eingeführt wurde. Es kombiniert verschiedene Metriken, die die Umsetzung der verschiedenen Aspekte von RAI messen, um Klassifikations-Modelle zu bewerten. Diese Arbeit bearbeitet die Bewertung von Clusterings.

Dazu wird eine Literaturübersicht über die existierenden Metriken erstellt, die die Fairness (beinhaltet Performance, Individuelle und Gruppen Fairness), den Datenschutz, die Sicherheit und die Erklärbarkeit messen. Es werden auch neue Ansätze vorgestellt. Dann werden einige Metriken ausgewählt, die jeweils genauer bearbeitet werden und zusammen ein leistungsstarkes Tool zur Bewertung der Verantwortung bilden.

Die umfangreiche Bewertung durch diese Metriken wird anhand von fünf Modellen demonstriert. Die Ergebnisse zeigen, dass die Metriken passende Bewertungen geben, besonders, wenn sie in Kombination genutzt werden und somit ihre individuellen Schwachstellen ausgleichen.

Insgesamt bietet die erarbeitete Erweiterung des Bewertungs-Frameworks RAI Forschern und Entwicklern eine gemeinsame Lösung für die Beurteilung der Verantwortung von Clustering-Modellen. Dies ermöglicht unter anderem das Nachvollziehen einer Verbesserung eines Modells durch Anpassungen und das Vergleichen mehrerer Modelle. Das macht diese Arbeit zu einem wichtigen Beitrag zur künftigen Entwicklung und Forschung von RAI.

Chris-Marian Forke

Title of Thesis

Responsible AI for Clustering Algorithms

Keywords

Artificial Intelligence, Responsible AI, Machine Learning, Unsupervised Learning, Clustering, Fairness Evaluation, Privacy Evaluation, Security Evaluation, Explainability Evaluation

Abstract

Responsible AI (RAI) is a topic of increasing importance. The ambition to create RAI leads to the need for a common measurement of the level of responsibility. This thesis extends the solution presented in [16], which combines metrics covering the aspects of RAI in an evaluation framework to evaluate classification tasks. In this thesis, the evaluation of clustering tasks is researched and integrated into this framework.

To achieve this, the available literature for metrics for the evaluation of fairness (including performance, individual fairness, and group fairness), privacy, security, and explainability is analyzed and enhanced with new approaches. A set of these metrics is selected and implemented, forming a powerful evaluation tool for responsibility.

The extensive assessment using this set is demonstrated on five different clustering models. The calculated scores validate the efficiency, especially through the combination of metrics to compensate individual limitations.

Altogether, the developed extension of the evaluation framework provides RAI researchers and developers with a common solution for the assessment of the responsibility of a clustering tasks. It can be used for many purposes, such as verifying the enhancement of a model after its manipulation or comparing multiple models. As such, it is an important contribution to the further development and research of Responsible AI.

Contents

List of Figures

List of Tables

Abbreviations

AI Artificial Intelligence.

CART Classification and Regression Trees.

DBC Density-Based Clustering.

DBI Davies-Bouldin-Index.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

DPL Differential Privacy Loss.

DT Decision Tree.

EM Expectation-Maximization Algorithm.

GMM Gaussian Mixture Models.

HC Hierarchical Clustering.

IC Individual Consistency.

MBC Model-Based Clustering.

NF Natural Fairness.

PBC Partition-Based Clustering.

RAI Responsible AI.

RD Representation Disparity.

SA Sensitivity Analysis.

SC Silhouette Coefficient.

SLC Single Linkage Clustering.

1 Introduction

Evaluation of Responsible AI for Clustering

In the field of Artificial Intelligence (AI), there has recently been increased interest in a special variation: Responsible AI (RAI). This kind of AI is still new and many topics are yet to be explored. Known research concludes that RAI has many different characteristics, e.g. [14, p. 8] defines it as "Human-centered, Trustworthy, Ethical, Explainable, Privacy(-preserving) and Secure AI".

To bring these characteristics together, [16] presents a framework capable of evaluating the level of responsibility of an AI model. It focuses on classification tasks only, though.

In this work, the framework is extended to supporting clustering algorithms. Clustering is an important facet of AI. Its aim is "to organize data into groups or clusters based on the inherent patterns and similarities within the data" as explained in [34, p. 1].

Since RAI has a wide spectrum, this work seeks to combine different metrics to evaluate a clustering model in the four important areas fairness, privacy, security, and explainability. Fairness is further composed of performance, group fairness and individual fairness. Combined, these aspects give a good impression of how responsible a model is.

Using this method shall enable developers to assess their clustering models and quantify the level of responsibility. In this regard, the method will also make it possible to see the improvements of a model after changes were made, which is especially helpful during the development process.

During the implementation of these metrics, a special focus is on comparability. All metrics are normalized so that the aspects of different models can easily be compared. Furthermore, the duration of the evaluation will be considered. If a metric with high complexity takes long to calculate a score, heuristics are added to accelerate this process. Thus, quick feedback after changes during the development of a model is possible.

Research Questions

This thesis aims to answer the following research questions:

RQ1: Which metrics are relevant to evaluate the aspects of RAI on clustering models?

To answer this question, an overview of the available metrics and ideas for metrics for clustering algorithms is given and briefly explained. Then, their compatibility with different clustering algorithms is checked, and the most fitting metrics are selected.

RQ2: To what extent is it possible to compare different models based on the evaluations of the metrics?

To ensure comparability, the metrics are normalized. The aim is to find a way of normalization that results in a score independent of the model's underlying data. Later, the metrics are used on different models.

RQ3: Which different kinds of clustering algorithms can be evaluated with these metrics?

This question is answered by using an experiment. Different clustering algorithms are applied to different datasets to build models. These models are then evaluated using the implemented metrics.

RQ4: Are these metrics able to point out highly responsible models?

A test model is created with specially designed data to form a clear clustering. This question will be answered by evaluating this model using the metrics.

Outline

The following work is organized in five parts. Chapter 2 begins with an introduction to the metrics, including their compatibility with different clustering algorithms. Several metrics are selected, and a weighting is created. Using the weighting, the metrics are combined to calculate the aspects of RAI. The different metrics are then presented in detail, including their specific calculation, the normalization and implementation.

In chapter 3, five different test models are presented. Four of them focus on different types of clustering algorithms, one focuses on special data.

These models are then evaluated, and the results are presented in chapter 4. This includes the different metrics and the combined RAI aspects.

Chapter 5 summarizes the findings and answers the research questions.

Chapter 6 provides an overview of problems that remain unsolved, as well as suggestions for future research.

2 Metrics

2.1 Overview

2.1.1 Targets

Generally, there are rather few known metrics for clustering algorithms. This is due to missing labels. Most metrics for other algorithms use labels and thus cannot be used on clusterings.

Below, in section 2.1.2, the existing metrics are presented. They are supplemented by ideas that are not yet specified. These approaches are available in a slightly larger quantity.

In section 2.1.3, it is analysed which of these metrics can be used for which specific clustering algorithm. Most are usable for many algorithms but not all.

Finally, in section 2.1.4, the metrics used in this work are presented. A weighting is also introduced to specify which metrics represent which RAI aspects. The selected metrics will be concretized in chapter 2.

2.1.2 Metrics for Different Categories

Performance

Evaluating the performance of a model without labels is a difficult task. There are no explicit metrics to accomplish it. However, several security metrics can help to approximate it. These are the Davies-Bouldin-Index (DBI), the Dunn-Index and the Silhouette Coefficient (SC), which are presented later in this chapter. All of them show compact, well-defined and well-separated clusters. These values are essential for security, but they also show a good performance.

One specific metric does exist though. Inertia is a common metric as shown in [32] that measures the distance of all data points to their cluster centers. It is usually mentioned in the context of k-means as it suits this algorithm perfectly.

Fairness

Group Fairness The fairness metrics can be divided into two categories: individual fairness and group fairness. The group fairness metrics analyze the similar treatment of different groups. This aims to protect whole groups from being treated worse than other groups.

Demographic Parity is described in [26] as a method to create fair classification models. A metric could be derived that compares whether certain attributes are equally distributed between clusters.

A very similar metric could be based on Disparate Impact. As mentioned in [2, p. 5], "In the context of clustering, fairness is often intended in terms of preventing disparate impact over categories of data points." This could be quantified by comparing the proportions of certain attributes between clusters.

The difference between the two metrics is that at Disparate Impact, the focus is on the ratio between individual clusters while Demographic Parity focuses on the ratio between all clusters at the same time. However, both are not yet specific metrics and need to be clearly defined.

Finally, Representation Disparity (RD) as proposed in [33] also is a similar idea. Here, the attribute occurrences in the individual clusters are compared to the whole dataset instead of other clusters.

Individual Fairness The individual fairness metrics are used to evaluate whether individual data points similar to each other, are also assigned to the same cluster.

Fairness Through Awareness is a metric often used in classification tasks as suggested in [12]. It can also be used for clustering, though. In this case, it is checked whether the similar data points are in the same cluster. A similar metric called Individual Consistency (IC) can be derived. It uses the same principle in a basic form which makes it especially suitable for clustering.

Another unnamed approach is introduced in [23], here referenced as Natural Fairness (NF). It focuses on assuring that each data point is closer to the other data points in its own cluster than those in other clusters.

Privacy

The most common privacy metric for clustering is Differential Privacy Loss (DPL). It is the loss function of the Differential Privacy method introduced in [19] that can be adapted and used as a metric. The idea is to add noise to the cluster centers and then check if the data points are still assigned to the same clusters.

An adaptation is the Sensitivity Analysis (SA), which measures the changes in the clustering after a new training with a data point missing. It can be derived from [18].

Some other metrics are k-Anonymity, l-Diversity and t-Closeness as in [10]. These can be used to ensure privacy in a dataset, but can be adapted to clusterings. For example, for k-Anonymity a simple check may be introduced to see if each cluster has multiple data points in it to reduce the risk of re-identification.

Security

For security, there are several metrics available. The Silhouette Coefficient (SC) as introduced in [30] calculates the relation between the distances a data point has to its own cluster's other data points and its distances to the data points of its closest other cluster.

The Dunn-Index introduced in [11] works in a similar way, but it compares the maximum distance within a cluster to the minimum distance between two clusters.

Another metric is the Davies-Bouldin-Index (DBI) introduced in [9]. It also works in a similar way. Here, the compared distances are the average distance of the data points of a cluster to its cluster center and the distance between two cluster centers.

Explainability

Metrics to evaluate a clustering's explainability are scarce. The simplest way is to use the methods before mentioned again, such as the SC, the Dunn-Index or the DBI. These indicate clearly separated clusters. Having clearly separated clusters, it is easy to explain the assignment of a data point to a certain cluster, i.e. the explainability is good.

A more obvious method are Decision Tree (DT) based explanations. If a DT is trained based on the data of a clustering with the assigned cluster as the label for each data point, then the complexity of this DT can be used to show if it is easy or hard to explain the clustering.

2.1.3 Compatibility

The mentioned metrics mostly work for clusterings in general. Some have restrictions though. Table 2.1 shows which metrics work for which of the following algorithms: Partition-Based Clustering (PBC), Hierarchical Clustering (HC), Density-Based Clustering (DBC), and Model-Based Clustering (MBC).

Table 2.1: Compatibility of Metrics and Clustering Types

Metric	PBC	HC	DBC	MBC
Inertia	✓	×	×	×
Demographic Parity	✓	✓	✓	✓
Disparate Impact	✓	✓	✓	✓
Representation Disparity	✓	✓	✓	✓
Fairness Through Awareness	✓	✓	✓	✓
Individual Consistency	✓	✓	✓	✓
Natural Fairness	✓	✓	✓	✓
Differential Privacy Loss	✓	×	×	×
Sensitivity Analysis	✓	✓	✓	✓
k-Anonymity	✓	✓	✓	✓
l-Diversity	✓	✓	✓	✓
t-Closeness	✓	✓	✓	✓
Silhouette Coefficient	✓	✓	✓	✓
Dunn-Index	✓	✓	✓	✓
Davies-Bouldin-Index	✓	✓	✓	✓
Explainability by Decision Tree	✓	✓	✓	✓

It can be observed that Inertia and DPL are only compatible with PBC. Inertia does work for other algorithms, but it is not meaningful. It assumes that the data points are closest to their cluster center. This is just given for the PBC methods though.

The DPL is even more extreme. It requires the algorithm to assign points to their closest cluster center. Only in this way, the change can be measured that is brought by the added noise to the cluster centers.

For some of the other metrics, it could be argued that they need to be adapted to suit some clustering methods or that their scores are less meaningful in some places. However, it is possible to use them all on the methods listed above and their score does give an indication to the evaluation. Thus, they are always useful to a certain degree.

2.1.4 Chosen Metrics and Weighting

Only a limited amount of metrics can be addressed in this work. They need to be able to evaluate all aspects of RAI for the different kinds of clustering types.

Table 2.2 shows which metrics were selected and to what extent they are combined to evaluate each aspect. The given numbers are factors. For example, to assess the performance, the sum of twice the Inertia score, once the SC and once the DBI is divided by four (2 + 1 + 1).

Table 2.2: Weighting of Selected Metrics for Different Aspects

Metric	Performance	Group Fairness	Individual Fairness	Privacy	Security	Explainability
Inertia	2x					
Representation Disparity		1x				
Individual Consistency			1x			
Natural Fairness			1x			
Differential Privacy Loss				2x	1x	
Sensitivity Analysis				1x	1x	
Silhouette Coefficient	1x				2x	1x
Davies-Bouldin-Index	1x				2x	1x
Explainability by Decision Tree						4x

In the table, it can also be seen that all aspects are covered by the selected metrics. Even if Inertia and DPL are not available for most algorithms, the aspects are still fully covered by the remaining metrics. The reason why these two metrics are still addressed in this work is that they are especially well-suited to evaluate performance and privacy respectively.

The additional metric for privacy evaluation is SA. Since it is similar to DPL, it is a fitting alternative. The other metrics k-Anonymity, l-Diversity and t-Closeness are more useful in order to create a model with higher privacy protection instead of assessing it. Therefore they are not further considered here.

DPL and SA are also relevant for the security aspect. Both of them evaluate the change of the model after a disruption. This shows their robustness against disturbances through attacks.

Apart from that, the metrics SC and DBI are both selected for the security aspect. The Dunn-Index works in a similar manner to both of them. Thus, it is redundant to a certain degree to use all three algorithms. Hence, the Dunn-Index was not implemented here. The two algorithms used focus more on security and are thus weighted higher than DPL and SA.

They are also relevant to evaluate the performance. However, they weigh less than Inertia, which has its focus on performance. If Inertia is not usable, they suffice alone.

As mentioned above, they also give an impression of the explainability. However, the DT approach is a lot more appropriate in this place and therefore implemented here and weighted twice as much as the other two scores together.

For the group fairness aspect, the RD was selected. It is similar to Demographic Parity and Disparate Impact, but transforming it into a metric is slightly more straightforward, based on its concrete definition.

To assess individual fairness, the metrics IC and NF were both selected. They are both well-suited for clustering. Fairness Through Awareness is redundant because it is very similar to IC. Both selected metrics are relevant and are weighted equally.

Even though most metrics are used for multiple aspects, they are still all categorized into just one in the following chapters. Chosen is the aspect in which the according metric weighs the most.

Eventually, after these aspects are calculated based on the metrics, performance, group fairness and individual fairness still need to be combined to form the aspect fairness, as in the base work [16]. To do so, their mean value is calculated.

Then, as a last step, the mean of these four aspects can be calculated as the complete RAI score.

2.2 Performance

2.2.1 Inertia

Explanation

The Inertia metric, as described in [32] among others, can be used to measure the performance of a clustering model.

It is calculated as the sum of all squared Euclidean distances of the data points in a cluster to their corresponding cluster centers. This makes the score highly dependend on the underlying dataset. Thus, when not normalized, it is mostly helpful for comparing clusters formed on the same dataset.

