Term Paper Data Science 1

Docent: Prof. Dr. Lena Wiese
Semester: Summer Term 2021

Institute of Computer Science
Goethe-Universität Frankfurt a. M.

Authors: Franziska Hicking
6673525
franziska.hicking@stud.uni-frankfurt.de
Master Bioinformatics, 2
Jonas Elpelt
6673181
elpelt@stud.uni-frankfurt.de
Master Bioinformatics, 2
Julian Rummel
6673334
s9594673@stud.uni-frankfurt.de
Master Bioinformatics, 2
Niklas Conen
6599913
conen@stud.uni-frankfurt.de
Bachelor Computer Science, 8

Date: June 20, 2021

Chosen Project Topic:
T4 - DISTANCE MEASURES AND CLUSTERING
Github Repository: `https://github.com/elpelt/datascience1_group42/`

All authors confirm that they contributed equally to the project report and were involved in the implementation and evaluation to the same extent.

Abstract

Clustering algorithms can be important tools during the analysis of datasets. They divide a dataset into groups of items based on a certain measure of similarity such as the distances between each of the items. In this work, we implemented and compared four different clustering algorithms (K-Means, K-Medoids, K-Medians, DBSCAN). For this, we selected four distinct datasets as well as multiple distance measures (Manhattan, Euclidean, Angular Cosine, Chebyshev). For efficient comparison of the clustering results we made use of multiple clustering indices. Additionally, we implemented a web frontend which provides the ability to run all clustering algorithms with distance measures, datasets and clustering indices chosen by the user. The results will be visualized afterwards. After running all algorithms with each of the datasets respectively and all distance measures where they could be applied, we compared the resulting values of the clustering indices. Finally we tried to estimate the best parameter combinations for each dataset.

Contents

1 Definition of Distance Measure

A distance measure is a function $d(x, y)$ that calculates a real value between two points in a space. If $d(x, y)$ satisfies the following three axioms the distance measure is classified as a *metric*:

$$d(x, y) = 0 \Leftrightarrow x = y \qquad \text{Identity of indiscernibles} \qquad (1a)$$

$$d(x, y) = d(y, x) \qquad \text{Symmetry} \qquad (1b)$$

$$d(x, y) \leq d(x, z) + d(z, y) \qquad \text{Triangle inequality} \qquad (1c)$$

The triangle-inequality imposes the condition that a distance reflects the shortest path between two points. Thus it is not possible to achieve a distance improvement by traveling via an intermediate point z [1].

Moreover all axioms enforce non-negative distances as an additional condition.

$$d(x, y) \geq 0 \qquad \text{Non-Negativity} \qquad (1d)$$

2 Different Distance Measurements

2.1 Manhattan Distance

To determine the distance between two items the Manhattan distance, also referred to as taxicab distance, may be used. This distance measure assumes a n-dimensional vector space with a fixed cartesian coordinate system. It is defined as following [2]:

$$d(x,y) = ||x-y||_1 = \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

where x and y are vectors

$$x = (x_1, x_2, ..., x_n) \text{ and } y = (y_1, y_2, ..., y_n)$$

The Manhattan distance is, like the Euclidean distance, part of the $L_p - metrics$ (see section 3), where the value for p is set to 1. Assuming a two dimensional space, the distance between two points is the shortest path between them with the restriction of only being able to move vertically and horizontally.
Prove of axioms described in section 1:

1. Identity of indiscernibles:
   Let $x = y$, then $|x - y| = 0$ and hence $\sum_{i=1}^{n} |x_i - y_i| = 0$
2. Symmetry:
   This axiom is fulfilled since $|x - y| = |y - x|$ for any x and y , which implies that $\sum_{i=1}^{n} |x_i - y_i| = \sum_{i=1}^{n} |y_i - x_i|$.
3. Triangle inequality:
   $|x - y| \leq |x - z| + |z - y|$ holds true for any $x, y, z \in \mathbb{R}$ This means that for $\sum_{i=1}^{n} |x_i - y_i| = \sum_{i=1}^{n} |x_i - z_i| + \sum_{i=1}^{n} |z_i - y_i|$ the equation $|x_i - y_i| \leq |x_i - z_i| + |z_i - y_i|$ is true for each i which in turn shows that the axiom is fulfilled.

4. Non-Negativity:
   This condition is trivial since the Manhattan distance is a sum of absolute values and can therefore never be negative.

2.2 Euclidean Distance

For a variable $p \in \mathbb{N}$ the *L_p-metrics* are defined as

$$d(x, y) = \sum_{i=1}^{n} (|x_i - y_i|^p)^{\frac{1}{p}} \tag{3}$$

Setting $p = 2$ expresses the Euclidean distance, which is defined as the positive square root of the sum of all squared differences in each dimension:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (|x_i - y_i|)^2} \tag{4}$$

The first two axioms defined in section 1 are easily shown to apply:

1. Identity of indiscernibles:
   For $x = y$ the value is obviously 0. Let $x = y$, then $(|x - y|)^2 = 0$ and $\sqrt{0} = 0$.
2. Symmetry:
   Symmetry is cleary given by the square of each distance.
   $(x - y)^2 = (y - x)^2$.

Non-Negativity is also shown quite easily. The square of any real number is always positive and the squareroot of any real positive number is always positive. Hence $d(x, y) \geq 0$.

The traingle inequality requires a more difficult proof. However, to keep it simple, the Euclidean space possesses the property that the sum of the lengths of Cathetus and Ancathetus is always longer than the length of the Hypothenuse [1].

2.3 Angular Cosine Distance

The Angular Cosine distance gives the (normalized) angle between two points x and y represented as vectors in an n-dimensional space. It does not make

a difference between a vector and a multiple of that vector. The Angular Cosine distance can be calculated by applying the arc-cosine function to the cosine of the angle θ between x and y [1].
It is based on the Cosine Similarity (cosine between two vectors x and y), which is definied as:

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sum_{i=1}^{n} y_i^2} \tag{5}$$

The Cosine Similarity, however, is not a distance as it is defined for positive values only. Therefore it has to be converted to the normalized angle between x and y as followed [3]:

$$\text{Angular Cosine Distance} = \frac{\arccos(\text{Cosine Similarity})}{\pi} \tag{6}$$

Note, that if x or y are zero vectors, the Cosine Similarity would not be defined. To prevent a division by zero the Cosine Similarity is set to 1 in this special case (based on the implementation of the pairwise distance in scikit-learn [4]).

The axioms for a distance measure are fulfilled for the Angular Cosine distance [1]:

1. Identity of indiscernibles:
   Two vectors can have a Angular Cosine distance of 0 if and only if they are located in the same direction. (This applies also to vectors that are multiples of one another and therefore are in the same direction.)
2. Symmetry:
   Symmetry is obviously given by the equality to measure an angle between x and y and an angle between y and x.
3. Triangle inequality:
   A rotation from x to y can be explained by a rotation from x to z and then to y. Therefore a sum of these two rotations is always bigger or equal than the rotation directly from x to y

4. No negative distances:
   Regardless of the dimensionality of the space the values of the Angular Cosine distance are between 0 and 180 degrees, therefore no negative distances can occur.

2.4 Chebyshev Distance

The Chebyshev distance (also known as Tschebyscheff distance, Maximum Value distance or L_∞ distance) is the limit of the before mentioned L_p-metrics (equation 3). On a vector space this metric is induced by the Supremum Norm (also called Chebyshev Norm or Infinity Norm), which again is the limit of the L_p-norms.
Descriptively the Chebyshev metric is the greatest difference between two vector entries. Formally it is defined as:

$$d(x, y) = \max(|x_i - y_i|) \tag{7}$$

which is the aforementioned limit of the L_p-metric and is therefore also called L_∞-metric:

$$d(x, y) = \lim_{p \to \infty} \left(\sum_{i=1}^{n} (|x_i - y_i|^p)^{\frac{1}{p}} \right) \tag{8}$$

The three axioms for a metric (section 1) are proven below:

1. For $x = y$ all entries of a vector are identical and all differences between $x_i - y_i$ are 0. Thus: $d(x, x) = \max(|x_i - x_i|) = \max(0) = 0$
2. Symmetry is given because of the symmetry of the absolute value function: $|x_i - y_i| = |y_i - x_i|$
3. The triangle equation can be shown using some estimates:

$$\begin{aligned} \max(|x_i - y_i|) &= \max(|x_i - z_i + z_i - y_i|) \\ &\leq \max(|x_i - z_i| + |z_i - y_i|) \\ &\leq \max(|x_i - z_i|) + \max(|z_i - y_i|) \\ \Rightarrow d(x, y) &\leq d(x, z) + d(z, y) \end{aligned}$$

Non-Negativity also results from the Non-Negativity of the absolute value function. Therefore the Chebyshev distance is classified as a metric.

3 Data Set Description

3.1 Housevotes

The Housevotes dataset, created by Jeff Schlimmer in April 1987, was taken from the UCI Machine Learning Repository [5]. The dataset consists of voting results of Congressmen of the U.S. House of Representatives on 16 key votes during the second session of Congress in 1984. The key votes and the voting results are identified by the Congressional Quarterly Almanac (CQA). The voting results are split into nine different types, which are consolidated into three results used in the dataset.

Voted for, paired for, and announced for count as a yes vote (y in the dataset). Voted against, paired against, and announced against count as a no vote (n in the dataset). Voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known are denoted as an unknown state (? in the dataset).

The set consists of 435 data points with 16 dimensions and can be grouped into two classes, 267 democrats and 168 republicans.

The distance based cluster algorithms described in section 4 all need numerical data in order to find clusters. Therefore the categorical data of this dataset needs to be encoded in a way that accurately describes the distance between the datapoints. The voting data has no order and all voting results should have the same distance to one another. Hence the dataset was one-hot-encoded.
With One-Hot-Encoding every attribute is represented as a binary vector. Each element of this vector represents a category value. The corresponding value of a sample is set to 1 in the binary vector. This increases the dimensionality of the problem, but represents an equal distance between every value an attribute can have.

3.2 Wine

This dataset contains the chemical analysis results of Italian wines from 3 different cultivators. It is also taken from the UCI Machine Learning Repository [5]. The dataset consists of 178 data points, each of them having 13 numeric attributes according to different measurements taken for different constitutuents (alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines, proline). Each instance belongs to either one of three classes containing 59, 71 and 48 data points. It was created by R. A. Fisher in July 1988 [4].
As it is recommended to standardise the data [5], we used a z-Scoring method (StandardScaler in sklearn [4]) to substract the attribute mean value and divide by the standard deviation per feature.

3.3 Iris

The Iris Dataset, introduced by Ronald Fisher in 1936, contains the petal and sepal measurements of three different species of Iris flowers [6]. The considered irises are Setosa, Versicolour and Virginica and each flower is represented in its own class. For each iris, 50 samples are included and for each sample the entirely numerical dataset contains the sepal length, sepal width, petal length and petal width in cm. Hence, the dataset consists of four columns where each represents one of the mentioned measurements and 150 lines, one for each individual flower. This dataset can be classified as multivariate as there are four different features for each instance.
Although is dataset is comparatively small with only 150 instances in total, it is widely known and very popular for various statistical classification methods such as machine learning and support vector machines [7]. Due to its popularity, it is not only included in the UCI Machine Learning Repository [5], but also in the base version of R [8] and in the Python scikit-learn package [4]. While one species of iris is linerally seperable from the other irises, the other two are not linerally seperable from each other [5]. This makes this dataset an interesting case for clustering.

3.4 Diabetes

The Diabetes dataset contains various information as numeric values about 442 diabetic patients, namely age, sex, body mass index, average blood pressure, and six blood serum measurements (first 10 columns), as well as a quantitative measure of disease progression one year after baseline, i.e., the response of interest (11th column). All characteristics were standardized to standard deviation times n samples and also mean centered.
This dataset is taken from the diabetes study conducted by Efron et al. [9] with the main goal of constructing a model that predicts the response (column 11) from the covariates (column 1-10).
Compared to all other datasets, there are numerical regression targets rather than categorical class values. Thus, there are no predefined clusters and thus no external cluster validation methods can be used for evaluation. Hence, the silhouette score is implemented and should be used for evaluation purposes.