Calculation

The Inertia score I is calculated as shown in equation 2.1. K represents the amount of clusters, S the data points, S_k data points in cluster k and $d(y_i, c_k)^2$ the squared Euclidean distance where y_i is the ith data point in S_k and c_k is the cluster center of cluster k.

$$I(K) = \sum_{k=1}^{K} \sum_{i \in S_k} d(y_i, c_k)^2 \tag{2.1}$$

Normalization

Standard Approach Due to the way the Inertia score is calculated, it can grow endlessly with each additional data point. Thus, a specific score value is not very meaningful because is unclear how much higher it could be.

Consequently, the score has to be normalized. This could be done by calculating the minimum and maximum possible value for the present data points and then using the equation 2.2.

$$normalized(x) = \frac{x - min(x)}{max(x) - min(x)} \tag{2.2}$$

When using Inertia, the minimum is always 0. This score can be reached when there is a separate cluster for each data point. In this situation, the difference between each point and its corresponding cluster center is always 0, so the sum is also 0. Knowing this, the equation can be simplified to equation 2.3.

$$normalized(x) = \frac{x}{max(x)} \tag{2.3}$$

Likewise, the maximum can be calculated by assuming there is just one cluster for all data points. This value then needs to be calculated for the current set of data points and cannot be generally specified.

Alternative Approach It can be observed that the Inertia score grows with each data point. This should not be the case for a normalized score. Therefore, it seems logical to divide the score by the amount of data points. This way, it does not grow with their number.

Simply dividing the Inertia score by the amount of data points does not yield a normalized score though. The distance between each data point and its corresponding cluster center still depends on the nature of the data.

A solution here could be to normalize this distance instead of the total score as in the Standard Approach. In order to apply equation 2.2 to each single distance before the aggregation, the maximum and minimum have to be determined again.

Just as before, the minimum possible distance between a data point and its cluster center is 0. This can be achieved when they are in the same position, e.g. when there is just this one data point in a cluster. Consequently, equation 2.3 can be used again.

To find the maximum, the same approach as before cannot be used again. It is possible for a data point to have a cluster center which is further away than it would be if all points were in one cluster. An example would be a situation with two places with many points and one point far away but included in one of those clusters as can be seen in figure 2.1.

Since this way does not work, the maximum distance between two data points can be considered. The cluster centers are the center points between data points. Therefore, they cannot be further outside than the furthest data points. Consequently, it is not

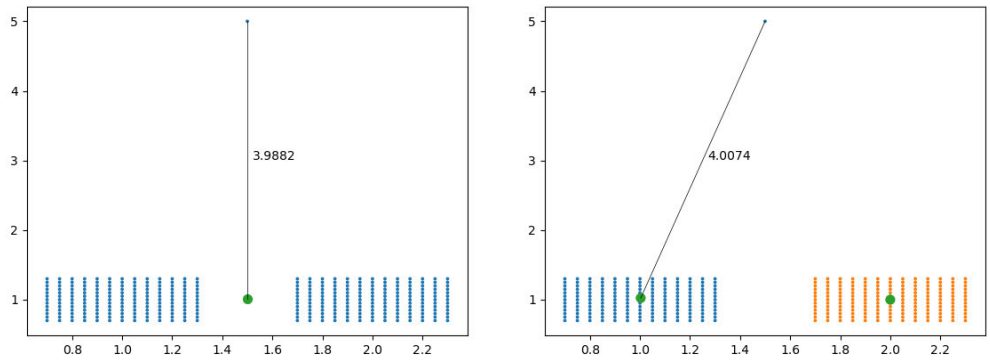


Figure 2.1: Proof that a point can be further away from its k-means cluster center than the maximal distance in one huge cluster

possible for a point to have a greater distance to its cluster center than the distance between the two data points with the longest distance.

The question now is whether this distance can also be reached or if the maximum is smaller. An easy example would be a collection of equal data points at one point and one additional data point somewhere else. The maximum distance between two data points here is the distance between any of the equal points and the other point. If there is just one cluster in this example, its center would be close to the amassing of equal data points, but slightly shifted in the direction of the outlier point. Thus the maximum difference between data point and center is a little smaller than between data point and data point. However, if there are more and more of those equal data points, then the cluster center will get closer and closer to the amassing. Thus, the distances also converge with unlimited additional data points.

As this approach can be used, the score $I_{normalized}$ would be calculated as stated in equation 2.5 with the maximum distance $D_{\max}$ between a data point and its cluster center as in equation 2.4.

$$D_{\max} = \max_{a,b \in S} d(a,b) \tag{2.4}$$

$$I_{normalized}(K) = \frac{\sum_{k=1}^{K} \sum_{i \in S_k} \frac{d(y_i, c_k)^2}{D_{\max}}}{|S|} \tag{2.5}$$

Discussion

During the implementation of the Alternative Approach, a big flaw became obvious. Finding the longest distance between two data points takes a lot of time. As seen in [27] this problem has a high complexity of $O(n^2)$ when using brute force or $O(n \log n)$ with some other algorithms. This makes it impractical to use.

Another problem is the fact that the best score 0 can always be reached by simply using one cluster per data point. In other words, this score can be improved by increasing the amount of clusters. Thus, it cannot be used to evaluate the used amount of clusters but only the performance of the distribution of cluster centers when using exactly this amount of clusters.

Apart from that, Inertia is a straightforward approach that is easy to comprehend when used on Partition-Based Clusterings with convex clusters. Using the Standard Approach, the algorithm is fast, which makes is practical to use for quick feedback when developing a model.

Implementation

Listing 2.1 shows the implementation of the presented Inertia metric.

```
    kmeans = KMeans(n_clusters=1, **kmeans_kwargs)
    kmeans.fit(data)
    max_inertia = kmeans.inertia_

    labels = clustering_model.predict(data, encoded=True)
    cluster_centers = data.groupby(labels).mean()
    inertia = 0.0
    for i, row in data.iterrows():
        row_label = labels[i]
        cluster_center = cluster_centers.loc[row_label]
        distance = sqeuclidean(row, cluster_center)
        inertia += distance

    normalized_inertia = 1.0 - (inertia / max_inertia)
```

Listing 2.1: Calculating Inertia Score

First, a new k-means model is initialized with the same data points as the model being evaluated. The new model is then fitted with just one cluster. This way, the maximum possible Inertia value is determined, as required by the Standard Approach. The used k-means algorithm is from the library scikit-learn [28] and already offers the Inertia score as an attribute.

Next, the Inertia is calculated for the test model. Since the test model may not be a k-means model, the Inertia score is not read from the model but manually calculated as explained above. Then, the normalized score can be calculated by a simple division. Finally, the score is subtracted from 1 to ensure that a higher score represents a better model.

2.3 Fairness

2.3.1 Representation Disparity

Explanation

The Representation Disparity (RD) metric as described in [33] is a metric to evaluate the group fairness.

The score is determined by averaging the disparities for all clusters for all manifestations of all attributes. A disparity represents the difference between how often a value relatively appears in a cluster and in the entirety of data points.

Calculation

Calculating the score R involves aggregating the disparities $d(a, v, k)$ of all values V of the attributes A over all clusters K as shown in equation 2.6. Equation 2.7 shows how the disparity $d(a, v, k)$ is determined, based on the proportion of the attribute a value v in the entirety $pe(a, v)$ as in equation 2.8 and its proportion in the cluster k as $pc(a, v, k)$ as in equation 2.9. S represents the data points, S_k the data points in cluster k and y_a the value of attribute a of data point y.

$$R = \frac{\sum_{a \in A} \frac{\sum_{v \in V_a} \frac{\sum_{k \in K} d(a,v,k)}{|K|}}{|V_a|}}{|A|} \tag{2.6}$$

$$d(a, v, k) = |\frac{pc(a, v, k)}{pe(a, v)} - 1| \tag{2.7}$$

$$pe(a, v) = \frac{\sum_{y \in S} 1_{\{y_a = v\}}}{|S|} \tag{2.8}$$

$$pc(a, v, k) = \frac{\sum_{y \in S_k} 1_{\{y_a = v\}}}{|S_k|} \tag{2.9}$$

The aggregation is achieved by getting the mean. This is done three times to make sure that all attributes have the same weight in the final score. If just one aggregation were used, the attributes with more manifestations would weigh more. The aggregation over the clusters could be joined with the one over the values since there is always the same number of clusters. Leaving it there keeps the equation more readable though and reduces the required memory during computation.

Normalization

This metric usually results in a score between 0 and 1. However, it is possible to reach a value greater than 1 if the data is distributed in extremely unfair ways. It is not necessary to normalize, since the score 1 already describes a poor fairness. The easiest way is to round down anything above 1 to 1. This way, the score still is the worst possible value.

After this, the score still needs to be inverted to ensure that 1 is the best score and 0 the worst. This can be done as shown in equation 2.10.

$$R_{normalized} = |\min(R, 1) - 1| \tag{2.10}$$

Discussion

One problem is the afore mentioned difficulty during the normalization. Though the shown solution minimizes this problem, the comparability of very specific data layouts is still limited. Concretely, this means that starting at a certain threshold of poor group fairness, the metric always returns the same score.

Another problem is the computation time. The computation of the metric requires the consideration of all possible values for all attributes in all clusters, which add up to a high amount of single calculations, depending on the data. As the implementation and execution has shown, this results in a fairly long time to calculate the score.

The first problem is rather small. Only very unfair scenarios are rounded to the value of 1 (before inverting). The high complexity is acceptable for the final evaluation of a model, but makes it hard to improve a huge model based on regular quick feedback. Apart from these points, the metric gives a good summary of the group fairness of a model.

Implementation

Implementing the RD metric requires several nested loops to iterate all values of all attributes for all clusters. During these loops, the individual disparities are aggregated. Listing 2.2 shows these loops.

```
    attribute_disparities = []
    for attribute in data.columns:
        value_counts = data[attribute].value_counts()
        value_disparities = []
        for value in data[attribute].unique():
            entirety_portion = value_counts.get(value) / value_counts
                .sum()
            disparities = []
            for cluster in data_per_cluster.keys():
                cluster_portion = sum([1 if datapoint[attribute] ==
                    value else 0 for datapoint in data_per_cluster[
                    cluster]]) / len(data_per_cluster[cluster])
                disparities.append(abs(cluster_portion /
                    entirety_portion - 1.0))
            value_disparities.append(np.mean(disparities))
        attribute_disparities.append(np.mean(value_disparities))
    disparity = np.mean(attribute_disparities)
```

Listing 2.2: Calculating RD Score

The normalization is simple for this metric, as shown in listing 2.3.

```
    if disparity > 1:
        disparity = 1
    disparity = abs(disparity - 1.0)
```

Listing 2.3: Normalizing RD Score

2.3.2 Individual Consistency

Explanation

The Individual Consistency (IC) metric evaluates the individual fairness as explained in [15, 21, 3]. The evaluation is the proportion of similar data points that are assigned to the same cluster.

Calculation

The score is calculated in three steps. The first step is the most complex. The similarity of the data points is calculated. To do so, the distances between all pairs of data points need to be determined. Those pairs with a smaller distance are considered similar. The distance used here is the Euclidean distance.

This threshold distance cannot be set to an absolute value, as different kinds of data can have very high differences in the distances between their data points. To find a fitting value, different approaches can be considered, but most are domain-specific and require a deeper understanding of the underlying domain. Since the tool implemented here should be universally applicable, this is hard to accomplish.

Thus, the best universal way is to take a certain percentage of pairs. A fixed number can be used, for example 3%. This way data pairs can be ordered by their distance and the 3% with the lowest distance are then considered similar and used for the following two steps.

During the second step, the clusters of the data pairs are compared. It is counted how many similar data pairs are assigned to the same cluster and then this number is relativized in the last step.

$$I = \frac{\sum_{p \in P_{similar}} 1_{(C(p_1)=C(p_2))}}{|P_{similar}|} \tag{2.11}$$

Equation 2.11 shows the exact formula. $P_{similar}$ is the set of similar data pairs. p is a single pair of the two similar data points p_1 and p_2. The function $C(x)$ returns the cluster assigned by the model to the passed data point.

Normalization

Since the score is the relative amount of similar pairs that are in the same cluster, it is by definition a number between 0 and 1 and thus does not require normalization.

Discussion

As mentioned above, the complexity is rather high again. Finding the distances between all data pairs is an $O(n^2)$ complex problem. Instead of exactly calculating everything, the score could be approximated by randomly selecting several data pairs and running the algorithm just on these. This way, a close approximation can quickly be reached for testing the individual fairness during its development. Should a model be finally evaluated precisely, the whole data can be used again.

Another problem with the algorithm is the dependency on the number of clusters. If there are many data points and a few clusters, the metric's score is comparatively high, even if the data points were randomly assigned to clusters. In an extreme case, with just one cluster, the score is always a perfect 1.

In case there are many clusters, even close data points might be correctly assigned to different clusters. Thus, it is possible to maliciously manipulate the score. However, it generally gives a good impression of the individual fairness when considered with caution. It is also easy to comprehend.

Implementation

Listing 2.4 shows how the 3% of all pairs of data points with the shortest distance are selected. These are used in listing 2.5 to calculate the IC score.

```
    distances_with_pairs = []
    for (point1, point2) in all_pairs
        dist = np.linalg.norm(point1 - point2)
        distances_with_pairs.append(((point1, point2), dist))
    sorted_distances_with_pairs = sorted(distances_with_pairs, key=
       lambda x: x[1])
    similar_pairs = sorted_distances_with_pairs[:max(1, int(len(
       sorted_distances_with_pairs) * 0.03))]
```

Listing 2.4: Finding similar pairs for IC Score

```
    same_cluster = 0
    for (pair, _) in similar_pairs:
        if self.model.predict([pair[0]], encoded=True) == self.model.
           predict([pair[1]], encoded=True):
            same_cluster += 1
    individual_consistency = same_cluster / len(similar_pairs)
```

Listing 2.5: Calculating for IC Score

Listing 2.6 further proposes a simple preselection of considered data points for bigger datasets. Here, a fifth of all unique pairs is selected, but no more than 100,000 pairs in total.

```
    selected_indices = set()
    max_indices = min(100000, (len(data) * (len(data) - 1)) // 10)
    while len(selected_indices) < max_indices:
        idx1 = random.randint(0, len(data) - 1)
        idx2 = random.randint(0, len(data) - 1)
        if idx1 == idx2:
            continue
        indices = tuple(sorted((idx1, idx2)))
        selected_indices.add(indices)
```

Listing 2.6: Heuristic for Preselection for IC Score

2.3.3 Natural Fairness

Explanation

The Natural Fairness (NF) metric was introduced in [23]. It shows the proportion of data points that fulfill the fairness condition. This condition is that the average distance to the points in the same cluster is smaller than the average distance to all other points.

Calculation

To calculate the score, the fairness condition mentioned above needs to be checked for every data point. To do so, two scores are required for each data point $p \in P$. The first one is the average distance to all other data points in the same cluster $d_{in}(p)$ as shown in equation 2.12. The term $C(p)$ references the cluster assigned to p and function $d(p, x)$ calculates the Euclidean distance between p and x.

$$d_{in}(p) = \frac{1}{|\{x \in P \setminus \{p\} \mid C(x) = C(p)\}|} \cdot \sum_{x \in P \setminus \{p\} | C(x) = C(p)} d(p, x) \tag{2.12}$$

The second score is the average distance to all data points in other clusters $d_{out}(p)$. This is presented in equation 2.13.

$$d_{out}(p) = \frac{1}{|\{x \in P \mid C(x) \neq C(p)\}|} \cdot \sum_{x \in P | C(x) \neq C(p)} d(p, x) \tag{2.13}$$

Next, these two scores need to be compared. The fairness condition $f(p)$ is fulfilled if $d_{in}(p)$ is smaller than $d_{out}(p)$. This can be expressed as an indicator function as in equation 2.14.

$$f(p) = 1_{\{d_{in}(p) < d_{out}(p)\}} \tag{2.14}$$

Finally, these scores need to be aggregated to reach the final score N. Equation 2.15 shows this.

$$N = \frac{1}{|P|} \cdot \sum_{p \in P} f(p) \tag{2.15}$$

Normalization

The score is a percentage of the data points and as such already a value between 0 and 1, with 1 meaning that every data point fulfills the fairness condition and thus being the highest score. There is no further normalization required.