4 Clustering Algorithms

4.1 K-Means

The K-Means algorithm aims to group together similar items of a given dataset into clusters. The total number of clusters is predefined and represented as the value for k. All considered items can be referred to as points, as this clustering algorithm assumes an Euclidean space. Hence, only distance measures which assume an Euclidean space, such as the Manhattan distance or the Euclidean distance, are sensibly applicable. The K-Means algorithm belongs to the point-assignment algorithms in clustering, as all points are considered successively and assigned to the most fitting cluster. The algorithm operates in the following steps [1]:

1. Initially, the algorithm picks k points whose positions each represent one cluster centroid
2. All points are considered in turn:
   - Find the nearest centroid of the considered point using the Euclidean distance measure at default (but can be exchanged for a different distance measure)
   - Assign point to cluster of that centroid
   - Adapt the position of this centroid minimizing the Euclidean distances of the centroid to all datapoints of the cluster
3. (optional) fix all centroids and reassign all points with the inclusion of the initial k points

The essential first step of initializing the clusters requires k points that have a high chance of being in separate clusters. This can be achieved by different approaches. One possible approach consists of picking points which are as far away as possible from each other. This can be achieved by conducting the following steps [1]:

1. A random point is picked as the first of k cluster centroids
2. For k-1 passes:
   Pick the point whose minimum distance is the largest considering all previously chosen points

Another way of determining the cluster centroids is the K++ initializer, described in section 5.1, which we used in our implementation.
After the K-Means algorithm assigned all points, an optional step of re-assigning the points with fixed centroids can be conducted. This may be sensible since it is possible that after a point has been assigned to a cluster the centroids move so far that the point would now belong to a different cluster.

The average runtime of the K-Means clustering algorithm is $O(ntk)$ where t is number of iterations the algorithm needs to converge [10] and n is the number of points. The K-Means clustering algorithm has many advantages including its simplicity which makes it easy to implement [11]. However, a disadvantage is that the value for k might be difficult to determine. If a specific amount of cluster is needed however, the predetermination of k is an advantage. A further disadvantage is that the starting points of the centroids greatly effect the outcome of the algorithm. This means that the algorithm might have to be run multiple times to achieve the best result.

4.2 K-Medoids

The K-Medoids clustering method is related to the above described K-Means algorithm, but uses medoids (representative points for each cluster) instead of means to define new cluster centers, which makes it more robust to outliers [12]. It partitions the dataset into k non-overlapping clusters by assigning each of the n data points to the closest of k cluster centers, which are defined by the most centrally located medoids. A medoid is a point with a minimal average dissimilarity (which is computed based on the chosen distance measure) to all other data points in the same cluster. The most commonly used algorithm to solve this NP-hard problem heuristically is the PAM (Partitoning Around Medoids) algorithm, which works in a greedy way: [13]

1. First greedily initialize the algorithm by selecting k data points to be the medoids and assigning every data point to its closest medoid. The initial data points can also be found by a K++ approach (similarly used by K-Means, see section 5.1) [10]. Note that for a possible non-deterministic initialization with random initial points the obtained clusters may differ for different runs of the algorithm.

2. Compare the average dissimilarity coefficient of a swap of each medoid m and a non-medoid data point $\bar{m}$. Find a swap between m and $\bar{m}$ that would decrease the average dissimilarity coefficient the most.

3. If no change of a medoid happened in the second step, terminate the algorithm, else reassign the data points to the new medoids and go back to step 2.

The runtime complexity of the algorithm is $O(n^2kt)$ where t is the number of iterations until convergence [10]. A disadvantage of K-Medoids is that the number of clusters has to be predefined (which is the case for all K-Algorithms).

4.3 K-Medians

The K-Medians cluster algorithm is also closely related to the K-Means algorithm, but is more robust to outliers, because it uses the median as statistics in order to determine the center of each cluster. Its main approach is to cluster data by minimizing the absolute deviations, corresponding to the Manhattan distance, between each point and its closest cluster center, i.e., creating k disjoint cluster by minimizing the following function [14].

$$Q(\{\pi_j\}_{j=1}^{K}) = \sum_{j=1}^{K} \sum_{x \in \pi_j} ||x - c_j||_1 \tag{9}$$

The geometric median is used for the minimization.

$$\underset{y \in \mathbb{R}^n}{arg\,min} \sum_{i=1}^{m} ||x_i - y||_2 \tag{10}$$

At the start of the algorithm, k cluster centers must be initialized. There are many different approaches to perform this task, such as Random Initialization, Density Analysis, Single Dimension Subsets, and many more [14]. In this work, the random approach was used because many of the other theories, while theoretically promising, are inferior or nearly equivalent in performance to the results produced by random initialization [14]. Achieving global optimization in K-Medians is known to be NP-complete [15].
The algorithm works as following [16]:

1. Assign each datapoint to a cluster, i.e. its nearest cluster center using the Manhattan distance as default (it can be substituted for a different distance measure)
2. Shift the cluster centers to the position of the vector whose elements are equal to the median value of each dimension of all instances in a cluster
3. There is no guarantee to get the perfect cluster because the starting cluster centers were initialized randomly. There is the approach of reinitializing the algorithm many times and securing the best cluster center of all iterations

The runtime of the K-medians algorithm is, similarly to the K-Means algorithm, $O(ntk)$ with n being the number of points, t being the number of iterations needed for convergence and k being the number of clusters. Due to the algorithms similarity to the K-Means algorithm, the advantages and diadvantages apply here as well.

4.4 DBSCAN

DBSCAN was developed by Martin Ester, Hans-Peter Kriegel, Jiirg Sander and Xiaowei Xu. All following definitions and descriptions are taken from their original publication [17] or their revisit of DBSCAN [18] and only apply to this algorithm.
Contrary to the aforementioned centroid-based partitioning algorithms (K-Means, K-Medoids and K-Medians) the DBSCAN (*Density Based Spatial Clustering of Applications with Noise*) algorithm uses point densities to determine clusters.
To introduce the definition of the density of a cluster, first the Eps-neighbourhood of a point is defined:

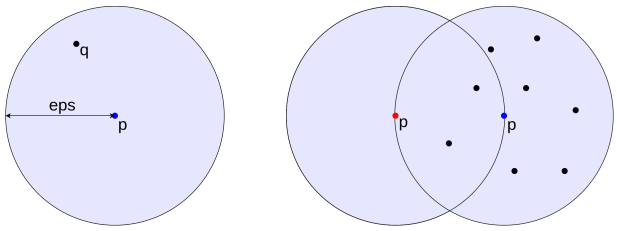
Definition 1: *Eps-neighbourhood*
A point q is part of the Eps-neighbourhood N_{Eps} of point p if the distance between them is smaller than or equal to a threshold distance called Eps.
The Eps-neighbourhood therefore is defined as $N_{Eps} = \{q \in D \mid ||p, q|| \leq Eps\}$ with D denoting the entirety of points that are supposed to be clustered and $||p, q||$ being the distance between p and q for an arbitrary distance measure.

The Eps-neighbourhood fails at being a reliable measure for the point density if a point is located at the border of a cluster. These points are called *border points*. Points that are located on the inside of a cluster are called *core points*. Hence the following definition is made:

Definition 2: *directly density-reachable and density-reachable*
A point p is directly density-reachable from a point q when

1. $p \in N_{Eps}(q)$
2. $|N_{Eps}(q)| \geq$ MinPts

with MinPts being the minimal number of points that $N_{eps}(q)$ should contain so that q is considered a core point of a cluster.
A point is *density-reachable* if there is a chain of points between p and q so that all neighbouring points in the chain are directly density-reachable.



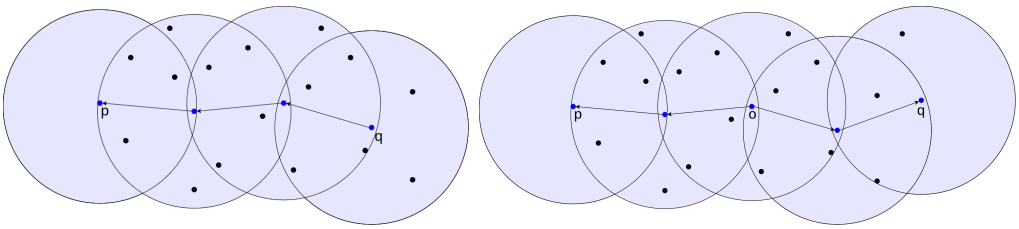
(a) eps- neighbourhood

(b) direct density-reachable

Figure 1: visualisations of the definitions for eps-neighbourhood (left) and direct density-reachable (right)

To complete the definition of what is considered part of a cluster density-connectivity is defined:

Definition 3: *density-connected*
Two points p and q are considered density-connected if there is a common point o which is density-reachable from p and q.



(a) density-reachable (b) density-connected

Figure 2: visualisations of the definitions for density-reachable (left) and density-connected (right)

Now a cluster can be described as:

Definition 4: *cluster and noise*
A cluster is a non empty subset $C \in D$ so that:

1. $\forall p, q : p \in C \wedge q$ is density reachable from $p \Rightarrow q \in C$
2. $\forall p, q \in C :$ p is density-connected to q

Noise is easily defined as every point that is not part of a Cluster C_i.

Using these definitions DBSCAN can start the clustering process with given values for Eps and MinPts. First all points are marked as not labeled. Starting from an arbitrary point p all points are iterated through in a linear fashion. For each point a `RangeQuery` function is executed finding all directly density-reachable neighbours of p. If less then MinPts neighbours are found, p is labeled as noise. Otherwise p is a core point and is labeled as part of the currently explored cluster. If this is the case the neighbourhood of p, from now on denoted as S, is expanded.

Unlabled points get checked for the core point condition (which equals a `RangeQuery` call) and all subsequent found neighbours are also added to S. Points that got labeled as noise beforehand and are part of S are labeled as part of the cluster. When the expansion comes to an end a cluster is yielded, the next unlabeled point is chosen as p and DBSCAN will continue to look for a new cluster.

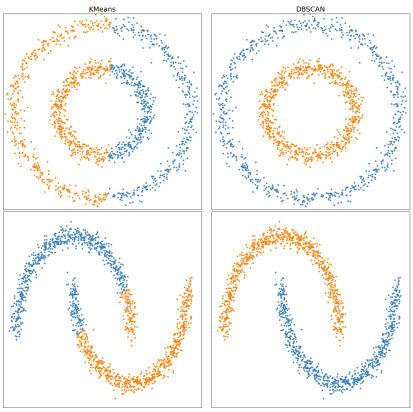


Figure 3: Comparision of DBSCAN to K-Means on spatially complex clusters

The biggest advantage of DBSCAN is its capability to find oddly shaped or spatially complex clusters. The aforementioned partitioning algorithms all fail at finding clusters that are not shaped like a sphere or a ellipsoid, if they are spatially entangled. scikit-learn published a great comparision of clustering algorithms on their website where the capabilities of DBSCAN are visualised [1]. An example generated using the python script on the aforementioned site is shown in figure 3.

Moreover, DBSCAN allows for clustering without knowing the exact number of clusters or a termination criterion. Though finding the best values for minPts and eps can be equally challenging. The heuristic described in section 5.4 tries to simplify the estimation of these parameters.

Lastly, DBSCAN will fail at finding multiple clusters if they are spatially close to each other. Because of the nature of the Eps-neighbourhood all clusters with points seperated by a distance equal or less than eps can not be distinguished.

The runtime complexity of DBSCAN heavily depends on the runtime of the `RangeQuery` function which is executed for every point in the dataset once. If `RangeQuery` is implemented using a linear scan, its complexity will be $\Theta(n \cdot D)$ with D being the time needed for calculating the distance between points. Therefore the runtime complexity of DBSCAN is $\Theta(n^2 \cdot D)$ [18].