Discussion

This algorithm has the complexity $O(n^2)$, which makes its execution slow on big datasets. This common problem might be reduced by finding a small selection of data points to consider during the computation.

Apart from this disadvantage, the NF score gives a very good measurement of the individual fairness. For each data point, it is separately checked whether it is closer to its own cluster than to the rest. If not, this will directly decrease the score. This way, it can directly be seen if there are data points which are not fairly assigned from their respective perspective.

Furthermore, this metric does not have huge impairments based on the used clustering algorithm. The form of the clusters does not matter because it is clearly checked for a point if it best fits its own cluster.

Implementation

Listing 2.7 shows the implementation for this algorithm. First, the distance matrix is calculated. This saves time because all distances are needed anyway. Then, the score is calculated for each data point and finally aggregated.

```
1     distances = pdist(data.values.astype(float), metric='euclidean')
2     distance_matrix = squareform(distances)
3     labels = self.model.predict(data, encoded=True)
4
5     fair_count = 0
6     for i in range(len(data)):
```

```
        same_cluster_distances = []
        other_cluster_distances = []
        for j in range(len(data)):
            if i == j:
                continue
            if labels[i] == labels[j]:
                same_cluster_distances.append(distance_matrix[i][j])
            else:
                other_cluster_distances.append(distance_matrix[i][j])
        din = np.mean(same_cluster_distances)
        dout = np.mean(other_cluster_distances)
        if din < dout:
            fair_count += 1

    natural_fairness = fair_count / len(data)
```

Listing 2.7: Implementation of NF Score

2.4 Privacy

2.4.1 Differential Privacy Loss

Explanation

The Differential Privacy Loss (DPL) is primarily used to understand the loss brought by the use of the Differential Privacy method to prevent privacy leaks in a clustering. In this method, noise is added to the cluster centers. [19] shows this while also focusing on the loss functions.

An example for a possible loss function is widely used as a metric to evaluate the privacy of the clustering. The function used here is the portion of data points that are in the same cluster before and after adding noise to the cluster centers.

Originally, this loss function shows how much the noise falsifies the functionality of the clustering, i.e. the clustering's performance loss when protecting the privacy. Here, it serves a different purpose, namely the evaluation how well the privacy is protected in the clustering.

That this purpose can also be achieved by this metric, can be seen as a higher score shows that the results of the clustering are stable even with added noise. This shows that slightly adapting the underlying data such as removing individual data points does not influence the result of the model much. Thus, the individual data points cannot easily be inferred by knowing the cluster centers.

Calculation

Since the score is calculated by comparing the original model with a manipulated model, the first step is to create this new model. This is achieved by adding noise to the cluster centers. There are different ways to randomize this noise. A commonly used one is the Laplace distribution [1].

To scale the noise, the parameter ϵ is required. It is hard to set it to a fixed value because the impact highly depends on the underlying data. Instead, it should be relative to the attributes' value range. Attributes with large values should have more noise. For

example, adding 0.5 noise to an attribute with values around 0.01 would be far too much while the same noise added to an attribute with values around 1,000 seems insignificant.

In conclusion, equation 2.16 shows the calculation of the scalar S_a for the Laplace distribution for one attribute a for all cluster centers C. c_a is the value for attribute a of cluster center c. In this formula, the mean of the attributes' values is calculated from the absolute values because it is searched for the average size of the attribute's values. This way, positive and negative values do not cancel each other out. Finally, the parameter ϵ is integrated to allow adjusting the metric's sensitivity. It can now be set to a fixed value without disregarding the attributes' differences.

$$S_a = \frac{\frac{\sum_{c \in C} |c_a|}{|C|}}{\epsilon} \tag{2.16}$$

Based on these scalars, the attributes $a_{c_{new}}$ of the new model's cluster centers $c_{new} \in C$ can be randomized as shown in equation 2.17. $a_{c_{old}}$ is the value of attribute a of cluster c of the old model.

$$a_{c_{new}} = a_{c_{old}} + \text{Laplace}(0, S_a) \tag{2.17}$$

Finally, the loss L is calculated by comparing the two models results $M_{old}(p)$ and $M_{new}(p)$ for all data points $p \in P$ as shown in equation 2.18.

$$L = \frac{\sum_{p \in P} 1_{M_{old}(p) = M_{new}(p)}}{|P|} \tag{2.18}$$

Normalization

This score is by definition a number between 1 and 0 and thus does not need to be normalized.

Discussion

As mentioned before, the amount of noise added depends highly on the nature of the attributes' values. Even though a general solution was found, this metric would deliver more meaningful information if the domain were manually examined for each attribute. This is a good approximation though.

Another minor downside is the random part. This way, the computation is not deterministic and results in slightly different results every time. To ensure that no unlucky results are calculated due to very specific randomized values, the score should be calculated several times. This way, the result range with the most occurrences can be considered the most relevant.

Generally though, the added noise is a powerful tool to reduce the identifiability of individual data points. This allows this metric to give a solid impression of the privacy protection level of a clustering.

Implementation

Listing 2.8 shows how the model is copied and noise is added to the new model. The comparison of the two models is shown in listing 2.9.

```
1    pd_model = copy.deepcopy(self.model.model)
2    ccs = self.model.model.cluster_centers_
3    epsilon = 10
4    pd_model.cluster_centers_ = ccs + np.random.laplace(0, np.abs(ccs
        ).mean(axis=0) / epsilon, ccs.shape)
```

Listing 2.8: Adding noise for DPL Score

```
1    original_prediction = self.model.predict(data, encoded=True)
2    noise_prediction = pd_model.predict(data)
3
4    same_prediction_count = sum([1 if o == n else 0 for o, n in zip(
        original_prediction, noise_prediction)])
5
6    differential_privacy_loss = same_prediction_count / len(self.data
        )
```

Listing 2.9: Comparing models for DPL Score

2.4.2 Sensitivity Analysis

Explanation

The Sensitivity Analysis (SA) can be used to evaluate the privacy protection offered by a clustering. There are many ways to measure the sensitivity of a model. Although not explicitly mentioned, one way can be derived from [18]. That is to compare how the clustering is changed when a specific data point is omitted during its creation.

If the overall clustering changes drastically based on the existence of single data points, then the sensitivity is high which means the privacy protection is not good. Instead, if its existence does not influence the clustering considerably, then this also indicates that the base data cannot easily be inferred from the model. Thus, the privacy is well-protected.

Calculation

Step Overview Since there is no research yet on how to specifically implement this metric, some questions need to be considered. First, the points to be omitted need to be chosen. Then, the model is refitted for each new dataset. This new model needs to be compared with the old model. Here, a specific formula is required to assign a score to the difference. Finally, these scores need to be aggregated.

Choosing Points to be Omitted The simplest solution to the first question is to omit every data point. This way, the most complete evaluation can be reached. For a final evaluation of a model, this is useful. However, for feedback during the creation process, this would take a long time. Here, it is more useful to get a quick approximation by considering only a small selection of the data points.

This selection $S \subseteq P$ should on the one hand be representative for the entire dataset P. On the other hand, it must especially be ensured that small clusters are not neglected. To fulfill the second condition, a certain number a of data points should be chosen from each cluster $c \in C$. Afterwards, to fulfill the first condition, a certain number b of random data points of the overall dataset, that were not chosen before, will be added. This way, a total of l data points will be chosen.

Having $l = a \cdot |C| + b$, the next problem is to find the right proportion. To get a proper summary of the entire data structure, and to prevent very small clusters from being overrepresented, it is suggested that $b = 2 \cdot a \cdot |C|$.

Setting a to a constant value is difficult because the structure of the clustering is not known. There could be very few or very many data points per cluster. Thus, they should be relative to the number of data points $|P|$ and the number of clusters $|C|$. A reasonable solution is to set a fixed percentage f such that $l = f * |P|$. Here, it is set to $f = 0.05$.

A third of the data points in S is chosen directly from the clusters. Thus, it must be ensured that $l >= 3 \cdot |C|$. This way, it is guaranteed that at least one data point can be chosen from each cluster. If this number is larger than the total number of data points, then this approach is not useful and instead, all data points can be omitted. This case is not mentioned in the below formulas but implemented in the productive code.

Having this minimum, it can be helpful to introduce a maximum m as well for very big datasets. Here, a constant value is sufficient. In this work, it is $m = 10000$. Since the maximum is just to reduce computation time, it is of lower importance than the minimum.

Combining these ideas results in equation 2.19 for the total number of data points to omit and equation 2.20 for the data points selected per cluster.

$$l = \max(3 \cdot |C|, \min(10000, 0.05 * |P|)) \tag{2.19}$$

$$a = \frac{l}{3 \cdot |C|} \tag{2.20}$$

One more problem would be, if in any cluster, there are fewer than a data points. In this situation, all data points can be added in the first step. To ensure the correct balance with the points randomly chosen from the full dataset, b should be lowered by 2 for each missing data point in the set. This results in equation 2.21 with $|c|$ standing for the number of data points assigned to cluster c.

$$b = \sum_{c \in C} \min(|c|, a) \cdot 2 \tag{2.21}$$

This way, the anticipated $|S| = l$ might no longer be true. The exact number of selected data points is shown in equation 2.22.

$$|S| = \sum_{c \in C} \min(|c|, a) \cdot 3 \tag{2.22}$$

Comparing Models In the next step, for each data point $s \in S$ a score $e(s)$ is calculated. For that, the model is first refitted on the original dataset without the current point $O_s = P \setminus \{s\}$. Then, every of these data points $o \in O_s$ has the cluster assigned in the original model $C_P(o)$ and the cluster assigned in the modified model $C_{O_s}(o)$.

These assigned clusters are essential for the score $e(s)$, but it is not enough to check for every data point if they are equal or not. Figure 2.2 shows a simple example with five data points. Here, after omitting one point, all other data points change their assigned cluster, but are still in the cluster with the same data points as before. This situation would be rated with a bad score if just the change of the cluster is considered. Instead, this example should reach a perfect score.

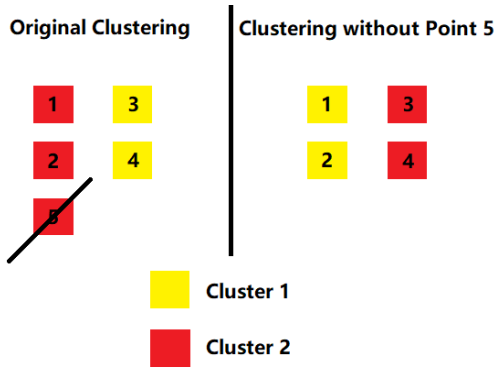


Figure 2.2: Example for clustering with absolute but not relative changes after omitting one point

To reach this, another approach is needed in which it is evaluated to what extent the data point is still with the same other data points in the same cluster as before. This new approach must fulfill three conditions: Every data point that was originally in the same cluster, but is not anymore in the new model, must decrease the score; every data point that originally was not in the same cluster, but is in the new model, also must decrease the score; every data point that was originally in the same cluster, and still is in the new model, must increase the score.

This can be achieved by a simple division. The denominator $u_s(o)$ is the total number of unique data points that are in at least one of the same clusters as o in the original and in the modified model. Just the omitted data point s is ignored. This is calculated as shown in equation 2.23.

$$u_s(o) = |\{k \in O_s \mid C_P(k) = C_P(o)\} \cup \{k \in O_s \mid C_{O_s}(k) = C_{O_s}(o)\}| \tag{2.23}$$

The numerator $i_s(o)$ is the number of data points that are as well in the same dataset as o in the old model as well as in the new model. Equation 2.24 shows the exact math.

$$i_s(o) = |\{k \in O_s \mid C_P(k) = C_P(o)\} \cap \{k \in O_s \mid C_{O_s}(k) = C_{O_s}(o)\}| \tag{2.24}$$

Dividing these two results in the score for the specific data point $r_s(o)$ as demonstrated in equation 2.25.

$$r_s(o) = \frac{i_s(o)}{u_s(o)} \tag{2.25}$$

Aggregating Scores Finally, these scores have to be aggregated. This happens over three stages. The first two stages are to find $e(s)$. Although this could be done in one step, it appears advantageous to first aggregate over the individual clusters. This way, smaller clusters, which are more prone to privacy issues, are weighed the same as larger ones. This increases the sensitivity of the algorithm.

The scores are aggregated over the clusters of the original model because these build the structure that will eventually be evaluated. In case that the omitted point s alone builds a whole cluster in this model, this cluster can be ignored. This case is no further considered in the below formulas but implemented in the actual code. Equation 2.26 shows the aggregation for each cluster $c \in C$ for the model omitting s as $g_s(c)$.

$$g_s(c) = \frac{1}{|c|} \cdot \sum_{j \in c} r_s(j) \tag{2.26}$$

Next, these scores can be aggregated for all clusters as presented in equation 2.27.

$$e(s) = \frac{1}{|C|} \cdot \sum_{c \in C} g_s(c) \tag{2.27}$$

The last stage and final step is the aggregation over all omitted data points to reach the final score E. This can be seen in equation 2.28

$$E = \frac{1}{|S|} \cdot \sum_{s \in S} e(s) \tag{2.28}$$

Normalization

Since the score already is between 0 and 1 with 1 being the best value, it does not need to be normalized any further.

Discussion

The SA requires a refitting of the model. In some cases, such as k-means, this might have severer consequences than anticipated if local minima are found, which would possibly result in a completely different new clustering. The determined score would then give a wrong impression. Such a case should be prevented, e.g. by training the model multiple times with different initializations. This would, in turn, increase the duration of the computation.

The duration is already long as it is, because many models need to be trained. Additionally, for each new model, all data points are evaluated. For large datasets, this is a lengthy process.

The preciseness of the score might also be partly dependent on the dataset. If a single point is omitted on a large dataset, the consequences might be less significant than on a smaller dataset. A solution in future work might be that the metric is extended to choose sets of data points. The data points in a set are omitted together. Their number would be adequate for the dataset size.

Nonetheless, this score is relevant. If single missing data points have a bigger impact, this would be a privacy problem that can be detected with this metric.

Implementation

The implementation of the SA can happen in the same steps as explained above. First, the data points that shall be omitted must be found. This can be seen in listing 2.10.

The next step in listing 2.11 is to calculate the score for each of those data points that were found before.