[1] `https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html`

5 Additional Methods Used

5.1 K++ Initializer

The K++ initializer is a method for determining the positions of the initial k cluster centroids before starting the K-means algorithm. The following steps outline how the K++ initializer operates [19]:

1. Choose one centroid at random from all possible datapoints.
2. Loop through all remaining datapoints (k-1):
   determine the distance $D(x)$ of the datapoint to the nearest already chosen centroid using the Euclidean distance
3. A weighted probability distribution is used to set the next cluster centroid. For each point, the probability to be chosen is proportional to $D(x)^2$, meaning points which are farther away are more likely to be chosen
4. Steps 2 and 3 are repeated until all k cluster centroids have been set

5.2 t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction technique, which was developed by Laurens van der Maaten and Geoffrey Hinton [20]. It can be used for visualizing high-dimensional data in a lower-dimensional (typically 2-dimensional) space such that more similar data points should be represented nearby in the lower-dimensional representation. This can lead to visual cluster formation based on the local structure of the data (and chosen parameters) [21].
The t-SNE algorithm first calculates the distances $d(x_i, x_j)$ (by default using the euclidean distance) between each of the N data points x_i and x_j [22]. Then it computes conditional probabilities $p_{j|i}$, " that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i." [20]
$p_{j|i}$ for $i \neq j$ is given as

$$p_{j|i} = \frac{exp(-d(x_i, x_j)^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-d(x_i, x_k)^2/2\sigma_i^2)} \tag{11}$$

and $p_{i|i} = 0$ is set.
The joint probability $p_i j$ is defined by

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{12}$$

Note that the Gaussian distributions should have their standard deviations σ_i such that the perplexity of the conditional distribution is equal to a predefined perplexity parameter [22]. It basically measures the effective number of neighbours of the data point i, that can be found performing a binary search. In the next step t-SNE searches for an embedding of the data points considering the previously computed similarities [22]. This is achieved by minimizing the Kullback-Leibler divergence between the modeled Gaussian distributions of the high-dimensional data points X and a Student t-distribution of the corresponding points Y in the lower-dimensional space. To do this we define q_{ij} for $i \neq j$ as followed

$$q_{ji} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_t\|^2)^{-1}} \tag{13}$$

and set $q_{i|i} = 0$.
Now the Kullback-Leibler divergence can be expressed as

$$KL(P||Q) = \sum_j \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{14}$$

The optimization procedure is performed by a gradient descent method to find a local minimum [22].
The final results may heavily depend on the chosen parameters, especially the perplexity value [21]. It is therefore recommended to compare different perplexity values to identify spurious clustering artifacts in the visualization.

5.3 Principal Component Analysis (PCA)

PCA is a dimension reduction technique to increase interpretability while minimizing information loss during the process. Working with a dataset containing p numerical variables and n entities, a $n \times p$ -Matrix X gets defined with p vectors as columns. Now linear combinations (see: 15) of

the columns of X with maximum variance (see: 16) are searched for, given by:

$$\sum_{j=1}^{p} a_j x_j = Xa \tag{15}$$

$$var(Xa) = a^T Sa \tag{16}$$

with a as vector of constants $a_1, ..., a_p$ and S as sample covariance matrix associated with the dataset [23].
These linear combinations are called principal components and are p uncorrelated, new variables for the initial variables. Most of the information of the original data is compressed in the first principal component, with reduced but maximized information in the following components. It is highly important to understand the correlation between variance and information. The greater the variance, the greater the dispersion and thus the greater the abundance of information. Eigenvectors and eigenvalues are needed to calculate the principal components, where the eigenvectors of the covariance-matrix S are the directions of the axes where most of the variance is present and the corresponding eigenvalues indicate the amount of variance contained in each principal component [24]. This produces the equation [23]:

$$Sa - \lambda a = 0 \Leftrightarrow Sa = \lambda a \tag{17}$$

Thus, ranking the eigenvectors in order of their eigenvalues, one obtains all principal components from 1 to p. In the following step, the user can decide whether to keep all principal components or discard some of them based on their calculated significance, by [23]:

$$\pi_j = \frac{\lambda_j}{\sum_{j=1}^{p} \lambda_j} \tag{18}$$

This results in a matrix called *feature vector*, which contains all the remaining components as columns and forms the final dimension of the reduced dataset [24]. Usually, the requirements of graphical representation lead to keeping the first two to three principal components [23]. Finally, the original data gets reorientated from the original axes to the axes represented by the principal components.

5.4 DBSCAN Parameter Estimation Heuristic

In the original publication of the DBSCAN algorithm [17] a simple way for approximating the MinPts and Eps parameters was proposed.

The described heuristic defines a function `k-dist` which calculates the distance d of each point to its k-nearest neighbour. The d-neighbourhood of each point then contains k+1 points in most cases. Theoretically it could happen that two points share the same distance d to a point. Then the d-neighbourhood could contain more than k+1 points though this case is most unlikely.

Sorting the resulting distances in decending order and plotting them against the points results in the so called *sorted k-dist graph*. This graph gives some insights about the datasets density distribution. Choosing any point in this graph, setting MinPts to k and Eps to the k-dist value of the point, will result in DBSCAN classifying every point with the same or smaller k-dist value as a core point. These points will therefore be assigned to a cluster. Points with a larger k-dist value could be, but do not have to be, classifyied as noise.

In the sorted k-dist graph a threshold point can be searched describing the maximal distance of the thinnest cluster. The DBSCAN paper [17] proposes a visual approach on finding such threshold points by looking for a kind of bend or "valley" in the graph. In figure 4 an example is shown for such a bend found in the 4-dist graph of the iris dataset, using the euclidean distance.

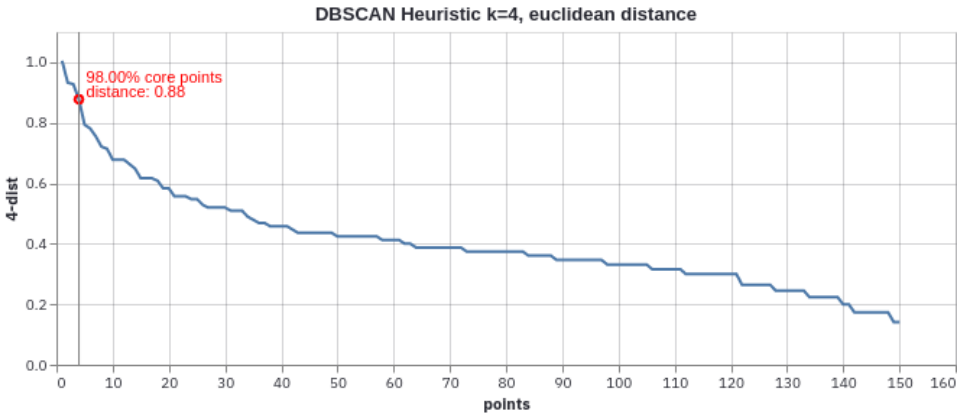


Figure 4: sorted 4-dist graph for Iris dataset and euclidean distance

While this method provides good results for some datasets, sometimes the estimated parameters do not produce a meaningful clustering or a bend may not be visible in the sorted k-dist graph. In some cases the distance chosen may result in one large cluster. Then eps must be reduced until DBSCAN can find multiple clusters, most likely resulting in more noise.

An interactive display of this heuristic with adjustable parameters for all datasets was also created using streamlit and altair and is hosted on streamlit sharing [2]. The plot in figure 4 was created using this interface.

[2] `https://share.streamlit.io/elpelt/datascience1_group42/main/code/heuristic_web.py`

6 Implementation

6.1 Description

The application follows an object oriented approach. To unify the implementation of all cluster algorithms a base class, not meant for actual instancing, was defined aggregating all functions that the cluster algorithms need. The cluster algorithms themselves are implemented as child classes inheriting all methods and attributes from the base clustering class. This allows setting different parameters for every cluster algorithm while guaranteeing the existence and uniform execution of methods used by all algorithms.

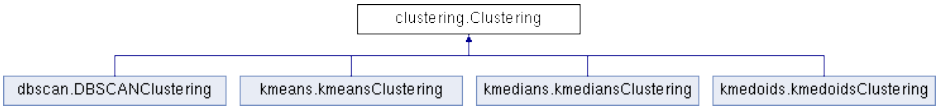


Figure 5: Inheritance diagramm for the cluster algorithm classes, taken from the Doxygen documentation

Three different libraries were used for the implementations of the used cluster algorithms. For K-Means and K-Medians the pyclustering [25], for K-Medoids the scikit-learn extra [10] and for DBSCAN the scikit learn implementations [4] were used, because of their flexibility in setting custom distance measures.

To reduce time spent on computing, a class was created for organizing already calculated results as json files. All clustering results using the option of a predefined fixed seed are, if available, loaded from a file, or are calculated and then saved. Optionally cluster results can be bulk computed using a given script (*result_calculation.py*). Because of the large amount of possible parameter combinations for DBSCAN we decided to precompute only results for the K-Algorithms with k ranging from 2 to 10 for all distance measures.

The calculation of cluster indices is packaged into an Indices class and uses scikit-learn implementations of the different scoring methods. Clustering results used for index calculations are saved using a data structure called SessionState. This structure is written by Thiago Teixeira [26] implementing a way of saving parameters per session, for example a browser tab in which the interface is opened. The variables saved in the SessionState are persistent

until a session is closed, eventhough streamlits framework is designed in a way, where every page is a sequential python script being completely reloaded when an action is triggered.

Finally the web frontend is implemented using streamlit and is hosted using their sharing service [3]. The cluster plots are generated using either seaborn or altair given a flag one can set in the frontend. The scoring results are visualised using matplotlib. The graphs which are visualised using altair are interactive in the way, that the user can freely move and zoom through the plots. By hovering over a point informations about this plot are shown in a small popup window. Where possible streamlits cacheing decorator is used, to reduce loading times when using any widget on the webpage.

Additionally, the DBSCAN heuristic described in section 5.4 was also implemented using streamlit and interactive altair charts for displaying the results. All utility functions for the heruistic were packaged into a class and are then called from the script implementing the frontend. The finished page is also hosted on streamlit sharing [4] and an URL is shown in the web frontend, when choosing DBSCAN as clustering algorithm.

All of the code base for this project is documented using the Doxygen style and an automatically generated documentation containing detailed description of every class, method and attribute can be accessed through the github project page [5].

6.2 Dependencies

Below all used libraries are described briefly.

6.2.1 pyclustering v0.10.1.2

pyclustering is a data mining library, focusing on clustering algorithms written in C++ and Python by Andrei Novikov [25]. The library contains a wide range of clustering algorithms implemented in Python with an optional C++ core. If possible pyclustering falls back to its C++ implementations utilising its efficiency and runtime benefits.

[3] `https://share.streamlit.io/elpelt/datascience1_group42/main/code/web_frontend.py`
[4] `https://share.streamlit.io/elpelt/datascience1_group42/main/code/heuristic_web.py`
[5] `https://elpelt.github.io/datascience1_group42`

6.2.2 scikit-learn v0.23.2

scikit-learn is a wide ranging data analysis toolkit for python encompassing algorithms for clustering, classification, regression and in general Data Science tools [4].
Our implementations heavily depends on scikit-learn implementations of distance measures, clustering algorithms, scoring, data preparation like standardising and projecting results.

6.2.3 scikit-learn extra v0.2.0

scikit-learn extra is an extension to scikit learn spanning algorithms that do not satisfy the inclusion criteria of scikit learn [10]. This library is used for its implementation of K-Medoids that is fully compatible with all other scikit learn algorithms.

6.2.4 numpy v1.19.2

numpy is a fundamental library mostly used for its array structure implementing a C++/Fortran like way of saving, organising and working with data while still being relatively easy to use [27]. Being a dependency of every other package used in this project we naturally use numpy arrays to store and work with our datasets.

6.2.5 matplotlib v3.3.2

matplotlib is a popular python library for generating and plotting various types of graphs [28]. matplotlib is a direct dependency of seaborn, which was used for most of our plots.

6.2.6 seaborn v0.11.0

seaborn is a powerful data visualisation library build on top the aforementioned matplotlib [29]. It simplifies plotting by providing predefined templates. The `scatterplot` function is used for the non-interactive TSNE and PCA projections of the cluster results. Moreover most of the plots contained in the Appendix are generated using seaborn.

6.2.7 pandas v1.2.4

pandas is a powerful data analysis tool [30, 31]. It provides an efficient data structure called DataFrame. This structure is used for handling clustering results, storing them in an external csv-file and preparing data for plotting.

6.2.8 streamlit v0.82.0

streamlit is an easy-to-use library for building web apps [32]. The webfrontend for our application is implemented using streamlit. Additionally, the streamlit hosting service is used for serving the webapp [6].