Finally, listing 2.12 shows the last step. The aggregation is split into three parts. The first two are in the same loop as in listing 2.11 and the last one is outside of this loop.

```
    labels_original = self.model.predict(data, encoded=True)
    clusters_original = {}  # key = cluster id; value = list of data
        point indices that are in the cluster
    for index, cluster in enumerate(labels_original):
        if cluster not in clusters_original:
            clusters_original[cluster] = []
        clusters_original[cluster].append(index)

    l = max(3 * len(clusters_original), min(10000, int(0.05 * len(
        self.data))))

    datapoint_indices_to_omit = []
    if l >= len(data):
        datapoint_indices_to_omit = [i for i in range(len(data))]
    else:
        a = l // (3 * len(clusters_original))

        b = sum(min(len(c_datapoints), a) * 2 for c, c_datapoints in
            clusters_original.items())

        # selecting data points to omit from clusters
        for c_datapoints in clusters_original.values():
            datapoint_indices_to_omit.extend(random.sample(
                c_datapoints, a) if len(c_datapoints) > a else
                c_datapoints)

        # selecting data points from whole set
        left_datapoints = [data_point_index for data_point_index in
            range(len(labels_original)) if data_point_index not in
            datapoint_indices_to_omit]
```

```
24          datapoint_indices_to_omit.extend(random.sample(
                left_datapoints, b))
```

Listing 2.10: Finding data points to omit for SA

```
1     estimations_for_missing_datapoints = []
2     for index_of_point_to_be_omitted, s in enumerate(
          datapoint_indices_to_omit):
3         # training new model
4         os_data = data.drop(index=s)  # no reset_index to make sure
              the indices do not include s below
5         os_model = copy.deepcopy(self.model)
6         os_model.model.fit(os_data)
7
8         # creating comparable data structure with attention to the
              indices of both models
9         labels_os = os_model.predict(os_data, encoded=True)
10        clusters_os = {}  # key = cluster id; value = list of data
              point indices that are in the cluster
11        for index, cluster in enumerate(labels_os):
12            if cluster not in clusters_os:
13                clusters_os[cluster] = []
14            clusters_os[cluster].append(index if index < s else index
                  + 1)
15
16        # comparing models for each data point
17        o_ratings = {}  # key = data point index; value = its
              evaluation
18        for o in os_data.index:
19            o_cluster_original_indices = set(next(c_datapoints for
                  c_datapoints in clusters_original.values() if o in
                  c_datapoints))
20            o_cluster_original_indices.discard(s)
21            o_cluster_os_indices = set(next(c_datapoints for
                  c_datapoints in clusters_os.values() if o in
                  c_datapoints))
22            o_union = len(o_cluster_original_indices.union(
                  o_cluster_os_indices))
23            o_intersection = len(o_cluster_original_indices.
                  intersection(o_cluster_os_indices))
24            o_rating = o_intersection / o_union
```

```
25            o_ratings[o] = o_rating
```

Listing 2.11: Calculating score for each data point for SA

```
1         # aggregating over clusters
2         scores_per_cluster = []
3         for c, c_datapoints_orig in clusters_original.items():
4             c_datapoints = [cdp for cdp in c_datapoints_orig if cdp
                  != s]
5             if len(c_datapoints) == 0:
6                 continue  # The omitted point s alone builds a whole
                      cluster -> cluster can be ignored
7             scores_per_cluster.append((1.0 / len(c_datapoints)) * sum
                  (o_ratings[j] for j in c_datapoints))
8
9         # aggregating for whole dataset
10        estimation = (1.0 / len(scores_per_cluster)) * sum(
              scores_per_cluster)
11        estimations_for_missing_datapoints.append(estimation)
12
13    # aggregating over all omitted data points
14    sensitivity_analysis_score = (1.0 / len(datapoint_indices_to_omit
          )) * sum(estimations_for_missing_datapoints)
```

Listing 2.12: Aggregating scores for SA

2.5 Security

2.5.1 Silhouette Coefficient

Explanation

The Silhouette Coefficient (SC), also known as the Silhouette Score, was introduced in [30]. It combines the intracluster and intercluster distances into one metric to describe whether the clusters have a high density and are well-separated.

The intracluster distance is the average distance between a data point and its own cluster's other data points. In turn, the intercluster distance is the average distance between a point and the data points of the closest adjacent cluster. If the relative difference between these two distances is high, the SC will show a good result.

This metric was not originally designed to measure the security of a model. However, it indirectly relates to security issues such as integrity. A high score shows a high consistency of the clustering. Thus, if the score is low, this might indicate that the data has been manipulated. As another example, outliers lower this score. Outliers might also hint at security breaches.

Calculation

The score is calculated for each data point individually and then aggregated. For a single data point $p \in P$, first the intracluster distance $d_{intra}(p)$ is calculated as shown in equation 2.29. C_p refers to the data points in the cluster containing p, and $d(i, j)$ is the Euclidean distance between i and j.

$$d_{intra}(p) = \frac{1}{|C_p| - 1} \cdot \sum_{j \in C_p, j \neq p} d(i, j) \tag{2.29}$$

Next, the intercluster distance $d_{inter}(p)$ is computed as shown in equation 2.30. Here, C_{next} refers to the data points in the closest cluster that does not contain p. The closeness here refers to the distance between the cluster center and p.

$$d_{inter}(p) = \frac{1}{|C_{next}|} \cdot \sum_{j \in C_{next}} d(i,j) \tag{2.30}$$

Having both results, the silhouette score $s(p)$ can be determined as shown in equation 2.31.

$$s(p) = \frac{d_{inter}(p) - d_{intra}(p)}{\max(d_{intra}(p), d_{inter}(p))} \tag{2.31}$$

Finally, these scores are aggregated over all data points to find the final result S. Equation 2.32 demonstrates this.

$$S = \frac{1}{|P|} \cdot \sum_{p \in P} s(p) \tag{2.32}$$

Normalization

The normalization of this metric may first seem simple. The result range of the metric is -1 to 1. This can be scaled to 0 to 1 in a straightforward manner. However, this might not be the best solution as can be seen when analyzing the meaning of the different values.

Negative results represent a clustering where the points are on average closer to a different cluster than their own. A score of 0 would mean that the points are equidistant between their own cluster and the next one. These results all show that the clustering is rather poor. Still, with a linear scaling, the original result 0 would be mapped to a 50% score. This seems far too high for the actual underlying meaning.

Taking this into account, the score should be normalized in a different manner. Since the original score 0 is poor, a fixed point should be found to map it to, which can be recognized by the user as a poor result.

In this work, 0 is mapped to 0.2. 20% should be low enough to suggest to most users that their model's security is insufficient. The value 0.2 does not have a deeper significance and should be analyzed further in future work to make sure that the score's proportion fits the other metrics'.

Having established this point, the scaling can be applied in two parts. First, the original scores -1 to 0 are linearly mapped to 0 to 0.2. Next, the scores 0 to 1 are also linearly mapped to the normalized results 0.2 to 1. This way, the normalization seems to represent the scores' deeper meaning sufficiently. The formula can be seen in equation 2.33.

$$S_{normalized} = \begin{cases} 0.2 \cdot (S+1) & \text{if } -1 \leq S \leq 0 \\ 0.8 \cdot S + 0.2 & \text{if } \quad 0 < S \leq 1 \end{cases} \tag{2.33}$$

Discussion

Although this metric is commonly used, there are a few points that should be considered. As mentioned above, it does not primarily focus on security. Even though the result does indicate the model's security quality, it should be combined with other metrics. This way, a better overall security evaluation can be reached.

Just like some other metrics such as 2.3.2, this metric's complexity is high because it needs to consider the connections between many pairs of data points. However, the implementation here, as shown below, uses an optimized library. This library uses code in a machine-oriented language and thus, the calculation is still rather fast compared to the other metrics.

Implementation

Using an algorithm from the library scikit-learn [28], the implementation can be completed concisely as listing 2.13 displays.

```
1    from sklearn.metrics import silhouette_score
2
3    sc = silhouette_score(data, self.model.predict(data, encoded=True
         ))
4    sc_normalized = 0.2 * (sc + 1) if sc <= 0 else 0.8 * sc + 0.2
```

Listing 2.13: Calculating the SC

2.5.2 Davies-Bouldin-Index

Explanation

The Davies-Bouldin-Index (DBI) was introduced in [9]. Similar to the SC, an intracluster score is compared to an intercluster score. These scores are not the same though. Here, they are the variance within a cluster and the distance between two cluster centers.

Just like the SC, this metric's original intention is not for security evaluation. However, it is heavily influenced by security risks such as anomalies. A good score also indicates well-defined clusters which are essential for a secure model. Thus, this score gives a good impression of a clustering's security.

Calculation

The DBI is computed for each cluster and finally aggregated. First, for each cluster $c \in C$, the intracluster variance $S(c)$ is calculated as in equation 2.34. m_c is the cluster center of cluster c and $d(p, m_c)$ is the Euclidean distance between p and m_c.

$$S(c) = \frac{1}{|c|} \cdot \sum_{p \in c} d(p, m_c) \tag{2.34}$$

Next, for each pair of clusters, the intercluster distance $M(c_1, c_2)$ is calculated as shown in equation 2.35.

$$M(c_1, c_2) = d(m_{c_1}, m_{c_2}) \tag{2.35}$$

Now, these two scores are combined. For each cluster, the ratio between them is calculated for all other clusters. The relevant one is the combination with the highest score, named $R(c)$. Equation 2.36 shows this procedure.

$$R(c) = \max_{c_i \in C \setminus \{c\}} \left(\frac{S(c) + S(c_i)}{M(c, c_i)} \right) \tag{2.36}$$

In the final step, the final score D is found as a simple aggregation of these scores over all clusters. This is shown in equation 2.37.

$$D = \frac{1}{|C|} \cdot \sum_{c \in C} R(c) \tag{2.37}$$

Normalization

The lowest and best possible score for this metric is 0. There is no upper limit though, which makes the normalization difficult. However, research such as [17] shows that the score often ranges between 0.269 and 0.77. The models tested in this work additionally show that higher scores of about 2.5 are not uncommon but less frequent.

Having this information, a sigmoid function is very suitable to solve the problem accordingly. Equation 2.38 shows the basic form of the sigmoid function.

$$f(x) = \frac{L}{1 + e^{-k \cdot (x - x_0)}} \tag{2.38}$$

Since the best score after normalization should be 1, the originally best score 0 must be mapped to 1. Since the score is often relatively low as explained above, it is good that the normalized score first has a higher negative slope to emphasize differences in this range. Afterwards, it can slowly flatten and approach 0.

With these considerations in mind, the inflection point x_0 can be set to 0 and consequently the maximum L must be 2. The steepness k must be negative to ensure the correct orientation of the function. Here, it is set to the value -0.5. This results in the curve shown in figure 2.3 and the final normalization function shown in equation 2.39.

$$f(x) = \frac{2}{1 + e^{0.5 \cdot x}} \tag{2.39}$$

The value -0.5, as requested, ensures a significant difference between the most common values while being still quite flat. This way, the original scores up to around 10 are well-distinguishable after normalization. Only above 10, the curve gets close to 0, where there are only slight differences.

This way, all possible values are considered with focus on the most important ones. However, in further research, the exact parameters might be improved after deeper inspection of the scores on many different datasets.

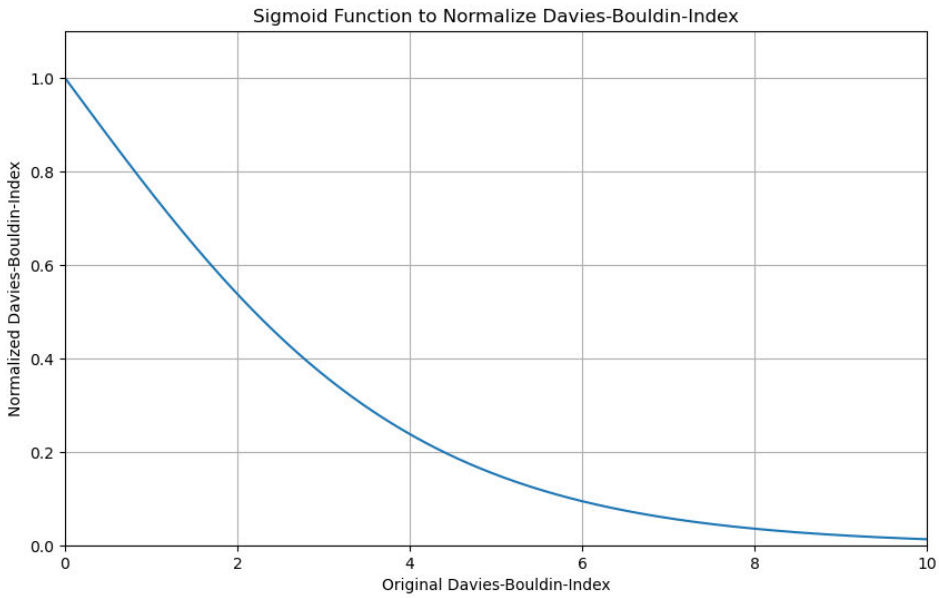


Figure 2.3: Visualization of Sigmoid Function for Normalization of Davies-Bouldin-Index

Discussion

The DBI clearly shows to what extent the clusters are compact within themselves and well-separated from other clusters. This generally gives a good impression of the security, especially with regard to discerning anomalies such as outliers.

The algorithm is also relatively fast. Its complexity is only $O(n^2)$ with n being the number of clusters instead of the number of data points. This makes the metric perfect for quick feedback during the development process of a new clustering.

However, since the metric was not originally introduced with security in mind, it should always be considered with caution and supplemented with another metric if possible.

Implementation

Listing 2.14 shows how the DBI can be implemented by using the available function from the library scikit-learn [28].

```
    from sklearn.metrics import davies_bouldin_score

    dbi = davies_bouldin_score(data, self.model.predict(data, encoded
        =True))
    dbi_normalized = 2.0 / (1.0 + np.exp(0.5 * dbi))
```

Listing 2.14: Calculating the DBI

2.6 Explainability

2.6.1 Explainability by Decision Trees

Explanation

The explainability of a clustering model is complex to evaluate. In literature, there are few approaches. Most publications concentrate on how to explain a clustering instead. Thus, the idea here is also to first explain the clustering and then, in a next step, to assess the quality or, more precisely, the simplicity of the explanation.

Most attempts to explain a clustering use a Decision Tree (DT) as support. [24] and [8] are examples. After training the DT, its structure can be analyzed and if it is rather simple, the clustering's explainability can be rated good.

Building the Decision Tree

Many different works such as [24, 8] have complex ideas to transform a clustering into a DT, which is easy to understand, and thus has a high explainability score. However, this is not required here, since the intention is not to actually explain the model well, but to find a comparable score indicating to what extent an explanation is possible.

Considering this, it is sufficient to create a DT in the simplest way. Since the form of creating it is the same for all clusterings to be evaluated, the comparability is thus not influenced.

The most straightforward way to create a DT is introduced in [7] as Classification and Regression Trees (CART). Using this kind of tree, a termination criterion is still required. Usually, criteria such as maximum depth or minimum samples per leaf are prominent. However, these directly influence the DT's structure. Since the structure shall later directly describe the explainability, this would circumvent the intention and result in similar results for any model.

Instead, it might be useful to add nodes to the tree until a certain accuracy is reached. In this case, the condition would be fulfilled that a clustering that can easily be explained is modeled into a simple DT while complex clusterings that are hard to explain would end up as a deep DT.

The exact required accuracy is not very significant. It is important to set it to a fixed value. This way, explainabilities of different models can be compared. Here, the value 95% was chosen to ensure that most data points are classified correctly, but the model will not be extremely deep in most cases. In further study, this value could be considered more detailed, especially to ensure that the final explainability score is comparable to other metrics. For example, the result range 0.8 - 1.0 should mean good explainability instead of 0.2 - 1.0.

Evaluating the Decision Tree

Once the DT is trained, the actual evaluation can be completed. Information such as the tree depth, number of leaves, number of features used or average path length describe the structure and can be combined into an explainability score.

For all these characteristics, it is a fact that the smaller their value, the better the explainability. Considering this, a combination can be found as shown below.

Calculation

To train the DT, the data points are split into a training group and a test group. The labels are the predictions of the clustering model. Using the training data, the tree is built by adding one leaf after another. After each added leaf, the test data is used to check if the required accuracy of 95% is reached.

Afterwards, the metrics can be calculated. The tree depth d can directly be read from the tree. It is the number of nodes passed to get from the root to the furthest leaf node. The number of leaves l can also be counted on the tree, just like the number of used features f.

Only the average path length p needs some calculating. Every data point dp in the test data set D is predicted using the Decision Tree and the number of nodes passed $d(dp)$ is counted. Then, they are aggregated as in equation 2.40.

$$p = \frac{1}{|D|} \cdot \sum_{dp \in D} d(dp) \tag{2.40}$$

After normalizing these scores as explained below, they are aggregated. Equation 2.41 demonstrates this. Achieved is the final explainability score E.

$$E = \frac{1}{4} \cdot d_{normalized} + l_{normalized} + f_{normalized} + p_{normalized} \tag{2.41}$$

Normalization

The normalization here needs to ensure that the different scores can be combined well into a final score. Thus, it is easiest to transform each of the four individual scores into a number between 0 and 1. This way, the aggregation shown above can be used to combine them.

The scores are all numbers that are at least 1 and to all, it applies that the smaller the number, the better the explainability. In addition, the numbers can grow infinitely. Only the number of used features is limited by the form of the underlying data. Since there is no limit for the number of features of the data though, this number can still grow endlessly.