6.2.9 altair v4.1.0

Altair [33] is a data visualisation toolkit for python based upon the vega-lite project [34], implementing a simple way of generating interactive graphs. Plots built in altair can be easily integrated into streamlit and are therefore used when possible, to display results in an interesting fashion.

[6] `https://share.streamlit.io/elpelt/datascience1_group42/main/code/web_frontend.py`

7 Evaluation Module

7.1 Implementation

All indices are implemented as a class function for the class *Indices*. The code is part of the scikit-library [4] and expects two inputs for external cluster validation methods and three inputs for the internal cluster validation method. The input contains two arrays, namely the computed cluster labels and the expected cluster labels from the original data. For the internal index, the input is expected to be the calculated labels, the original data points, and a metric to calculate the distances (default = selected metric for cluster calculation). The output is a numerical value between 0 and 1 for Homogeneity Score and Completeness Score. For the Adjusted Rand Index (ARI) and the internal index Silhouette Score a value between -1 and 1 is calculated. Adjusted Mutual Info Score (AMI) returns a maximum value of 1, but can also drop below 0.
The user can add all scores to a comparison using a data structure called SessionState (see section 6.1). All added calculations are visualized in the form of an interactive histogram created with Altair (see section 6.2.9).

7.2 External Scoring Methods

7.2.1 Definition

Validation of clustered data is a fundamental challenge in the clustering problem. To obtain a comparable and evaluable numerical value, validity indices are usually used. There are many different indices calculating this value in different ways. For example, external cluster validation scores. External criteria are defined as the evaluation of the result with respect to a given structure, here given as a classification in the original data. Thus, an external index compares the predefined cluster labels with the set of computed labels and returns a value based on the differences and similarities respectively. In short, it is based on previous knowledge of the data [35].

All cluster algorithms can be inconsistent in naming clusters due to randomized initialisation. Thus, all indices require that a permutation of the cluster label values does not change the score value in any way, because the clustering remains the same. All four external indices explained below have this property. These scoring methods were also chosen, because they are implemented in

the scikit-learn library [4], making it easy to include all of them.

7.2.2 Adjusted Rand Index

ARI is an external cluster validation method with a value between -1 and 1, where a value close to 0 represents a random partitioning and a value close to 1 represents a nearly identical cluster compared to the original labels of the data. Thus, maximizing this score argues for perfect clustering.
It is defined by [36]:

$$ARI(P^*, P) = \frac{\sum_{i,j}\binom{N_{i,j}}{2} - [\sum_i \binom{N_i}{2} \sum_j \binom{N_j}{2}]/\binom{N}{2}}{\frac{1}{2}[\sum_i \binom{N_i}{2} + \sum_j \binom{N_j}{2}] - [\sum_i \binom{N_i}{2} \sum_j \binom{N_j}{2}]/\binom{N}{2}} \tag{19}$$

N is the number of data points in the dataset and $N_{i,j}$ describes the number of data points in a class label $C_j^* \in P^*$ associated with cluster C_i in partition P. N_i represents the number of data points in cluster C_i of partition P, while N_j represents the number of data points in class C_j^* [36].

7.2.3 Completeness Score

The Completeness Score is another external clustering index. Essentially, the Completeness Score determines how many items with the same labeling are also put into the same cluster. It is symmetric to the Homogeneity Score and is defined as following [37]:

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{20}$$

where

$$H(C|K) = -\sum_{c,k} \frac{n_{ck}}{N} \log\left(\frac{n_{ck}}{n_c}\right) \tag{21}$$

Consistent to the Homogeneity Score, C represents the true cluster labels of the datapoints and K the labels predicted by the clustering algorithm. n_{ck} is the number of items in a cluster k, which are labeled c, i.e. share the same label. n_c stands for the total number of points labeled c. When all items labeled c are put into one single cluster k, the Completeness Score is 1.

7.2.4 Homogeneity Score

The Homogeneity Score is an external clustering index to determine the quality of a calculated clustering in comparison to a preexisting grouping of items. It is defined as following [37]:

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{22}$$

where

$$H(C|K) = -\sum_{c,k} \frac{n_{ck}}{N} \log\left(\frac{n_{ck}}{n_k}\right) \tag{23}$$

Here, C represents the true cluster labels of the datapoints and K represents the labels predicted by the clustering algorithm. n_{ck} is the number of items in a cluster k, which are labeled c, i.e. share the same label. n_k stands for the total number of labels present in cluster k. When each cluster k contains only items with the same label, the Homogeneity Score equals one.

7.2.5 Adjusted Mutual Index

The AMI is an external cluster validation method, in order to calculate the information content of a partitioning computed by a clustering algorithm. It is an adjustment of the Mutual Information (MI) to account for chance [4]. This scores upper boundary is 1 and is expected to be 0 for random partitions, but can also be negative. The score is calculated as following [38]:

$$\begin{aligned} AMI_{max}(U,V) &= \frac{NMI_{max}(U,V) - E\{NMI_{max}(U,V)\}}{1 - E\{NMI_{max}(U,V)\}} \\ &= \frac{I(U,V) - E\{I(U,V)\}}{max\{H(U),H(V)\} - E\{I(U,V)\}} \end{aligned} \tag{24}$$

NMI stands for Normalized MI, which is the MI normalized to values in the range of 0 to 1 [4]. $E\{I(U,V)\}$ is the calculated expected index and $I(U,V)$ the actual Index [38].

7.3 Internal Scoring Methods

7.3.1 Definition

In addition to external cluster valdiation indices, there are also internal indices. These indices are based on internal criteria, defined as the evaluation of the result in terms of information inherent in the data alone. This is needed for datasets where no predefined classification is given, such as the Diabetes dataset. Compared to external indices, internal indices are known to be more accurate in finding groups in a given clustering structure, but show poorer time complexity [35].

7.3.2 Silhouette Score

The Silhouette Score is an internal cluster validation method that calculates the mean Silhouette coefficient (SC) of all samples [4]. It calculates a separation distance between resulting clusters, by stating how close each point in one cluster is to the points in a neighboring cluster [39]. The score is calculated as following for each sample [4]:

$$SC = \frac{b - a}{max(a, b)} \tag{25}$$

b is defined as the distance between the distance between a sample and the nearest cluster to which the sample does not belong, and a as the mean intra-cluster distance.
The SC can be calculated either for each cluster or the entire dataset. To obtain a comparable value, in this project the score for the entire dataset is calculated as the mean value over all samples [4].
This index is necessary to compare the different results for the clustering of the Diabetes dataset, since this dataset does not contain a predefined classification.

8 Web Frontend and User Manual

The web frontend is designed to provide an intuitive exploration of the different datasets, clustering algorithms and distances. In an interactive interface multiple clustering settings can be chosen and a visualisation of the results is directly generated. A cluster-table is used to store previous calculated results, which can be plotted within the evaluation module.
The checkbox “Use precalculated results (with random seed for reproduction)” at the top of the page (see Figure 6 A), which is set by default, allows the use of precalculated clustering results, which have been computed beforehand (with a random seed value) and are stored in the github repository. This was done for reproduction and runtime purposes. The second checkbox ”use interactive charts” (see Figure 6 B), also set by default, allows the option for interactive projection plots for PCA, t-SNE, and the evaluation module.
The user can choose between four datasets (see Figure 6 C), four distance measures (see Figure 6 E) and four different algorithms (see Figure 6 D) via a drop-down menu. If one of the K-Algorithms is chosen a value for the parameter k between 1 and 10 has to be set (see Figure 6 F). The default value for k is 3. If DBSCAN is chosen the user has to define a value for Eps and MinPts. (see Figure 7 A). Additionally, a link to a webpage implementing the DBSCAN heuristic helping with estimating the values for epsilon and minPts is shown. A short manual for this page is given in section 8.1. The parameter settings can be adjusted with an interactive slider widget.

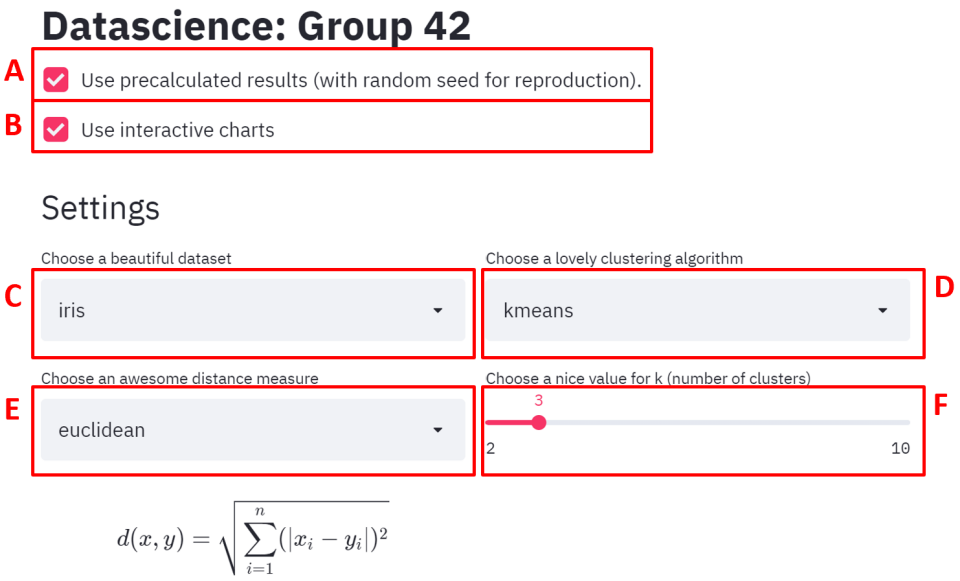


Figure 6: First part of the web frontend. Setting options for clustering parameters for K-Means, K-Medians and K-Medoids.

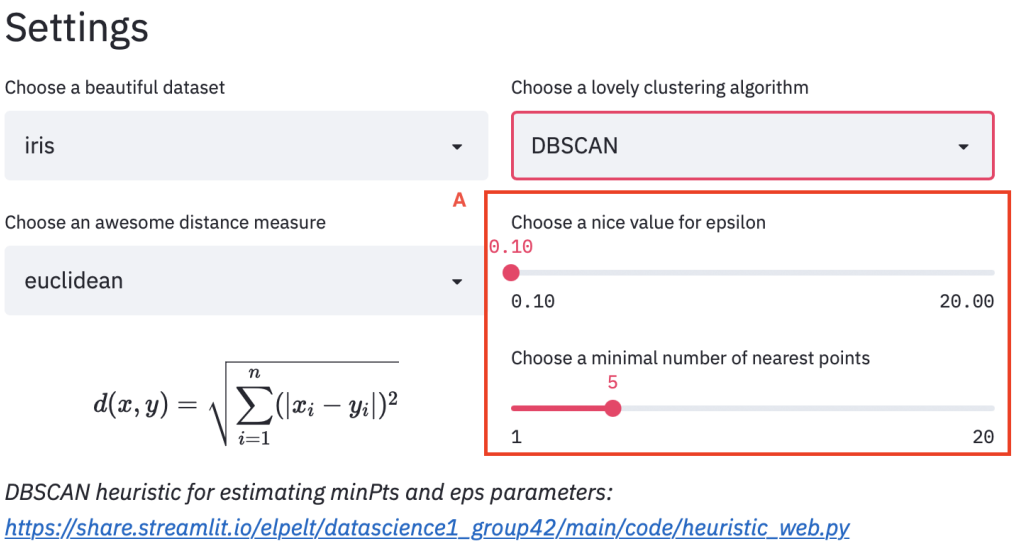


Figure 7: Epsilon and minimal number of nearest points setting options for DBSCAN.

For every dataset some main information can be retrieved via an expander. The dataset dimension (see Figure 8 A), pre classified cluster of the data (not for diabetes dataset) (see Figure 8 B), datatypes per column (see Figure 8 C), dataset preview (see Figure 8 D), mean per column (see Figure 8 E), and performed changes on the dataset are accessible (if performed).

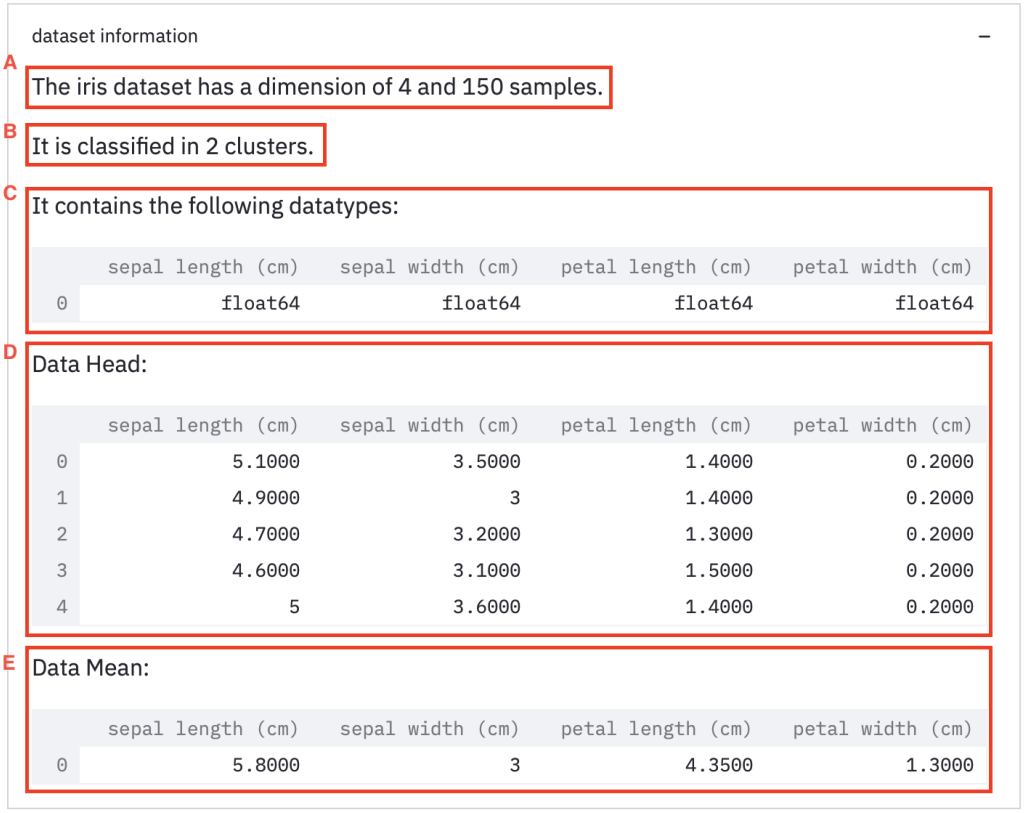


Figure 8: Displayed dataset information by expanding the box by clicking on the plus. Iris dataset as example.

Moreover the perplexity value for the t-SNE projection can be set individually between 5 and 50 with a slider (see Figure 9 A). The default value is 25. The t-SNE and PCA plots are shown next to each other to allow a direct comparison of the lower-dimensional projections. For K-Medoids the medoid, for K-Means the mean, and for K-Medians the median for each cluster is marked as diamond (see Figure 9 D). For the PCA plot both axis are labeled with the percentage of the first two components. To virtually interact with the plots, the button shown in Figure 9 C can be clicked. Please note that this option is only available when the checkmark shown in Figure 6 B is set. Furthermore, this button provides the ability to save the plot in different formats. The calculated runtime is displayed right under the plots (see Figure 9 B). This info is only accessible if precalculated results are not used

(see Figure 6 A). The frontend is reloaded entirely if a parameter value is changed, a different setting is made or a button is clicked.

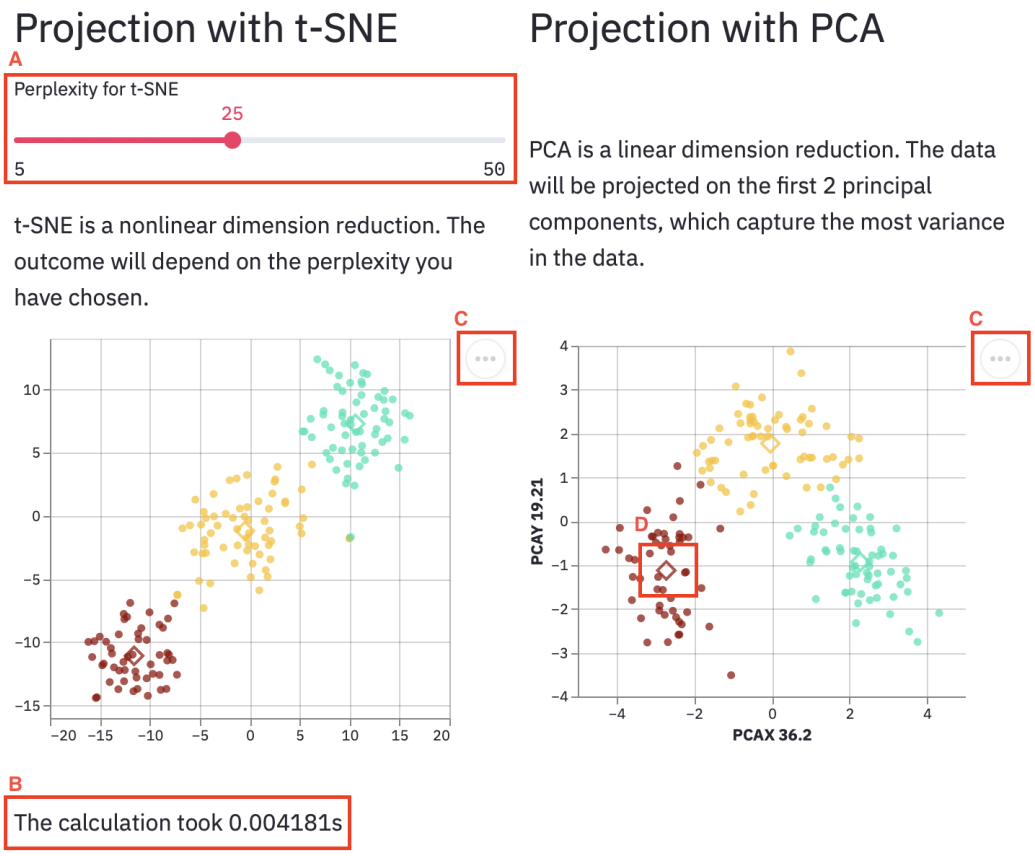
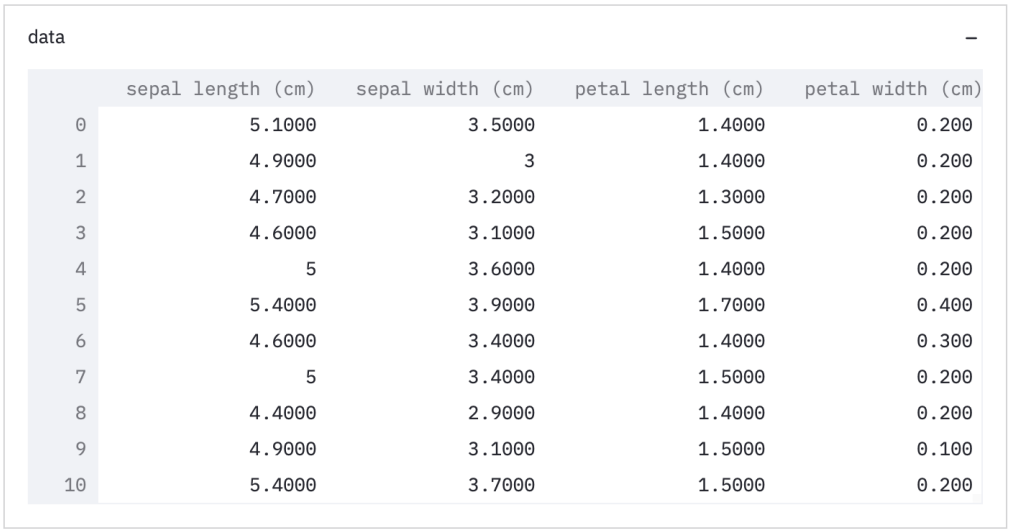


Figure 9: Second part of the web frontend. Projection results of a clustering.

The selected dataset can be viewed in a table format below the plots (see Figure 10).


data

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1000	3.5000	1.4000	0.200
1	4.9000	3	1.4000	0.200
2	4.7000	3.2000	1.3000	0.200
3	4.6000	3.1000	1.5000	0.200
4	5	3.6000	1.4000	0.200
5	5.4000	3.9000	1.7000	0.400
6	4.6000	3.4000	1.4000	0.300
7	5	3.4000	1.5000	0.200
8	4.4000	2.9000	1.4000	0.200
9	4.9000	3.1000	1.5000	0.100
10	5.4000	3.7000	1.5000	0.200

Figure 10: Projection of dataset as expander. Iris dataset as example.

All clustering results can be saved in a cluster table for comparison via the *Add*-button button (see Figure 11 A). To compare this result to clusterings with other settings, the *Add*-button can be clicked repeatedly after desired adjustment of the clustering settings. To start over and compare further clustering indices, the *Reset*-button (see Figure 11 B) can be clicked. The previous display of resulting indices will be cleared as well as the clustering table. For the evaluation of the clustering results, one of five clustering indices can be chosen via a drop-down menu as shown in Figure 11 C. An interactive barplot is displayed immediately, after adding a result to the cluster-table (see Figure 12). A maximum reference value (1) is also always displayed (see Figure 12 A).

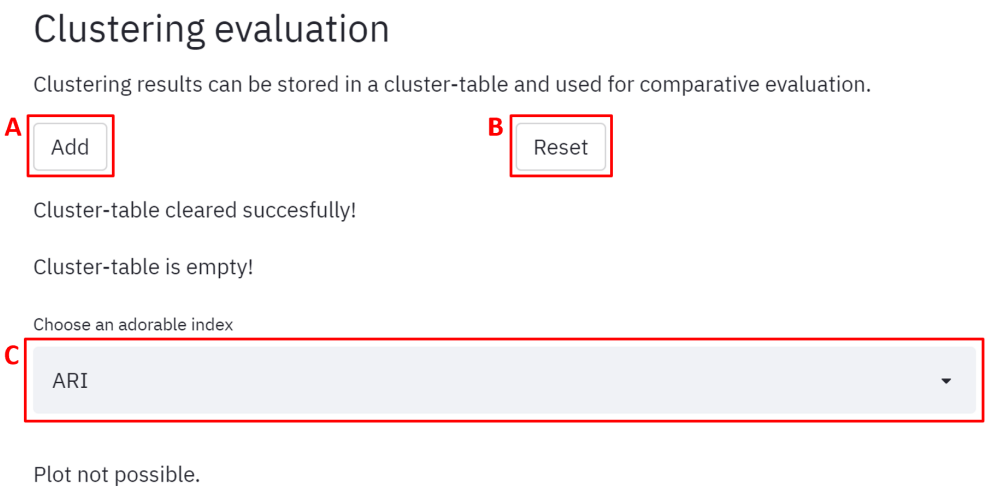


Figure 11: Third part of the web frontend. Evaluation of clustering results.