Considering this, the functions normalizing the scores can use the same structure. The normal distribution as in equation 2.42 works well here. Since the highest point is always 1, the parameter a can be set to 1 for all scores. b and c need to be assigned individually.

$$norm(x) = a \cdot \exp\left(-\frac{(x-b)^2}{2c^2}\right) \tag{2.42}$$

For the tree depth, the smallest possible result is 1 if the root node is directly connected to the leaf nodes. Thus, b can be set to 1. The standard deviation c influences how quickly the score falls. The value 4 was chosen. This way, very shallow trees are rated good while trees as deep as 10 nodes are rated under 10%. This seems appropriate because 10 layers deep trees are not very easy to understand and thus the explainability score should be low. Rearranging it accordingly leads to equation 2.43. The plotted curve can be seen in figure 2.4a.

$$d_{normalized} = \exp\left(-\frac{(d-1)^2}{2 \cdot 4^2}\right) = \exp(-0.0315(d-1)^2) \tag{2.43}$$

The smallest result for the number of leaves should be the number of clusters. Only this way, every possible cluster can be predicted using the DT. However, since the algorithm used here does not require this, there could be fewer leaves in an unequal clustering. Thus, the smallest possibility is 2 because the used library requires at least two leaf nodes.

Having set b to 2, the standard deviation c needs more careful consideration. At first, choosing a certain value seems to work here, as well. However, a clustering with more clusters usually does indeed need more leave nodes. This would mean that c should depend on the number of clusters. Implementing this, however, would have several disadvantages. It would make it easy to influence the score by adding nearly empty clusters to the base model. Moreover, this would result in a better score even though the explanation is still not easier to understand. From this point of view, it would seem more purposeful to ignore the base model structure. Then, a more complex clustering with many clusters is harder to explain than an easy one with few clusters. In common sense, this seems logical.

Due to this, a fixed value is chosen. This should be a lot higher than for the tree depth though, because having more leaf nodes than layers is normal and does not influence the explainability that much. The value 20 has been chosen. This way, there is still a good score with several leaves. Only starting from 45 leaves will the score drop to under 10%. This fits the purpose well. Equation 2.44 is the adapted formula and figure 2.4b shows the curve.

$$l_{normalized} = \exp\left(-\frac{(l-2)^2}{2 \cdot 20^2}\right) = \exp(-0.00125(l-2)^2) \tag{2.44}$$

The minimum number of used features is 1 because at least one separation must exist in the tree, therefore $b = 1$ again. As for the standard deviation c for this metric, the same consideration as for the number of leaves can be made. Here, it also seems that it should not depend on the number of clusters. The difficulty of understanding the explanation is not lower because there are more clusters. Thus, a fixed value should be set again. The number 3 was selected in this work. With this curve, using 8 or more features already results in a score below 10%. This is as desired because 8 different features are already complex to understand in a tree. The formula is shown in equation 2.45 and the curve in figure 2.4c.

$$f_{normalized} = \exp\left(-\frac{(f-1)^2}{2 \cdot 3^2}\right) \approx \exp(-0.05556(f-1)^2) \tag{2.45}$$

Finally, the average path length also has the minimum value of $b = 1$, just as for the tree depth. This metric is similar to the tree depth. In a perfectly balanced tree with all paths having the same depths, it would be the same value. But usually, it is smaller. Thus, the standard deviation should also be similar, but a little bit smaller. The same curve as was used for the number of used features with $c = 3$ fulfills this condition and is presented in equation 2.46 and figure 2.4d.

$$p_{normalized} = \exp\left(-\frac{(p-1)^2}{2 \cdot 3^2}\right) \approx \exp(-0.05556(p-1)^2) \tag{2.46}$$

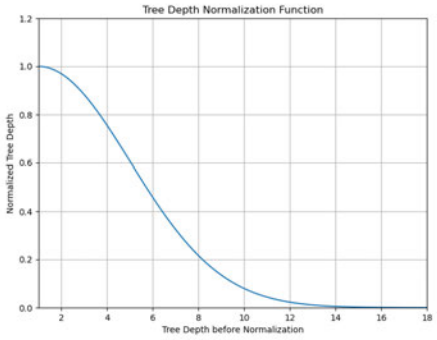
Discussion

This metric as described still has several disputed points with room for improvement. Besides the ones mentioned above, it is still possible to have data which never reaches an accuracy of 95% for the tree. This would result in an endless calculation. However, there is a simple solution to this problem. A limited number of iterations can be used. When reaching the limit, the worst score of 0 is returned.

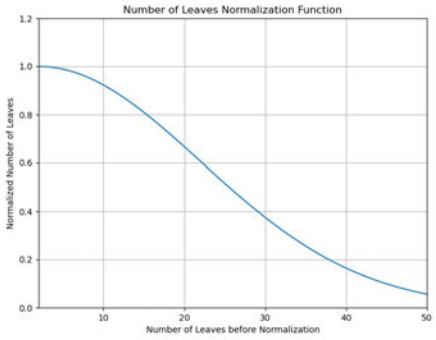
Another idea would be to create a DT with a fixed number of leaves or similar. Then the tree's accuracy could be used as the score. This was not implemented because it seems more likely to answer the question of whether a simple explanation for the clustering is sufficient instead of the question how understandable a complete explanation is. The idea could still be considered in further research though.

A more concrete issue with the current approach is the argumentation used for the normalization curves. The parameters of the functions were chosen based on the layout of rather simple clusterings. Having very complex data with many clusters would automatically end in a more complex explanation. It is questionable if it is fair to measure these models with the same standards.

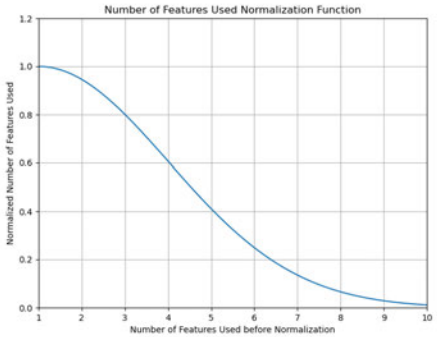
Ignoring these issues, the presented metric still seems to provide a close approximation of the explainability though.



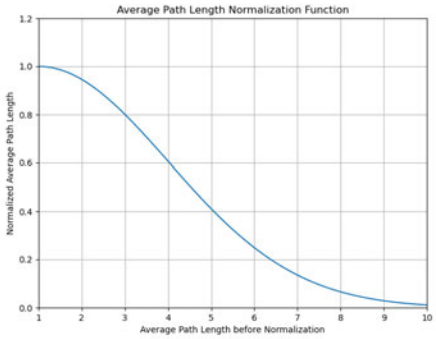
(a) Normalization Function Plot for the Tree Depth



(b) Normalization Function Plot for the Number of Leaves



(c) Normalization Function Plot for the Number of Features Used



(d) Normalization Function Plot for the Average Path Length

Figure 2.4: Normalization Function Plots for Explainability Score using Decision Trees

Implementation

When implementing this metric, the first step is to prepare the data. Listing 2.15 shows this process. Next, listing 2.16 shows how the DT is trained step by step until the required accuracy is reached. Finally, the metrics are calculated and normalized and combined as shown in listing 2.17.

```
1    labels = clustering_model.predict(data, encoded=True)
2    x_train, x_test, y_train, y_test = train_test_split(data, labels,
         test_size=0.2, random_state=0)
```

Listing 2.15: Preparing data for DT Explainability

```
    tree = DecisionTreeClassifier(max_leaf_nodes=1, random_state=0)
    max_leaf_nodes = 1
    max_leaf_nodes_limit = 1000
    accuracy = 0

    while accuracy <= 0.95 and max_leaf_nodes < max_leaf_nodes_limit:
        max_leaf_nodes += 1
        tree.max_leaf_nodes = max_leaf_nodes
        tree.fit(x_train, y_train)
        y_pred = tree.predict(x_test)
        accuracy = accuracy_score(y_test, y_pred)
```

Listing 2.16: Training the tree for DT Explainability

```
    if max_leaf_nodes >= max_leaf_nodes_limit:
        explainability = 0.0
    else:
        tree_depth = tree.get_depth()
        tree_depth_score = math.exp(-0.03125*(tree_depth - 1)**2)

        number_of_leaves = tree.get_n_leaves()
        number_of_leaves_score = math.exp(-0.00125*(number_of_leaves
            - 2)**2)

        used_features = np.unique(tree.tree_.feature)
        used_features = used_features[used_features >= 0]  # -2
            represents leaves
        number_of_features = len(used_features)
        number_of_features_score = math.exp(-0.05555555555555555*(
            number_of_features - 1)**2)

        average_path_length = np.mean(tree.decision_path(x_train).sum
            (axis=1))
        average_path_length_score = math.exp(-0.05555555555555555*(
            average_path_length - 1)**2)

        explainability = (tree_depth_score + number_of_leaves_score +
             number_of_features_score + average_path_length_score) / 4
```

Listing 2.17: Calculating metrics for DT Explainability

3 Test Models

3.1 Clustering Algorithm Types

As mentioned in chapter 2.1.3, there are different kinds of clustering algorithm, as described e.g. in [22, 34]. In this work, four of them are considered: Partition-Based Clustering (PBC), Hierarchical Clustering (HC), Density-Based Clustering (DBC), and Model-Based Clustering (MBC). In the following sections, these are presented and a model is introduced for each type.

These models are later used to test the implemented metrics and show their functionality. To do so, there are very few requirements for these models. The target here is not to create models with perfect scores, but to show how the metrics work and highlight their diversity when handling different algorithms.

Consequently, the models were created using the basic clustering algorithms and no additional focus on responsibility.

3.2 Partition-Based Clustering Model

Principle

At PBC, a fixed number of cluster centers is needed. The data points are assigned to their closest cluster center. Then, these cluster centers are moved to the middle of their assigned data points. These two steps are repeated until the assignment does no longer change or another stop criterion is reached.

Specific Algorithm

The most common specific algorithm is k-means, introduced in [25]. This algorithm is chosen here for the test model. It follows the steps explained above, but is more specific. Here, the cluster centers are the centroids of their cluster's data points based on the Euclidean distance.

Data Set

The used dataset is the one published at [31]. It contains data for patients with diabetes, including some personal data such as age and weight and many statistics regarding their health such as blood values and information about their hospitalization.

The dataset has more than 100,000 rows. Although the metrics work with this amount of data, the calculation for some takes a longer time. To increase the speed during the development, only a part of the data was used, specifically 1,000 rows.

The values of this dataset are not all numeric. The according attributes were encoded using ordinal encoding. Some other attributes such as IDs were removed because they have no influence on the patient's medical condition.

No further preprocessing of the data was performed, again, because the targets do not include building a good model. This way, the data can already be used to create a test model.

Implementation

The complete implementation can be found in appendix A.2.1.

3.3 Hierarchical Clustering Model

Principle

HC uses a tree-like approach to cluster data. There are two general ways to achieve this. Agglomerative Nesting initializes every data point as its own cluster and then joins close

clusters step by step. Divisive Analysis starts with one cluster containing all data points. The cluster is then split apart step by step.

Specific Algorithm

In this work, the Single Linkage Clustering algorithm is used, which was introduced in [20]. It is a version of the Agglomerative Nesting and thus starts with each data point representing an individual cluster.

Then, in each step, the pairwise distance between all clusters is calculated. It is defined as the smallest distance between one point in the first cluster to one point in the second. The two closest clusters are then combined. Here, the used distance metric is the Euclidean distance.

This procedure is repeated until a stop criterion is reached. In our case, a simple criterion is selected. When there are only ten clusters left, the algorithm finishes.

Data Set

For this model, the dataset [4] was used. It contains health data of patients with heart disease, as well as their age and gender. It contains about 300 patients' records, which makes even high-complexity metrics finish quickly on a model. Thus, all data was used to train the model.

This dataset only contains numeric data and after removing the ID value, there is no further preprocessing required and the model can be trained.

Implementation

The concrete implementation is in appendix A.2.2.

3.4 Density-Based Clustering Model

Principle

DBC algorithms define clusters as the areas with a high density of data points. Areas with few data points separate these clusters. Clusters are created incrementally. They are extended by adding close data points.

Specific Algorithm

The most well-known algorithm Density-Based Spatial Clustering of Applications with Noise (DBSCAN), introduced in [13], is used here. It uses two parameters, a radius and a minimum number. Every data point with at least this number of data points not further away than the radius is considered a core point. Every data point with fewer data points this close by, but in range of a core point is considered a border point. The remaining data points are noise points.

The algorithm then begins with an initial data point and if it is a core point, it is initialized as a cluster and the neighboring data points are added to it. Then this process is repeated. For every core point, the surrounding, not yet to another cluster assigned core and border points are added to its cluster.

Data Set

For this model, the dataset from [5] was used. It contains data for patients with panic disorder. There is information about the personal situation, e.g. symptoms and if there are more cases in the family. Personal information such as age and gender are also listed and the living environment.

Many attributes in this dataset are non-numeric and are preprocessed the same way as the other datasets for the other models using ordinal encoding. Here, the ID was removed as well.

There are 120,000 records in this dataset. For easier testing, only 1,000 records are used for the clustering.

Implementation

In appendix A.2.3, there is the full code and an explanation of the implementation.

3.5 Model-Based Clustering Model

Principle

The concept of MBC is to use probabilistic models. Every cluster is represented by a statistical model which shows the probability that a data point is included in this cluster.

Specific Algorithm

Gaussian Mixture Models (GMM) is a common MBC algorithm using Gaussian distributions. For this algorithm, every cluster is described as a normal distribution which is iteratively approximated using the Expectation-Maximization Algorithm (EM), as explained in [6].

Data Set

The used data is the dataset on [29]. It contains records of breast cancer patients with many statistic values describing the cancer. Apart from that, there is no personal data.

After removing the ID and ordinal encoding the only non-numeric attribute which describes if the cancer is benign or malignant, the data can be used. Since there are fewer than 600 records, they are all used in this model.

Implementation

The implementation is in appendix A.2.4.

3.6 Handmade Model

The afore mentioned models are able to demonstrate that the metrics are generally usable on these kinds of clustering algorithms. However, it is hard to evaluate how accurate they are because the quality of the models is not known and would require a deep analysis of the data.

Instead, a new model is created with manually fabricated data. The data contains 100 records with x and y values. It builds four well-separated clusters consisting of 25 data points each. The data points were randomly chosen around the cluster center using a normal distribution. Figure 3.1 shows the exact arrangement. The red crosses here show the cluster centers and the data points of the same cluster have the same color.

Figure 3.1: Visualization of Clustering of Manually Created Data Points for Test Model

As can be seen in the figure, this model should have very good scores. This way, the metrics' ability to recognize responsible models can be evaluated.

Technically, this model is represented by a k-means model. Not only can k-means perfectly portray the model in its supposed way, but it can also be evaluated with all metrics, including Inertia and DPL. The implementation is in appendix A.2.5.

4 Experiment Results

4.1 Introduction

The introduced metrics are all executed on the prepared test models. This chapter gives an overview of the results. The scores of the metrics are separately analyzed. For the first four models, it is shown how far the metrics generally work for the different versions of clustering algorithm. Using the last model, their proficiency in pointing out responsible models is highlighted.

For each model, first the scores are all shown in a diagram. After a short analysis, they are aggregated to show the assessment of the aspects of RAI as defined in chapter 2.1.4. During this, the values are always rounded to three decimal places for the metrics or two decimal places for the RAI aspects. The calculation is based on the original values though, so some rounding errors might occur.

4.2 K-Means Results

The scores calculated by the metrics for the prepared k-means model are illustrated in figure 4.1. It can be seen that no values are very high or very low. This is in line with expectations, considering that during the creation of the model no precautions were taken to ensure good RAI scores.

- The Inertia score of 0.704 shows that the clusters are quite compact, but there is still room for improvement. However, this score is also related to the underlying data and the chosen number of clusters. It is best to use this score to compare models on the same data with the same number of clusters. This way, improvements can be highlighted. In absolute terms, it is possible that this rating is already very high for this combination.

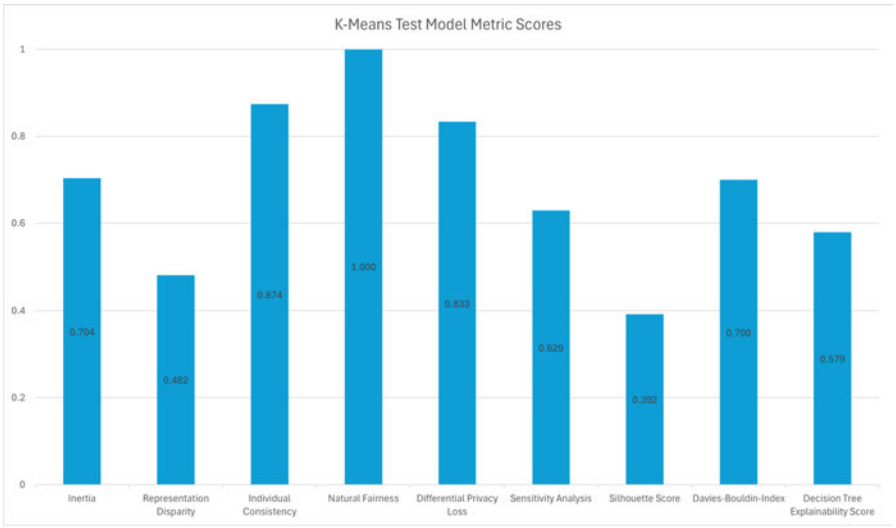


Figure 4.1: Metrics Results for K-Means Test Model

- The RD score of 0.482 is mediocre. This metric does not evaluate any of the test models as satisfactory. This might be due to its nature which is further discussed below where the other test models score even worse.
- The average IC score is 0.874. Since the metric heuristically selects data pairs to allow faster evaluation for larger models, the scores vary a little. This average is based on three results: 0.874, 0.880 and 0.869. It can be seen that the fluctuation is not high though, showing that the metric is quite stable. The score itself shows that about 87% of data pairs that are close to each other are in the same cluster. This is a high score, but the remaining 13% could still be improved if the clusters were even better separated.
- The NF is at 100%, i.e. every data point is closer to the other data points in its own cluster than the other data points. Hence, the clustering is fair for every individual data point. The nature of k-means might be supportive of this high score though, because every data point is by definition already in the cluster with the closest center. Nevertheless, this is no guarantee for a high NF score.
- The DPL reaches 0.833. This is again a mean of multiple scores: 0.730, 0.875 and 0.895. The added noise is random and as can be seen, this has a huge influence on the result. However, even though the result range is larger here, a conclusion can still be drawn. After the added noise, about 83% of data points end up in the same cluster. This means that every sixth data point is wrongly assigned. This shows

that a manipulation of the model results in a noticeable change which might allow the partial inference of data. Thus, the privacy protection should be improved for this model.

- The SA scores at 0.629, again as an average of 0.625, 0.626 and 0.637, due to the random selection of data points to be omitted. This shows that one missing data point has a huge impact on the entire clustering, which shows a poor privacy, consistent with the DPL score.
- The SC is 0.392. This means that the clusters are not well-separated with a high density or there are many outliers. This is problematic regarding the security score of the model. The reason might be that the chosen number of clusters does not fit the data or the form of the clusters does not represent the data well.
- The DBI of 0.700 indicates a better separation of clusters. The score, significantly higher than the SC, suggests that the separation of the clusters is indeed acceptable but several data points are not optimally assigned to the clusters.
- The DT Explainability score lies at 0.579, meaning the explanation of the model is on a medium-level difficulty.

Altogether, it is clear that the metrics work as expected and successfully evaluate the k-means model. To find the scores for the aspects of RAI, these results are combined according to the weighting presented in chapter 2.1.4. Figure 4.2 illustrates these combined results.

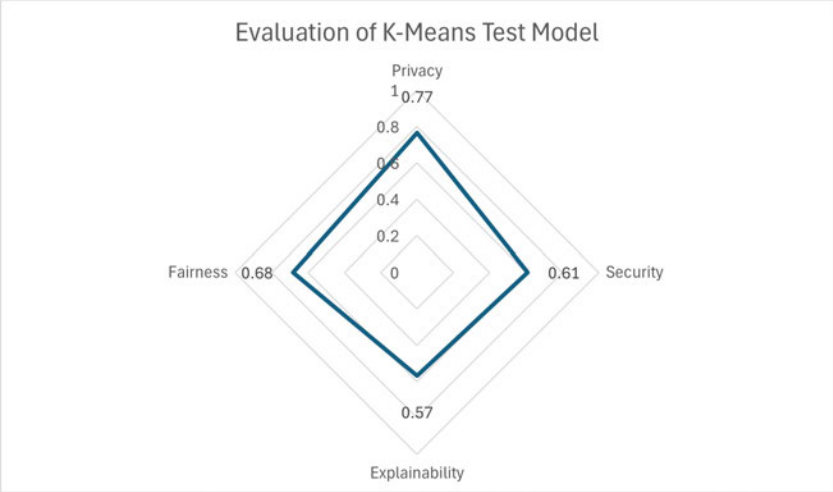


Figure 4.2: RAI Aspects Results for K-Means Test Model

The values are all in a range from medium to satisfactory, but none are really acceptable. In combination, the final responsibility score is their mean value 0.656. This means the model is not responsible yet and still needs improvement.

4.3 Single Linkage Clustering Results

Figure 4.3 shows the metrics' scores for the Single Linkage Clustering (SLC) test model. As explained in chapter 2.1.3, Inertia and DPL are only compatible with PBC and thus not included here.

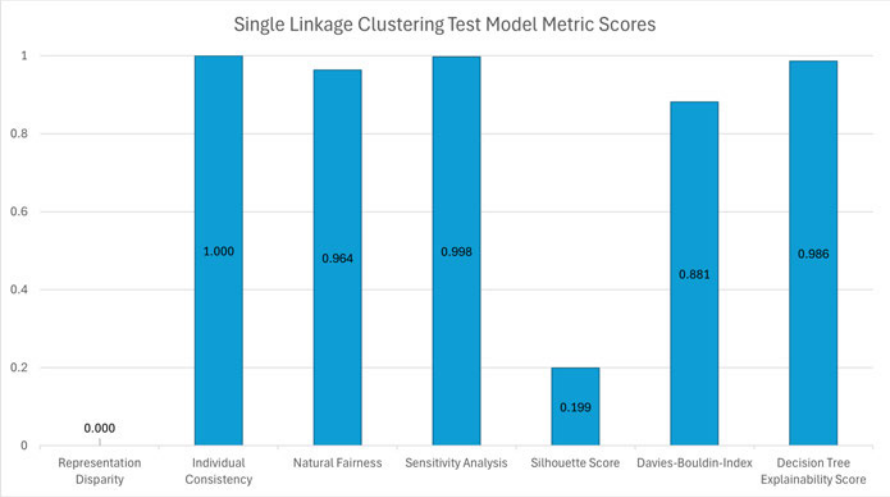


Figure 4.3: Metrics Results for Single Linkage Clustering Test Model

For this test model, the most scores are relatively high. There are two exceptions though, which are very low. The following analysis explains this unexpected circumstance.

- The RD score is exactly 0.000. The normalization process described in chapter 2.3.1 explains that particularly low scores are cut at the minimum value of 0. This is the case here. A reason for this is explained below in the analysis of the handmade test model in chapter 4.6.
- The IC reaches the perfect score of 1.000, even after multiple executions. All pairs of similar data points are assigned to the same cluster, indicating a great individual fairness.

- The NF is also very high with 0.964. This supports the finding of the IC. However, these scores are far higher than anticipated. To make sure this is no problem, a deeper look into the clustering was made. In appendix A.3, the allocation of data points to their clusters is shown. Here, it can be seen that almost all data points are in one single big cluster. This way, it is logical that these two metrics have high scores. This problem should be considered when evaluating a model.
- The SA score is after multiple executions stable at 0.998. A missing data point does not impact the model which would usually be an indicator for a great privacy evaluation. However, considering the unequal data point allocation, this is not surprising. Here again, when using this score, it must be paid attention to this problem.
- The SC is very low with 0.199. This is a clear warning and points out the problem mentioned above. Apart from that, this also shows a problematic security because of many outliers or an unclear separation of clusters. Since most clusters have just one data point, these can be considered outliers, explaining this low score.
- The DBI is 0.881. This means that the clusters have a good separation. This situation again highlights the need to combine different scores to evaluate RAI aspects. Neither SC nor DBI alone are enough to clearly assess the security here.
- The DT Explainability score values 0.986. Here again, this excellent explainability is easy to understand. Since almost all data points are in one cluster, a direct assignment to this cluster would directly result in a high accuracy for the explaining DT model.

On the one hand, this test model exposes some of the downsides of the individual metrics. However, it also shows the importance and efficiency of multiple metrics used in combination. On the other hand, it also shows that the metrics have no difficulty evaluating a HC.

The next step is anew to calculate the RAI aspects using the weighting in chapter 2.1.4. The scores are shown in figure 4.4.

The privacy and explainability here are quite high while the security and fairness are far behind with medium values. Considering the mentioned unequality of this model, it becomes obvious that it must be necessary for a model to reach high scores in all separate aspects in order to be responsible. The mean of these aspects is 0.744, which appears too

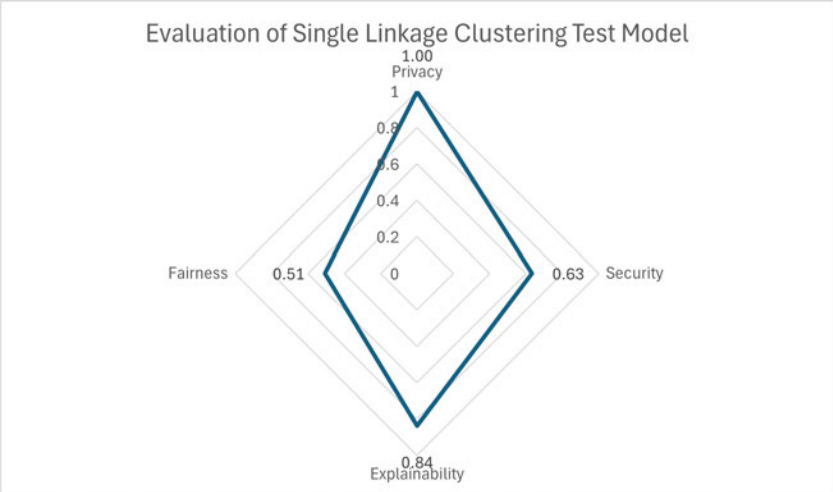


Figure 4.4: RAI Aspects Results for Single Linkage Clustering Test Model

high for such a problematic model. Thus, this overall score alone might not be sufficient to evaluate the responsibility of a model.

4.4 DBSCAN Results

The DBSCAN test model's test results are in 4.5. Here again, Inertia and DPL are missing, due to their compatibility.

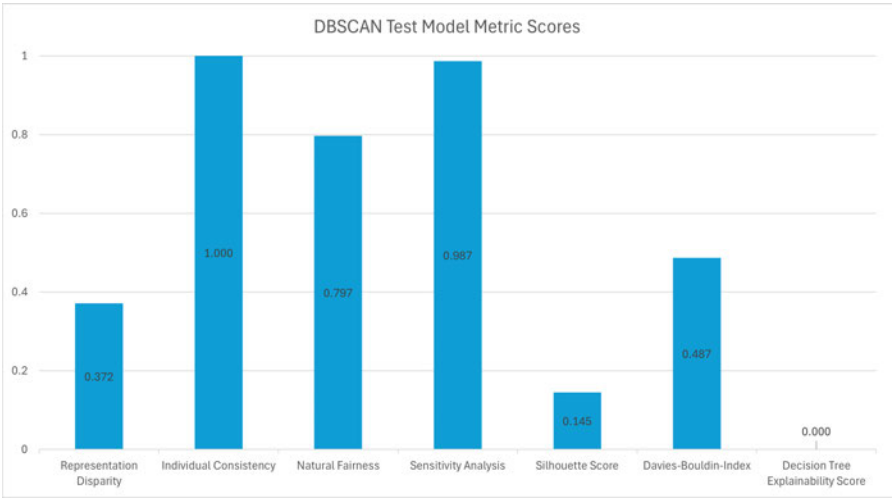


Figure 4.5: Metrics Results for DBSCAN Test Model

- The RD score is 0.372. This score is again rather low. The reason is explained below at the handmade model in chapter 4.6. Considering that, this score can still be considered quite acceptable.
- The IC is consistently at 1.000 again. Similar data points are always in the same cluster. Facing the same situation as before, the cluster allocation is studied again. Appendix A.3 shows that there are several clusters with few data points but also multiple clusters with many data points. Since even the bigger clusters have no very close members (as indicated by this score), the individual fairness is indeed good.
- The NF confirms this, although it is slightly lower at 0.797. When taking the fact into consideration, that all outliers are treated as one cluster here, this can be explained. Since they are spread widely, it is quite probable that an individual outlier is closer to the other clusters than to the other outliers. The overview in appendix A.3 shows that there are more than 30% of outliers. The missing 20% of the NF score fit into this set. In conclusion, the model offers a very good individual fairness.
- The SA scores 0.987 on average. The individual results of the calculation are very close though: 0.986, 0.987 and 0.988. This proves that single missing data points have very little influence on this model, profiting its privacy.
- The SC is again quite low with 0.145. This is as expected, considering the high percentage of outliers. For this metric, it is beneficial that they are considered as one cluster. The high distance between data points in this cluster leads to the low rating which suits the high number of outliers which indicate a low security.
- The DBI score is 0.487. It is less influenced by the outliers but still enough to be unsatisfactorily low. However, the high difference to the SC is an indication that the non-outlier clusters are relatively well-separated.
- The DT Explainability score here is 0.000, which exposes a significant problem with this metric. During the creation of the DT, the stop criterion of 95% accuracy is not reached. Instead, overfitting leads to a stagnant accuracy on the data used for testing. This problem is less prominent on models with more data points. For example, using a similar model trained on 20,000 data records of the same dataset results in a very high score of 0.986. However, the metric as defined here should be

used with caution on models with less data. An idea to solve this problem is given in chapter 6.

These results verify that the metrics can generally be used on DBC models. Yet, it is also shown that the uniform handling of outliers as one cluster is not the optimal solution for all metrics and there could be improvements. Furthermore, a serious issue in the DT Explainability metric was unveiled.

Next, the RAI aspects are presented in figure 4.6. The scores are still based on the weighting introduced in chapter 2.1.4.

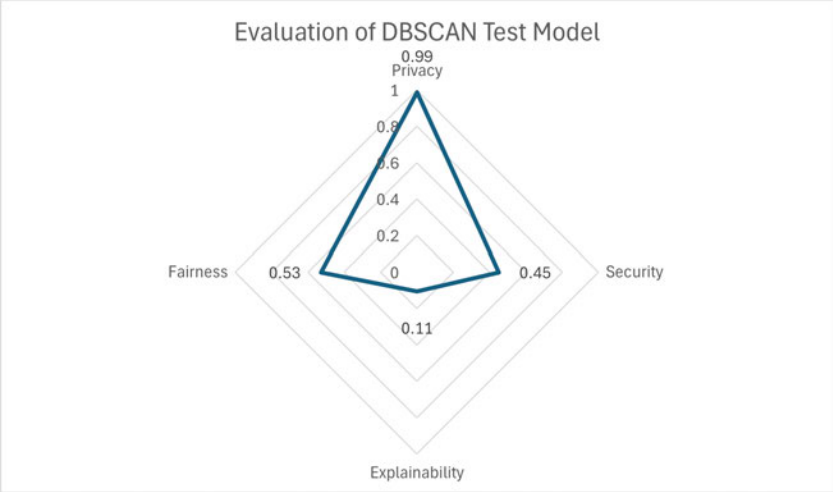


Figure 4.6: RAI Aspects Results for DBSCAN Test Model

The explainability of this model is heavily negatively influenced by the problematic DT Explainability score, resulting in a very low value. The fairness and security are in the medium range, which is primarily due to the high amount of outliers. Privacy has an excellent score. In combination, a total score of 0.518 is reached.

This is still heavily influenced through the inadequate DT Explainability. If instead the score mentioned above for the model with more data is used, the total score can be improved to 0.682. Still having the outliers in mind, this score seems decent. It is high enough to show the existing capacity of the model, while still showing outlier-based shortcoming.