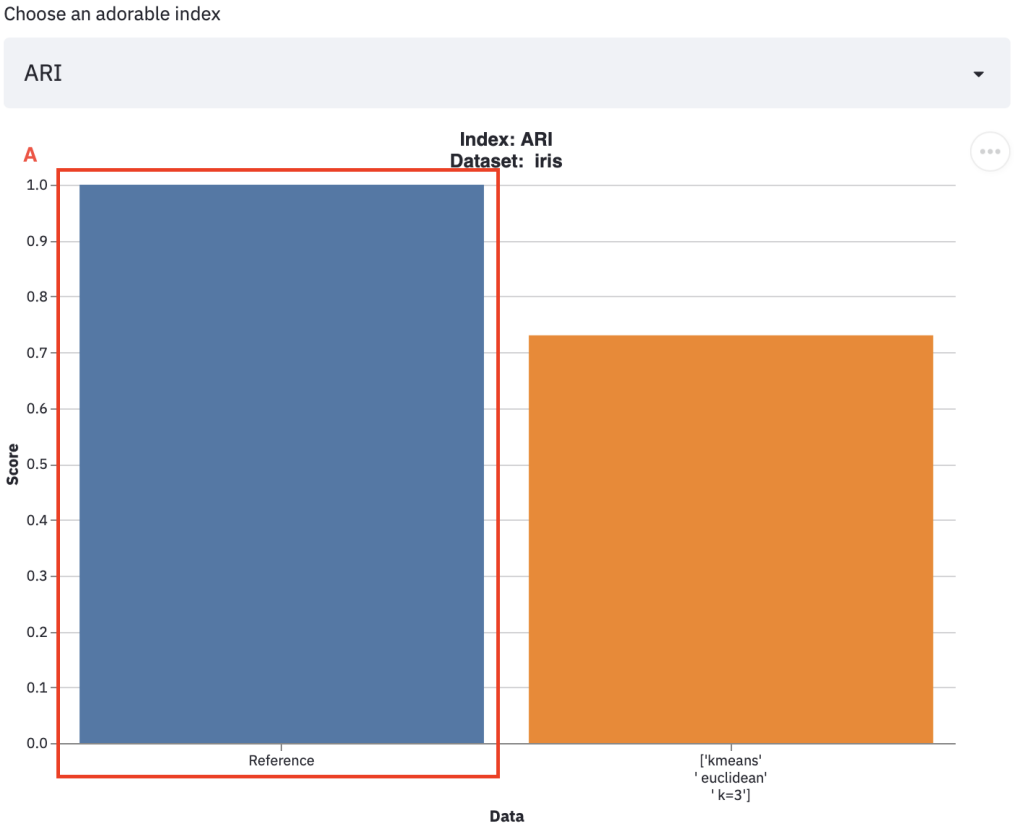


Figure 12: Barplot for evaluation comparison. Kmeans, euclidean, k=3, iris, and ARI as example.

Ancillary streamlit settings can be found at the upper right corner. Additional explanatory texts provide help and more information.

8.1 DBSCAN Heuristic Page

The webpage for calculating the sorted k-dist graph for the DBSCAN heuristic is build similarly to the main apps page described above. A simple form is presented were dataset, distance measure and the k value can be chosen using drop-down menus or sliders (see Figure 11 A, B, C). (Note: k in this case is not the number of clusters. k stands for the k-nearest neighbour of any point)

Only when clicking the *Calculate kdist Graph*-button (see Figure 11 D) the sorted k-dist graph is calculated and plotted (see Figure 11 E).

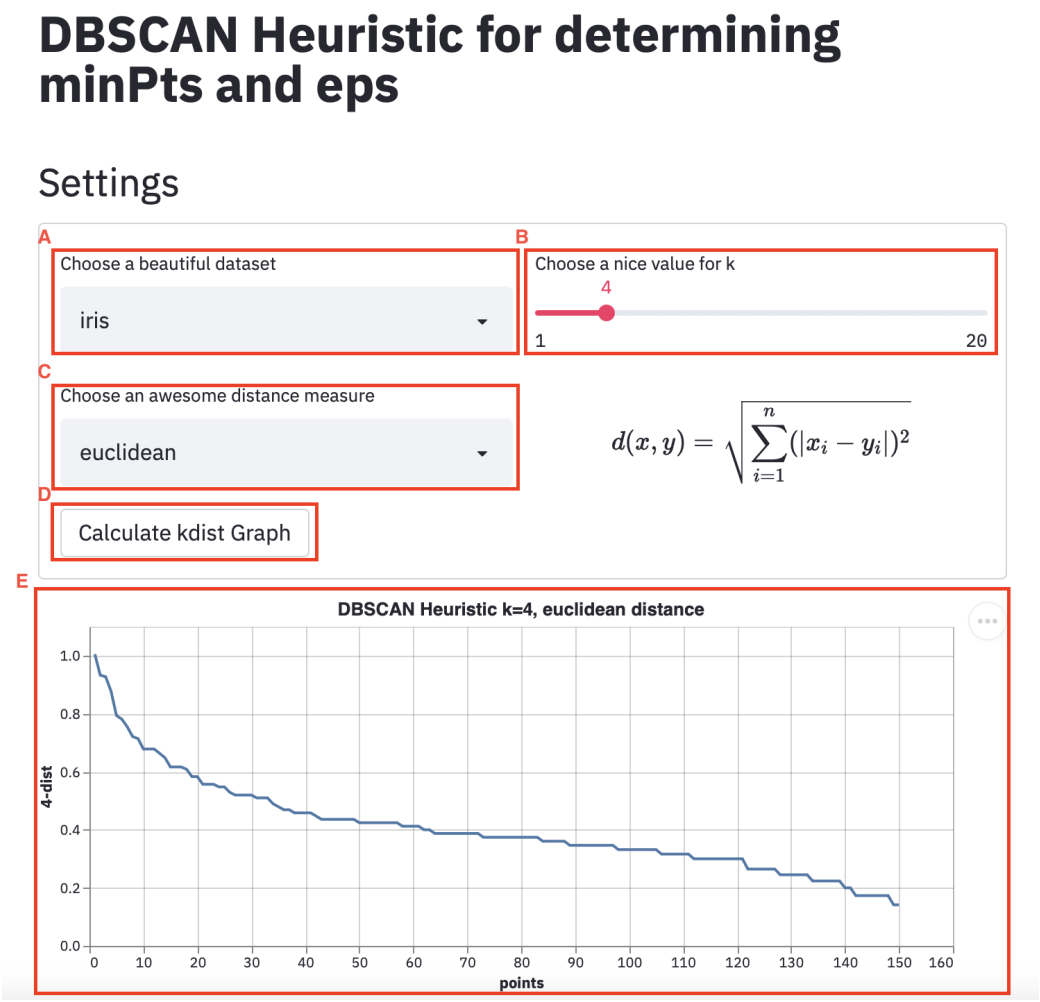


Figure 13: DBSCAN heuristic settings

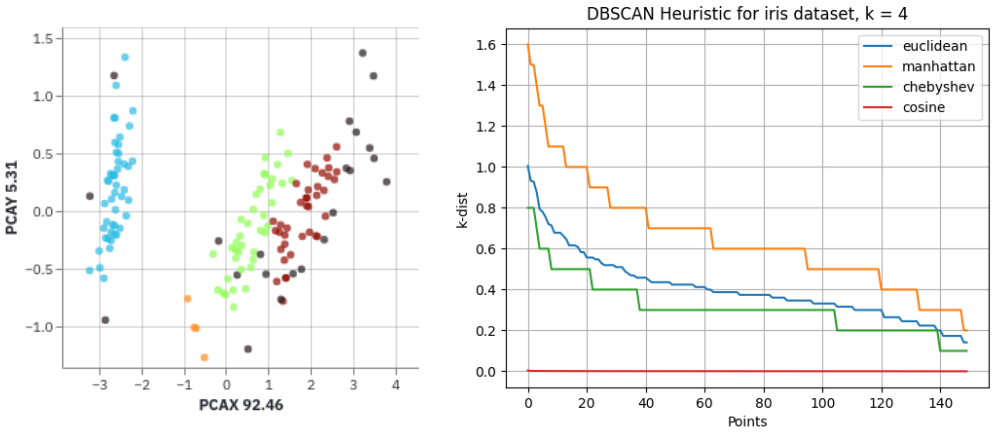
9 Conclusion

For Wine and Housevotes datasets we found that a clustering with the Chebyshev distance produces clusterings with consideribly low scores for all evaluated indices. For Iris and Wine datasets ARI scores for the used K-

Algorithms have a peak for k=3 (Figure 31, Figure 32), which corresponds to the true number of labels in these datasets. By comparing different clustering algorithms for the Wine dataset Figure 32 we found that the K-Medians algorithm is more unstable for this specific dataset than K-Means and K-Medoids as small peturbations of k can lead to remarkable differences in the resulting clustering scores.

In our test cases the Angular Cosine distance always took the longest computation time for all clustering algorithms and datasets. The Euclidean distance had longer runtimes than Manhattan distance and Chebyshev distance. This is reasonable as for the Angular Cosine distance three sums and a square root need to be computed. The runtime for Euclidean distance is higher than for Manhattan or Chebyshev as a computation of an additional square root is necessary.
The fastest algorithm is DBSCAN, whereas K-Medoids takes the longest time to compute for all datasets besides Iris (where K-Medians with Angular Cosine distance took extraordinary long). The runtime increases with the number of samples n in the dataset.

For the Iris dataset a clustering calculated by a K-Algorithm (with k=3, Figure 31) with the Angular Cosine distance gives the highest score values for all evaluated external scores. Therefore, the resulting clustering labels calculated with the Angular Cosine distance evidentely match the true labels in the most accurate way in comparison with all other considered distance measures. Comparing the performances of all K-Algorithms we found that K-Medians and K-Means give slightly higher scores than K-Mediods. The results are very similar and differ only in a few differently assigned data points. A parameter combination for DBSCAN was found, where parts of the three clusters known (eps=0.4, minpts = 3, see Figure 14) were indentified. However a large amount of noise had to be considered, making DBSCAN not viable for clustering this dataset. The 4-dist graph (see figure Figure 14) also shows no visible valleys, as explained in section 5.4, making it quite difficult to find a value for eps, where clusters can be clearly distinguished without risking a large amount of noise.

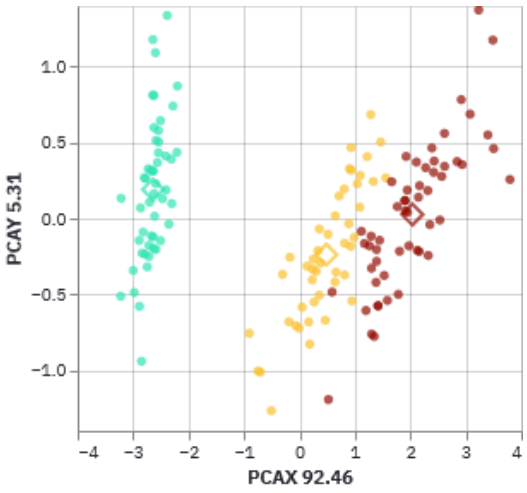


(a) DBSCAN, Euclidean Distance, eps=0.4, minPts=3

(b) 4-dist graph of the iris dataset for all distances

Figure 14: DBSCAN result and 4-dist graph for the iris dataset

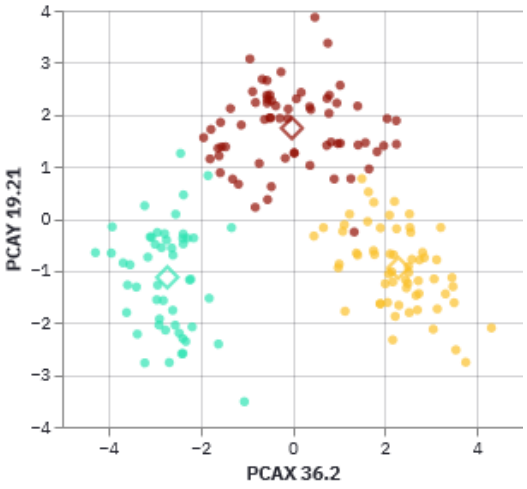
Finally, we would recommend to use the combination of the K-Means algorithm, the Angular Cosine distance and a value k=3 for the Iris dataset (see Figure 15). While the results of the K-Medians and the K-Means algorithm are the exact same, the runtime of K-Means is better by a factor of around 17 in comparison to the K-Medians algorithm, making it the preferred choice in this case.



(a) K-Means, Angular Cosine Distance, k=3

Figure 15: PCA projections of clustering results with preferred parameters for Diabetes dataset

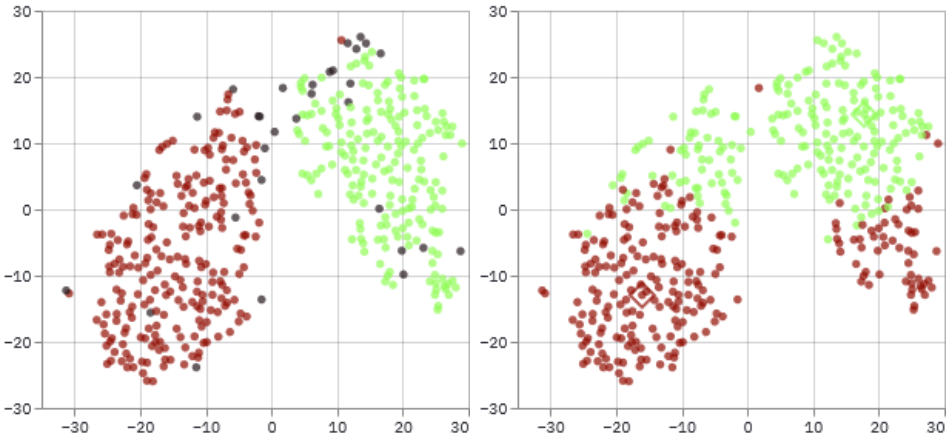
Using the Chebyshev distance for the Wine dataset results in clusterings with the lowest scores compared to all other distance measures. In contrast, K-Means with the Euclidean distance and K-Medians with the Manhattan distance perform relatively well for this dataset (Figure 32). We were not able to produce a result similar to the ones generated by the K-Algorithms using DBSCAN. Therefore, we suggest to use the K-Means algorithm (k=3) together with the Euclidean distance for the Wine dataset (see Figure 32).