4.5 GMM Results

Figure 4.7 visualizes the evaluations of the metrics on the GMM test model. Inertia and DPL are not available here either.

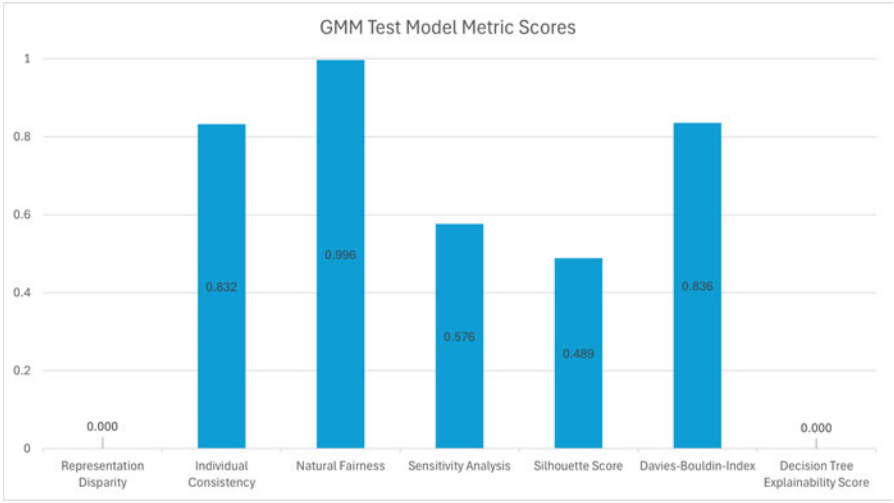


Figure 4.7: Metrics Results for GMM Test Model

- The RD score is here 0.000 again. As before, this is because of the implemented capping and might actually be rational, as explained in chapter 4.6.
- The IC score amounts to 0.832 which is the mean of three runs with the scores 0.840, 0.831 and 0.826. This means most similar data pairs are treated equally. The remaining ones close to 17% are still significant though and the model should still be improved in this regard to ensure individual fairness.
- The NF score is almost perfect with 0.996. This means almost every data point is closer to its own cluster than to the other clusters and underlines the high individual fairness of this model. Checking the distribution of data points to clusters in appendix A.3 further confirms this, since the distribution is relatively balanced.
- The SA reaches 0.576. This is the mean of the scores 0.569, 0.606 and 0.553. The model is very sensitive to small changes such as a single data point. This is a problematic plight in regards to the privacy.

- The SC of 0.489 is higher than at the previous models. This means that the clusters are better separated and have a higher density here. The value is still not satisfactory though, pointing to security problems here as well.
- The DBI score is 0.836. Just like at the k-means model, it is siginificantly higher than the SC, indicating that some data points might not be assigned to the best fitting cluster while the clusters are relatively dense after all.
- The DT Explainability score is 0.000 again, based on the overfitting problem mentioned above. The highest accuracy is 83.3%, reached after adding eight leaf nodes to the DT. When adding more, the accuracy starts to decrease. The targeted 95% is never reached. This distorts the model's actual explainability score and underlines the need to find a solution for this predicament.

It can be seen that these metrics basically work for GMM clusterings as well. Further, the previously mentioned DT vulnerability shows up again, emphasizing its gravity.

Finally, the scores can be combined using the weighting in chapter 2.1.4. The results are shown in figure 4.8.

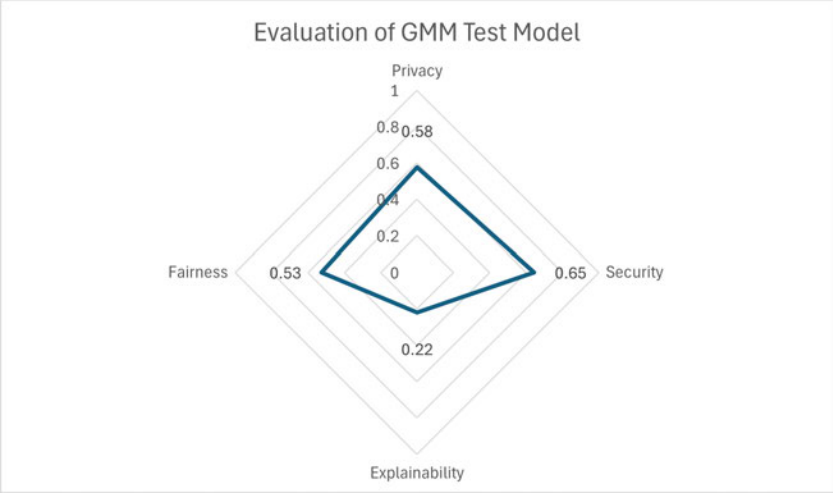


Figure 4.8: RAI Aspects Results for GMM Test Model

The privacy, fairness and security scores are all medium, which is in the area of expectation for a model trained without any focus on these aspects. The explainability score is, similar to the DBSCAN test model, far lower than it realistically should be. The final responsibility score is consequently 0.492.

4.6 Handmade Model Results

The final test model is the one based on specially shaped data. Since it is implemented using k-means, Inertia and DPL are available. The results of the metrics are shown in figure 4.9. Because of the nature of the data, very high scores are expected for this model. Most metrics fulfill this condition.

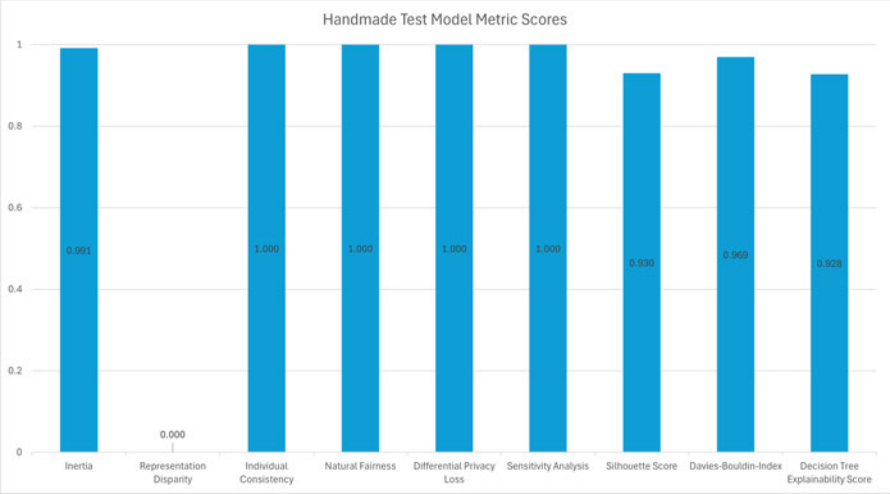


Figure 4.9: Metrics Results for Handmade Test Model

- The Inertia of this model is 0.991, confirming that the data points are all relatively close to their cluster center.
- RD is the only metric giving a low score for the model. It is 0.000. For the previous models, this score did mostly show low scores as well. This circumstance can be explained well for this model. The metric's expectation for a good score is that the data points with the same value for an attribute are equally distributed across the clusters. This is useful for an attribute such as race or gender in a situation where it is determined which people successfully get hired for a job.

  However, in this test model, this might not be useful. For example, looking at the attribute x, there are data points with the value 100. These data points are all in the two clusters on the left, as can be seen in figure 3.1 in chapter 3.6. All other values for both attributes are respectively limited to two of four clusters. Consequently, the distribution is always very poor, resulting is a low total score.

Though this evaluation for the group fairness is not wrong, it might be more meaningful to define a clear context before calculating this score. In clustering, it is usually expected to split the data based on certain values for attributes. Considering such a splitting as inadequate might go against the clustering's purpose.

If instead some sensitive attributes were selected for which a fair distribution is expected, and the RD is calculated based on these, the result would be more meaningful. The problem is that this procedure depends on the model. This conflicts with this works' purpose of being independent of the context. In further research, it is desirable to find a better solution to this problem.

In conclusion, this metric works as expected. However, it is important to be careful with the result and understand its meaning and to what extent a good score should be expected for the concrete model. For this handmade test model, a good RD score of this kind is not actually desirable.

- The IC for this model is 1.000. All pairs of close data points are in the same cluster. Since the data is designed with dense and distant clusters, this is logical and correct.
- The NF is 1.000 as well. Considering the data structure, it is correct that every data point is closer to the other data points in its own clusters than to those in other clusters. Thus, this evaluation is logical as well.
- The DPL score is also 1.000. Since the clusters are extremely well-separated with big distances in between, adding noise to the cluster centers is supposed to have no impact. Thus, this score is as expected.
- The SA has a constant score of 1.000. As expected, there is no change in the clustering after any single data point is removed from the dataset.
- The SC is 0.930, confirming the high density and clear separation of clusters. This score is not 100% though, since the data points are not all perfectly in the corresponding cluster center.
- The DBI of 0.969 is high as well and also shows that the clusters are very well-separated and dense.
- The DT Explainability score reaches 0.928. This means that it is easy to explain the model.

This model demonstrates how the different metrics work excellently at pointing out a very responsible clustering. It also helps to comprehend the RD score and its limits.

The RAI aspects, combined of the metric scores as explained in chapter 2.1.4, are shown in figure 4.10.

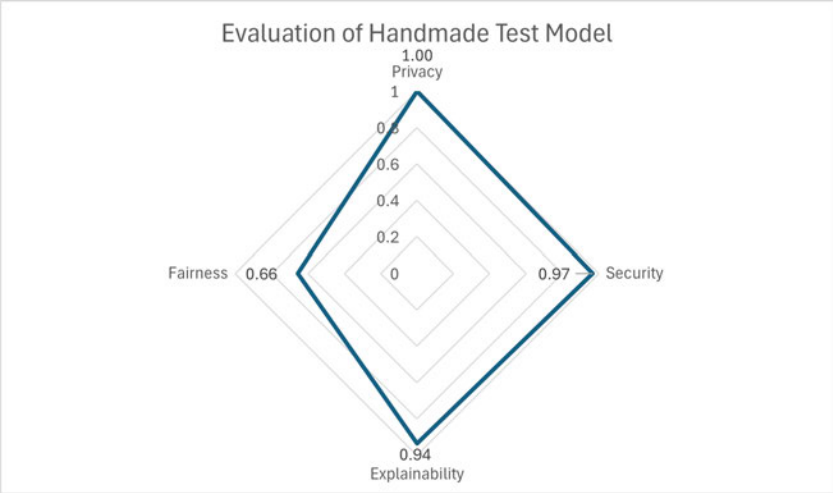


Figure 4.10: RAI Aspects Results for Handmade Test Model

Privacy, security and explainability all reach very high scores for this model. Just the fairness score is mediocre because of the included, very low group fairness value. Altogether, the responsibility score for this test model is 0.890.

5 Conclusion

Summary of Findings

In this work, a set of metrics was defined to evaluate to what extent a clustering model conforms to the aspects of RAI. These metrics were presented in detail and partially adapted to make them usable in a general context. They were further normalized, if necessary, and then implemented, adding to the framework presented in [16], available in the GitLab repository.[1]

To demonstrate the metrics, five test clustering models were built. Four of them represent the different kinds of clustering methods PBC, HC, DBC, and MBC. One model focuses on the underlying data, which is specially designed with the expectation of high scores.

During the test, the metrics' general functionality on different kinds of clustering algorithms is confirmed. At the same time, some still existing challenges were found that obstruct the full use of some of the metrics on all models. Further, the capability to recognize high-quality clusterings was confirmed.

Answering Research Questions

All four research questions are answered in this work.

RQ1: Which metrics are relevant to evaluate the aspects of RAI on clustering models?

An overview of available metrics was given, extended with further ideas for metrics, mostly based on algorithms aiming to improve the creation of a clustering. The metrics Inertia, Representation Disparity (RD), Individual Consistency (IC), Natural Fairness

[1] https://gitlab.com/sabrinagoellner/verifai_test_lab

(NF), Differential Privacy Loss (DPL), Sensitivity Analysis (SA), Silhouette Coefficient (SC), Davies-Bouldin-Index (DBI) and Explainability by Decision Tree were selected and it was shown how they can work together to evaluate the different aspects of RAI.

RQ2: To what extent is it possible to compare different models based on the evaluations of the metrics?

During the definition of the metrics and their application, a special focus was on the normalization and independence of the underlying data. Nonetheless, the experiments show that the data can still influence some of the metrics, making a universal comparability difficult. The improvement made through changes on the model based on the same data can be clearly shown, though.

RQ3: Which different kinds of clustering algorithms can be evaluated with these metrics?

Models for PBC, HC, DBC, and MBC were tested. By definition, Inertia and Differential Privacy Loss (DPL) are only meaningful for PBC models. The results show that the remaining metrics support all models. The occured problems could all be traced back to the data instead of the type of clustering algorithm.

RQ4: Are these metrics able to point out highly responsible models?

During a final test with a model based on data designed to form clear clusters, it was confirmed that the metrics reach high scores. The only exception is RD. Its score was correct though and the cause of the problem could be identified, although not resolved.

Limitations

While testing the metrics, some general findings were made. Since some metrics can be manipulated, e.g. by special data, these metrics have a limited meaning when used alone. Thus, it is always necessary to use multiple metrics in combination. Only in this way can they offset each other's shortcomings and give a reliable evaluation.

It was also found that the combined reliablity score can be heavily biased. It should not be used without analyzing the individual scores of the aspects of RAI. If it is used alone, the threshold for considering a model reliable should be chosen high enough to ensure high scores for all of the single aspects.

Further, heuristics were introduced for metrics with a high complexity. These allow quick evaluations, but they are also less exact. Especially the DPL highly varies in its results, even if the model is not changed. This makes a direct comparison difficult.

6 Further Research

There are still many possible improvements to be made in the future. In general, it is useful to add more metrics to improve the quality of the evaluation. The last kind of clustering algorithm, Grid-Based clustering, should also be considered and its compatibility with the current metrics.

Additionally, the combination of the aspects of RAI should be further analyzed. A different method of aggregation might be more reliable in defining whether a model is responsible. A proposal, for instance, could be to increase the weight of the aspect with the lowest score.

Another suggestion is to change the handling of the outliers of a DBSCAN model. After analyzing which metrics are more precise when considering each outlier as a different cluster, more significant evaluations could be reached.

Furthermore, the problem of overfitting when training the DT to evaluate the explainability needs to be solved. Apart from common methods to avoid overfitting, an idea how to achieve this would be to record the accuracy of the DT during its creation. If it does not reach the targeted 95%, the version with the highest accuracy could be used for the evaluation instead. Depending on its accuracy, the final score could be reduced accordingly.

A major general improvement could additionally be the manual definition of a distance metric. If the used distance metric perfectly suits the data in a model, it can be evaluated far more precisely. The Euclidean distance used throughout this work ignores differences in the value ranges of different attributes, which might be unfair. For example, the age of a person (value range about 0-120) would have a far lower impact than the income (value range about 0-100,000). With the implementation of this feature, the tested models could also use all kinds of data, such as images, as long as the suitable distance metric is given.

Bibliography

[1] *Laplace Distribution*. S. 294–295. In: *The Concise Encyclopedia of Statistics*. New York, NY : Springer New York, 2008. – URL https://doi.org/10.1007/978-0-387-32833-1_219. – ISBN 978-0-387-32833-1

[2] Almanza, Matteo ; Epasto, Alessandro ; Panconesi, Alessandro ; Re, Giuseppe: k-Clustering with Fair Outliers. In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. New York, NY, USA : Association for Computing Machinery, 2022 (WSDM '22), S. 5–15. – URL https://doi.org/10.1145/3488560.3498485. – ISBN 9781450391320

[3] Anderson, Nihesh ; Bera, Suman K. ; Das, Syamantak ; Liu, Yang: *Distributional Individual Fairness in Clustering*. 2020. – URL https://arxiv.org/abs/2006.12589

[4] Awan, Abid A.: *Heart Disease Patients*. 2021. – URL https://www.kaggle.com/datasets/kingabzpro/heart-disease-patients. – Accessed on January 3, 2025

[5] Azeem, Muhammad S.: *Panic Disorder Detection Dataset*. 2023. – URL https://www.kaggle.com/datasets/muhammadshahidazeem/panic-disorder-detection-dataset. – Accessed on February 13, 2025

[6] Bishop, Christopher M. ; Nasrabadi, Nasser M.: *Pattern recognition and machine learning*. Springer, 2006

[7] Breiman, L. ; Friedman, J. ; Stone, C.J. ; Olshen, R.A.: *Classification and Regression Trees*. Taylor & Francis, 1984. – URL https://books.google.de/books?id=JwQx-WOmSyQC. – ISBN 9780412048418

[8] Dasgupta, Sanjoy ; Frost, Nave ; Moshkovitz, Michal ; Rashtchian, Cyrus: Explainable k-Means Clustering: Theory and Practice. In: *Proceedings of the 37th*

International Conference on Machine Learning (ICML), URL http://interpretable-ml.org/icml2020workshop/pdf/06.pdf, 2020

[9] Davies, David L. ; Bouldin, Donald W.: A Cluster Separation Measure. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1 (1979), Nr. 2, S. 224–227

[10] Domingo-Ferrer, Josep ; Sánchez, David ; Soria-Comas, Jordi: *Beyond k-Anonymity: l-Diversity and t-Closeness*. S. 47–51. In: *Database Anonymization: Privacy Models, Data Utility, and Microaggregation-based Inter-model Connections*. Cham : Springer International Publishing, 2016. – URL https://doi.org/10.1007/978-3-031-02347-7_6. – ISBN 978-3-031-02347-7

[11] Dunn†, J. C.: Well-Separated Clusters and Optimal Fuzzy Partitions. In: *Journal of Cybernetics* 4 (1974), Nr. 1, S. 95–104. – URL https://doi.org/10.1080/01969727408546059

[12] Dwork, Cynthia ; Hardt, Moritz ; Pitassi, Toniann ; Reingold, Omer ; Zemel, Richard: Fairness through awareness. In: *Proceedings of the 3rd innovations in theoretical computer science conference*, 2012, S. 214–226

[13] Ester, Martin ; Kriegel, Hans-Peter ; Sander, Jörg ; Xu, Xiaowei: A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1996 (KDD'96), S. 226–231

[14] Goellner, Sabrina ; Tropmann-Frick, Marina ; Brumen, Bostjan: *Responsible Artificial Intelligence: A Structured Literature Review*. 2024. – URL https://arxiv.org/abs/2403.06910

[15] Gupta, Shivam ; Jain, Shweta ; Ghalme, Ganesh ; Krishnan, Narayanan C. ; Hemachandra, Nandyala: *Group and Individual Fairness in Clustering Algorithms*. S. 31–51. In: Mukherjee, Animesh (Hrsg.) ; Kulshrestha, Juhi (Hrsg.) ; Chakraborty, Abhijnan (Hrsg.) ; Kumar, Srijan (Hrsg.): *Ethics in Artificial Intelligence: Bias, Fairness and Beyond*. Singapore : Springer Nature Singapore, 2023. – URL https://doi.org/10.1007/978-981-99-7184-8_2. – ISBN 978-981-99-7184-8

[16] Göllner, Sabrina ; Tropmann-Frick, Marina: VERIFAI -A Step Towards Evaluating the Responsibility of AI-Systems, 03 2023

[17] Idrus, Ali: Distance analysis measuring for clustering using k-means and davies bouldin index algorithm. In: *TEM Journal* 11 (2022), Nr. 4, S. 1871–1876

[18] Imola, Jacob ; Epasto, Alessandro ; Mahdian, Mohammad ; Cohen-Addad, Vincent ; Mirrokni, Vahab: Differentially private hierarchical clustering with provable approximation guarantees. In: *Proceedings of the 40th International Conference on Machine Learning*, JMLR.org, 2023 (ICML'23)

[19] Imola, Jacob ; Kasiviswanathan, Shiva ; White, Stephen ; Aggarwal, Abhinav ; Teissier, Nathanael: Balancing utility and scalability in metric differential privacy. In: Cussens, James (Hrsg.) ; Zhang, Kun (Hrsg.): *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence* Bd. 180, PMLR, 01–05 Aug 2022, S. 885–894. – URL https://proceedings.mlr.press/v180/imola22a.html

[20] Johnson, Stephen C.: Hierarchical Clustering Schemes. In: *Psychometrika* 32 (1967), Nr. 3, S. 241–254

[21] Kar, Debajyoti ; Kosan, Mert ; Mandal, Debmalya ; Medya, Sourav ; Silva, Arlei ; Dey, Palash ; Sanyal, Swagato: *Feature-based Individual Fairness in k-Clustering*. 2023. – URL https://arxiv.org/abs/2109.04554

[22] Kashyap, M ; Gogoi, S ; Prasad, RK u. a.: A Comparative Study on Partition-based Clustering Methods. In: *Int. J. Create. Res. Thoughts (IJCRT)* 6 (2018), S. 1457–1463

[23] Kleindessner, Matthäus ; Awasthi, Pranjal ; Morgenstern, Jamie: A notion of individual fairness for clustering. In: *arXiv preprint arXiv:2006.04960* (2020)

[24] Laber, Eduardo ; Murtinho, Lucas ; Oliveira, Felipe: Shallow decision trees for explainable k-means clustering. In: *Pattern Recognition* 137 (2023), S. 109239. – URL https://www.sciencedirect.com/science/article/pii/S0031320320007180X. – ISSN 0031-3203

[25] Lloyd, S.: Least squares quantization in PCM. In: *IEEE Transactions on Information Theory* 28 (1982), Nr. 2, S. 129–137

[26] Loukas, Orestis ; Chung, Ho-Ryun: *Demographic Parity: Mitigating Biases in Real-World Data*. 2023. – URL https://arxiv.org/abs/2309.17347

[27] Malandain, Grégoire ; Boissonnat, Jean-Daniel: Computing the Diameter of a Point Set. In: *Proceedings of the 10th International Conference on Discrete Geometry for Computer Imagery (DGCI)*. Berlin, Heidelberg : Springer-Verlag, 2002, S. 197–207. – URL https://www-sop.inria.fr/asclepios/Publications/Gregoire.Malandain/dgci-2002.pdf

[28] Pedregosa, Fabian ; Varoquaux, Gaël ; Gramfort, Alexandre ; Michel, Vincent ; Thirion, Bertrand ; Grisel, Olivier ; Blondel, Mathieu ; Prettenhofer, Peter ; Weiss, Ron ; Dubourg, Vincent ; Vanderplas, Jake ; Passos, Alexandre ; Cournapeau, David ; Brucher, Matthieu ; Perrot, Matthieu ; Duchesnay, Édouard: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[29] Reddy, T. Eshwar K.: *Breast Cancer (Diagnostic) Data Set*. 2023. – URL https://www.kaggle.com/datasets/teshwarkanthreddy/breast-cancer-diagnostic-data-set. – Accessed on March 2, 2025

[30] Rousseeuw, Peter J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. In: *Journal of Computational and Applied Mathematics* 20 (1987), S. 53–65. – URL https://www.sciencedirect.com/science/article/pii/0377042787901257. – ISSN 0377-0427

[31] Roy, Lakshya ; Tatman, Rachael: *Diabetic_ Dataset*. 2018. – URL https://www.kaggle.com/datasets/smit1212/diabetic-data-cleaning. – Accessed on September 11, 2024

[32] Rykov, Andrei ; De Amorim, Renato C. ; Makarenkov, Vladimir ; Mirkin, Boris: Inertia-Based Indices to Determine the Number of Clusters in K-Means: An Experimental Evaluation. In: *IEEE Access* 12 (2024), S. 11761–11773

[33] Singh, Harvineet ; Chunara, Rumi: Measures of Disparity and their Efficient Estimation. In: *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society (AIES '23)*. Montréal, QC, Canada : Association for Computing Machinery, Inc., 2023, S. 927–938. – ISBN 979-8-4007-0231-0

[34] Yin, Hui ; Aryani, Amir ; Petrie, Stephen ; Nambissan, Aishwarya ; Astudillo, Aland ; Cao, Shengyuan: *A Rapid Review of Clustering Algorithms*. 2024. – URL https://arxiv.org/abs/2401.07389

A Appendix

A.1 Tools

Table A.1 lists the tools used in the development of this thesis.

Table A.1: Tools

Tool	Usage
LaTeX	Creation of the document
Microsoft Copilot	Language improvements and organizational assistance
PyPlot	Creation of figures
Microsoft Excel	Creation of figures

A.2 Implementations

A.2.1 K-Means Implementation

The listing A.1 shows the concrete implementation of the test model using k-means. After reading the data from its file, it is preprocessed. The form of preprocessing is saved to a file to allow additional data to be processed likewisely. Finally, the model is created using the library scikit-learn [28] and saved as well.

```
1    data = pd.read_csv("resources/diabetic_data.csv", nrows=1001)  #
        title row + 1000 rows
2
3    ordinal_encoders = {}
4
```

```
5     fields_to_transform = ["race", "gender", "age", "weight", "
          payer_code", "medical_specialty", "diag_1", "diag_2", "diag_3"
          , "max_glu_serum", "A1Cresult", "metformin", "repaglinide", "
          nateglinide", "chlorpropamide", "glimepiride", "acetohexamide"
          , "glipizide", "glyburide", "tolbutamide", "pioglitazone", "
          rosiglitazone", "acarbose", "miglitol", "troglitazone", "
          tolazamide", "examide", "citoglipton", "insulin", "glyburide-
          metformin", "glipizide-metformin", "glimepiride-pioglitazone",
           "metformin-rosiglitazone", "metformin-pioglitazone", "change"
          , "diabetesMed", "readmitted"]
6     fields_to_omit = ["encounter_id", "patient_nbr"]
7     for field in fields_to_transform:
8         ordinal_encoder = OrdinalEncoder(handle_unknown="
              use_encoded_value", unknown_value=-1,
              encoded_missing_value=-1)
9         ordinal_encoder.fit(data.loc[:, [field]])
10        data.loc[:, [field]] = ordinal_encoder.transform(data.loc[:,
              [field]])
11        ordinal_encoders[field] = ordinal_encoder
12    for field in fields_to_omit:
13        data = data.drop(columns=[field])
14
15    tokenizing = (fields_to_omit, ordinal_encoders)
16
17    with open('model/kmeans_tokens.pkl', 'wb') as file:
18        pickle.dump(tokenizing, file)
19
20    # initialize kmeans parameters
21    kmeans_kwargs = {
22        "init": "random",
23        "n_init": 1,
24        "random_state": 1
25    }
26
27    kmeans = KMeans(n_clusters=10, **kmeans_kwargs)
28    kmeans.fit(data)
29
30    with open('model/kmeans_model.pkl', 'wb') as file:
31        pickle.dump(kmeans, file)
```

Listing A.1: Implementation of K-Means Test Model

A.2.2 Single Linkage Clustering Implementation

Listing A.2 shows the concrete implementation of the test model using SLC. The structure is the same as for the k-means model in appendix A.2.1, although no data needs to be encoded.

```
    data = pd.read_csv("resources/heart_disease_patients.csv")

    fields_to_omit = ["id"]
    for field in fields_to_omit:
        data = data.drop(columns=[field])

    tokenizing = (fields_to_omit, {})

    with open('model/slc_tokens.pkl', 'wb') as file:
        pickle.dump(tokenizing, file)

    slc = AgglomerativeClustering(n_clusters=10, linkage='single')
    slc.fit(data)

    with open('model/slc_model.pkl', 'wb') as file:
        pickle.dump(slc, file)
```

Listing A.2: Implementation of SLC Test Model

A.2.3 DBSCAN Implementation

In listing A.3 is the code implementing the test model using DBSCAN. The structure is the same as for the previous models, such as k-means in appendix A.2.1. It should be considered that the implementation used here assigns all outliers to one special cluster which is treated as a regular cluster by the metrics implemented in this thesis.

```
    data = pd.read_csv("resources/panic_disorder_dataset_training.csv
        ", nrows=1001)  # title row + 1000 rows

    ordinal_encoders = {}

    fields_to_transform = ["Gender", "Family␣History", "Personal␣
        History", "Current␣Stressors", "Symptoms", "Severity", "Impact
        ␣on␣Life", "Demographics", "Medical␣History", "Psychiatric␣
```

```
        History", "Substance Use", "Coping Mechanisms", "Social
        Support", "Lifestyle Factors"]
    fields_to_omit = ["Participant ID"]
    for field in fields_to_transform:
        ordinal_encoder = OrdinalEncoder(handle_unknown="
            use_encoded_value", unknown_value=-1,
            encoded_missing_value=-1)
        ordinal_encoder.fit(data.loc[:, [field]])
        data.loc[:, [field]] = ordinal_encoder.transform(data.loc[:,
            [field]])
        ordinal_encoders[field] = ordinal_encoder
    for field in fields_to_omit:
        data = data.drop(columns=[field])

    tokenizing = (fields_to_omit, ordinal_encoders)

    with open('model/dbscan_tokens.pkl', 'wb') as file:
        pickle.dump(tokenizing, file)

    dbscan = DBSCAN(eps=3.25, min_samples=5, metric='euclidean')
    data["cluster"] = dbscan.fit_predict(data)

    with open('model/dbscan_model.pkl', 'wb') as file:
        pickle.dump(dbscan, file)
```

Listing A.3: Implementation of DBSCAN Test Model

A.2.4 GMM Implementation

The test model using GMM is presented in the listing A.4. This model has the same structure as the previous models, such as k-means in appendix A.2.1.

```
    data = pd.read_csv("resources/breast cancer.csv")

    ordinal_encoders = {}

    fields_to_transform = ["diagnosis"]
    fields_to_omit = ["id"]
    for field in fields_to_transform:
```

```
8           ordinal_encoder = OrdinalEncoder(handle_unknown="
                use_encoded_value", unknown_value=-1,
                encoded_missing_value=-1)
9           ordinal_encoder.fit(data.loc[:, [field]])
10          data.loc[:, [field]] = ordinal_encoder.transform(data.loc[:,
                [field]])
11          ordinal_encoders[field] = ordinal_encoder
12      for field in fields_to_omit:
13          data = data.drop(columns=[field])
14
15      tokenizing = (fields_to_omit, ordinal_encoders)
16
17      with open('model/gmm_tokens.pkl', 'wb') as file:
18          pickle.dump(tokenizing, file)
19
20      gmm = GaussianMixture(n_components=10, covariance_type='full').
            fit(data)
21
22      with open('model/gmm_model.pkl', 'wb') as file:
23          pickle.dump(gmm, file)
```

Listing A.4: Implementation of GMM Test Model

A.2.5 Handmade Test Model Implementation

The test model with the self-made data is created as in listing A.5. The creation of the model has the same structure as the previous models, such as k-means in chapter A.2.1. Because of the specially prepared data, no preprocessing is needed here.

```
1       data = pd.read_csv("resources/example_good_int.csv")
2
3       tokenizing = ([], {})
4
5       with open('model/handmade_tokens.pkl', 'wb') as file:
6           pickle.dump(tokenizing, file)
7
8       # initialize kmeans parameters
9       kmeans_kwargs = {
10          "init": "random",
11          "n_init": 5,
```

```
        "random_state": 1
    }

    kmeans = KMeans(n_clusters=4, **kmeans_kwargs)
    kmeans.fit(data)

    with open('model/handmade_model.pkl', 'wb') as file:
        pickle.dump(kmeans, file)
```

Listing A.5: Implementation of the Manual Test Model

A.3 Test Model Cluster Sizes

Table A.2 shows the amount of data points per cluster for the five test models.

Table A.2: Test Model Cluster Sizes

Cluster ID	K-Means Model	SLC Model	DBSCAN Model	GMM Model	Handmade Model
0	110	289	199	198	25
1	160	3	252	26	25
2	87	1	123	47	25
3	112	4	4	4	25
4	51	1	4	99	
5	80	1	39	7	
6	69	1	5	12	
7	140	1	5	44	
8	139	1	4	40	
9	53	1	12	92	
10			4		
11			5		
12			2		
13			7		
14			4		
15			4		
16			3		
Outliers			325		

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original