(a) K-Means, Euclidean Distance, k=3

Figure 16: PCA projections of clustering results with preferred parameters for Wine dataset

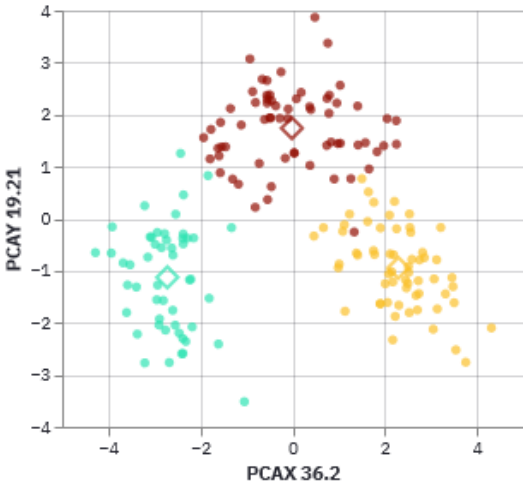
For the Diabetes dataset the K-Medians algorithm with k=2 and the Manhattan distance performs the worst considering the Silhouette Score (Figure 33). The K-means algorithm along with the Euclidean, Manhattan or Angular Cosine distance, as well as the results of the K-Medians or K-medoids algorithm with Angular Cosine distance, give the highest scoring results. A visual inspection of the t-SNE projected data points leads to an assumption of two main clusters in the data. Those clusters are well captured by a DBSCAN-Clustering with $eps = 0.1$ and $minPts = 4$, but not with a K-Algorithms clustering (Figure 17). Therefore we would suggest to use a DBSCAN clustering approach for this dataset.



(a) DBSCAN ($eps = 0.1$ and $minPts = 4$)(b) K-Means ($k = 2$, Euclidean distance)

Figure 17: t-SNE projections of clustering results with DBSCAN and K-Means for Diabetes dataset

For the Housevotes dataset (Figure 34) the Chebyshev distance performs the worst. The highest external scores for k=2 can be achieved with K-Means or K-Medians using the Euclidean, Manhattan or Cosine distance. Similarly to the Iris dataset, DBSCAN was not able to distinguish the two known groups without labeling a large amount of samples as noise. We suggest to select the K-Means algorithm (k=2) and in combination with the Euclidean distance (see Figure 18) for the Housevote dataset. Looking at the ARI Score, this combination produces the highest score. Although for all other scores the combination of K-Medians and Cosine distance is slightly better, the difference is minimal (changes only in the second or third percentile). Additionally, the runtime for K-Means in this case is over 40 times faster than K-Medians, undermining the choice for this algorithm.



(a) K-Means, Euclidean Distance, k=2

Figure 18: t-SNE projections of clustering results with preferred parameters for Housevotes dataset

In general, our study is restricted to a small set of algorithms and distance measures. As a result, it can only provide a limited overview. A variety of additional algorithms, distance metrics, parameter settings and cluster evaluation measures could be considered to get a more exhaustive in-depth analysis. By including more datasets a more detailed comparison of the algorithms and distance measures and their behaviour on differently distributed data might be possible. Furthermore, even more clustering indices, particularly internal indices, could give further insights on the clustering quality.

Finally, we did not find the best clustering algorithm or the superior distance measure which would give the best results on every dataset. This general problem in the field of Data Science is widely known as the "No Free Lunch Theorem" [40]. As specific requierements may differ for diverse use cases multiple algorithms and parameters should be explored and compared according to individual needs.

Glossary

K-Algorithms

One of the studied clustering algorithms, that work with a parameter k: K-Means, K-Medians, K-Medoids.

One-Hot-Encoding

Encoding method for categorical attribute values where every attribute is represented as a binary vector and each element in the vector represents a category..

RangeQuery

Function to find all points in a Eps-Neighbourhood around a given point..

Acronyms

AMI	Adjusted Mutual Info Score.
ARI	Adjusted Rand Index.
MI	Mutual Information.
MinPts	Minimum number of points to be considered in Eps-neighourhood for DBSCAN.
PCA	Principal Component Analysis.
SC	Silhouette coefficient.
t-SNE	t-Distributed Stochastic Neighbor Embedding.

References

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. Definition of a distance measure. In *Mining of Massive Datasets - THIRD EDITION*, page 97. Infolab Stanford EDU, 2020.

[2] Susan Craw. *Manhattan Distance*, pages 639–639. Springer US, Boston, MA, 2010.

[3] Cosine distance, cosine similarity, angular cosine distance, angular cosine similarity, Jul 2017.

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

[5] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017.

[6] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[7] Marten Schutten and Marco Wiering. An analysis on better testing than training performances on the iris dataset. 11 2016.

[8] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020.

[9] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32:407–499, 2004.

[10] scikit-learn-extra. `https://github.com/scikit-learn-contrib/scikit-learn-extra`, 2021.

[11] Marina Santini. Advantages & disadvantages of k-means and hierarchical clustering (unsupervised learning). *URL: http://santini.se/teaching/ml/2016/Lect_10/10c_Unsupervise dMethods. pdf (Accesed 17.04. 2019)*, 2016.

[12] Xin Jin and Jiawei Han. *K-Medoids Clustering*, pages 564–565. Springer US, Boston, MA, 2010.

[13] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[14] Christopher Whelan, Greg Harrell, and Jin Wang. Understanding the k-medians problem. *Int'l Conf. Scientific Computing*, pages 219–222, 2015.

[15] Sirion Vittayakorn, Sanpawat Kantabutra, and Chularat Tanprasert. The parallel complexities of the k-medians related problems. *IEEE Xplore*, 2008.

[16] K-means and k-medians, 2020.

[17] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 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*, KDD'96, page 226–231. AAAI Press, 1996.

[18] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.*, 42(3), July 2017.

[19] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[20] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[21] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016.

[22] Mathworks documentation, t-sne, 2021.

[23] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *The Royal Societey Publishing*, April 2016.

[24] Zakaria Jaadi. A step-by-step explanation of principal component analysis (pca), April 2021.

[25] Andrei Novikov. PyClustering: Data Mining Library. *Journal of Open Source Software*, 4(36):1230, apr 2019.

[26] Thiago Teixeira. gist: Sessionstate.py. `https://gist.github.com/tvst/036da038ab3e999a64497f42de966a92`, 2021. Accessed: 10th June 2021.

[27] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[28] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[29] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[30] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[31] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[32] streamlit. `https://github.com/streamlit/streamlit`, 2021.

[33] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: Interactive statistical visualizations for python. *Journal of Open Source Software*, 3(32):1057, 2018.

[34] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2017.

[35] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M. Quiroz. Internal versus external cluster validation indexes. *International Journal of Computers and Communications*, 5, 2011.

[36] Yun Yang. *Temporal Data Mining Via Unsupervised Ensemble Learning*. Elsevier, 2017.

[37] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007.

[38] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 2010.

[39] Abhishek Mishra. Understanding silhouette scoring. `https://datascience.foundation/datatalk/understanding-silhouette-scoring`, 2020. Accessed: 16th June 2021.

[40] David Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8, 03 1996.

[41] Moshe Lichman. UCI Machine Learning Repository, 2013.

[42] Harro Heuser. *Lehrbuch der Analysis : Teil 2*. Mathematische Leitfäden. B.G. Teubner, Stuttgart, 11. auflage edition, 2000.

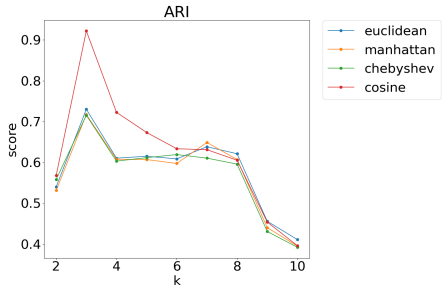
[43] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

Appendices

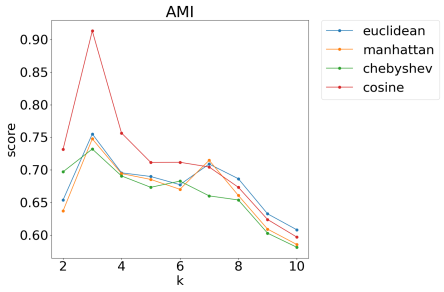
A Plots

A.1 Comparison of different k values

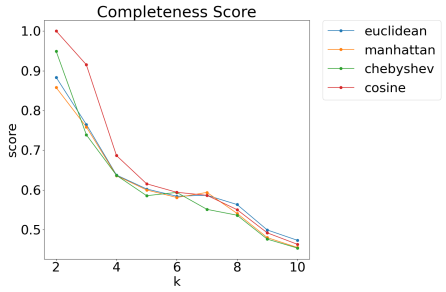
To estimate the number of meaningful clusters (clusterings with highest index scores) in the datasets and test the robustness of different distance measures we compared scorings as a function of k (for K-Means, K-Medians and K-Medoids). The diabetes dataset was only evaluated with an internal index (silhouette score) as it does not have categorical target values, which would not lead to meaningful scores of external indices.
As the silhouette score can be evaluated with different distance metrics, we have included the four distance measures to be considered independently for each distance measure used for the clustering algorithm.



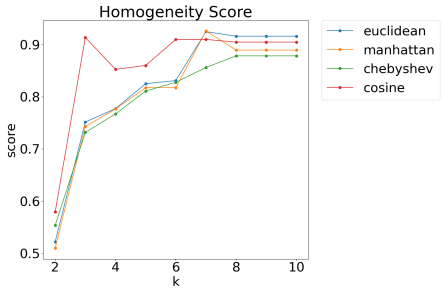
(a) ARI



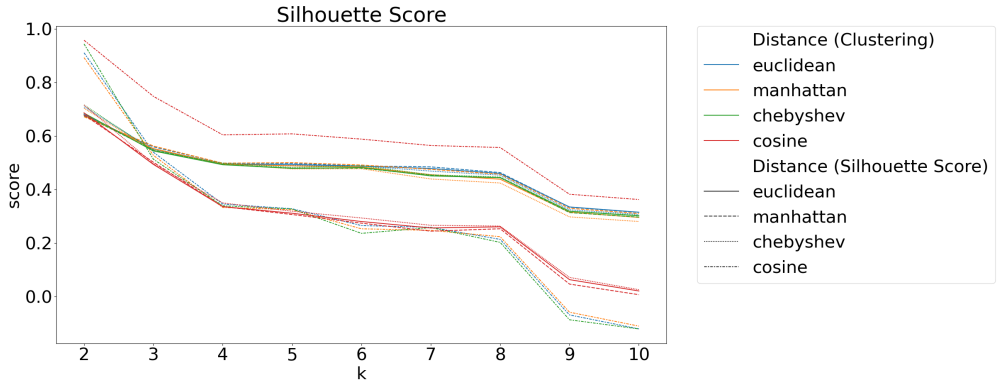
(b) AMI



(c) Completeness Score

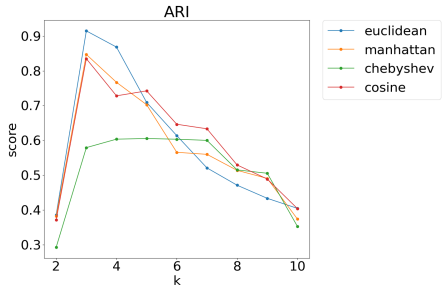


(d) Homogeneity Score

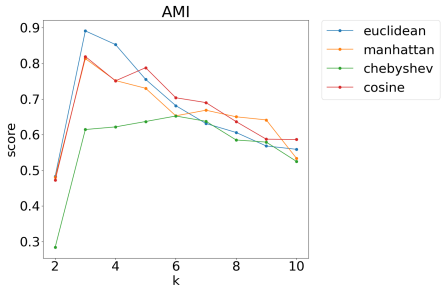


(e) Silhouette Score

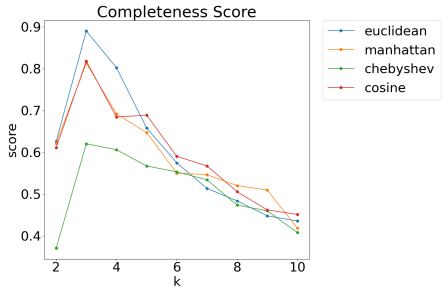
Figure 19: Comparison of clustering scores for K-means-clustering on Iris dataset



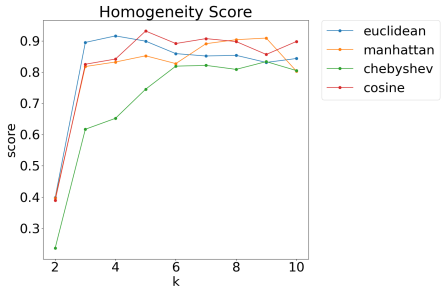
(a) ARI



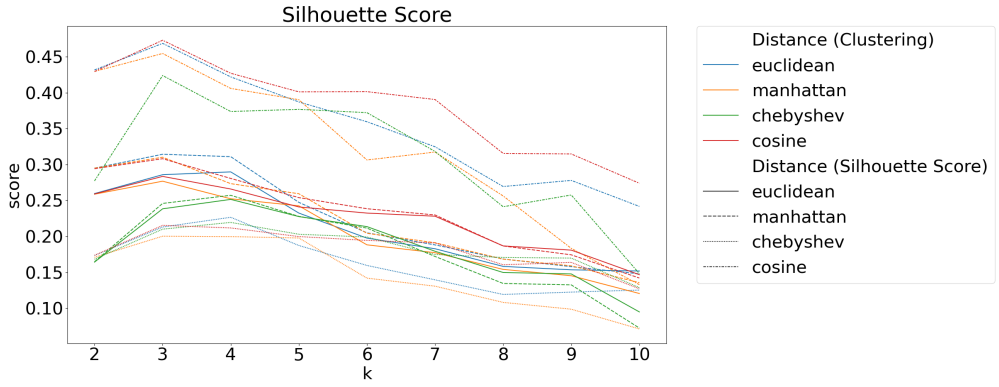
(b) AMI



(c) Completeness Score

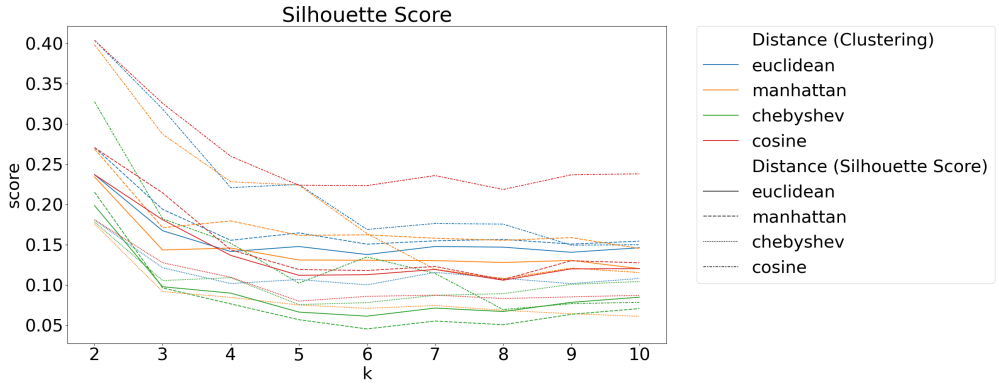


(d) Homogeneity Score



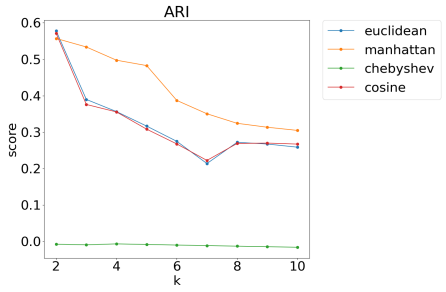
(e) Silhouette Score

Figure 20: Comparison of clustering scores for K-means-clustering on Wine dataset

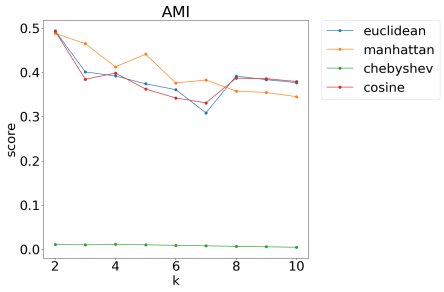


(a) Silhouette Score

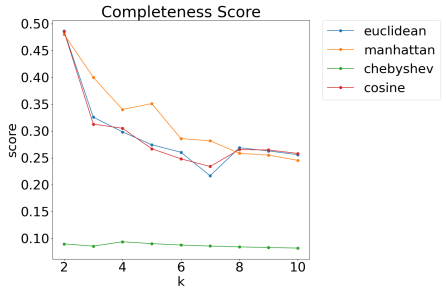
Figure 21: Comparison of clustering scores for K-means-clustering on Diabetes dataset



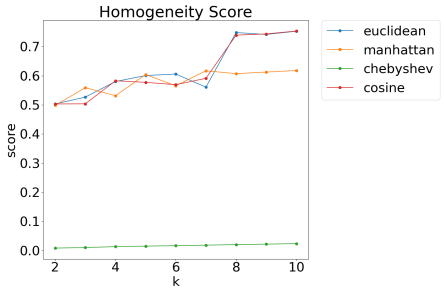
(a) ARI



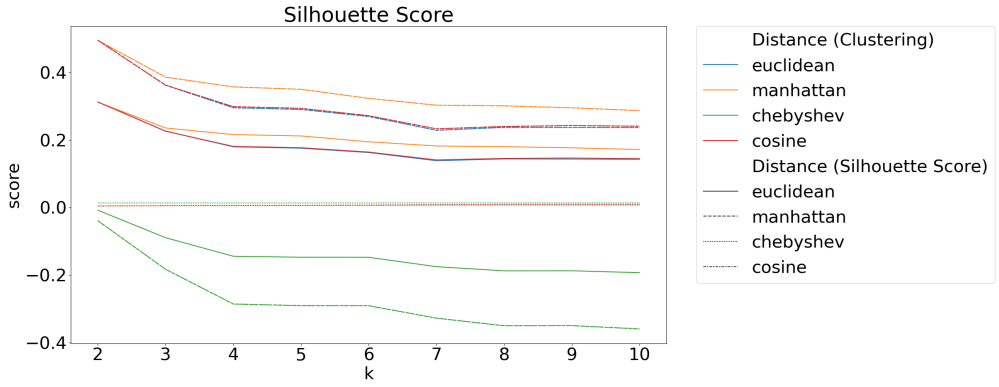
(b) AMI



(c) Completeness Score

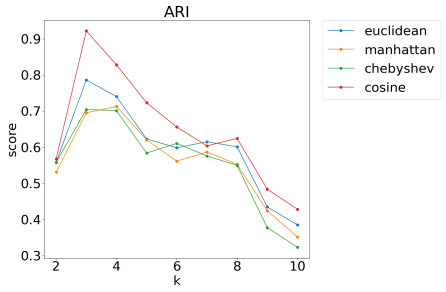


(d) Homogeneity Score

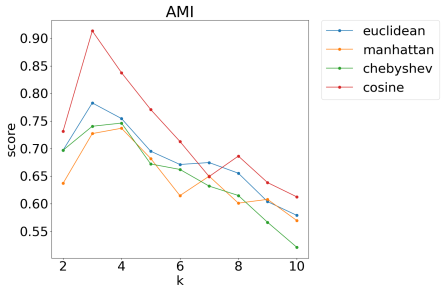


(e) Silhouette Score

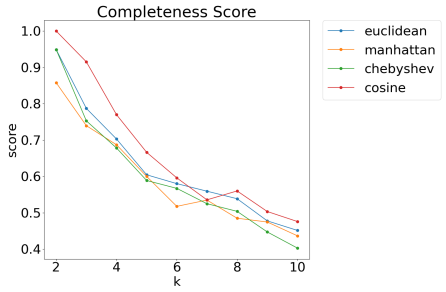
Figure 22: Comparison of clustering scores for K-means-clustering on House-votes dataset



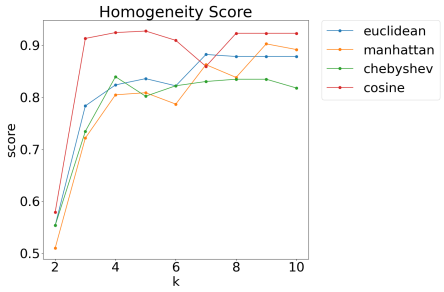
(a) ARI



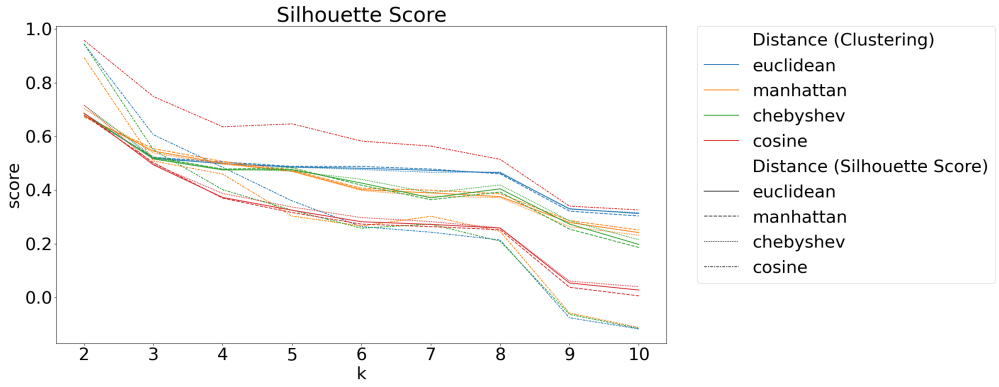
(b) AMI



(c) Completeness Score

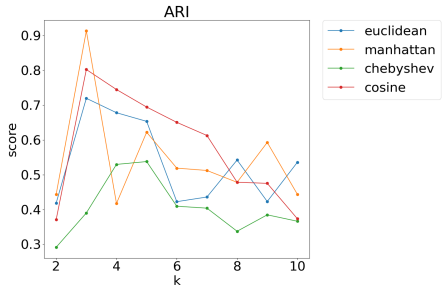


(d) Homogeneity Score

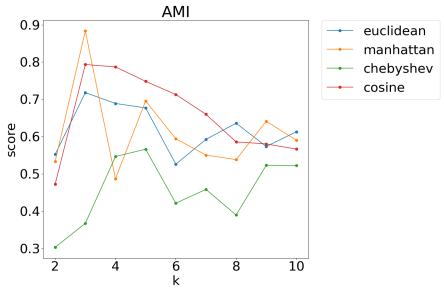


(e) Silhouette Score

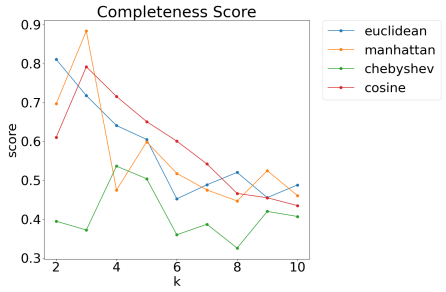
Figure 23: Comparison of clustering scores for K-medians-clustering on Iris dataset



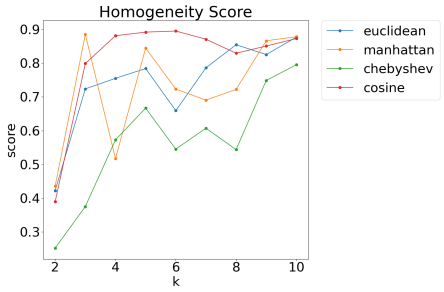
(a) ARI



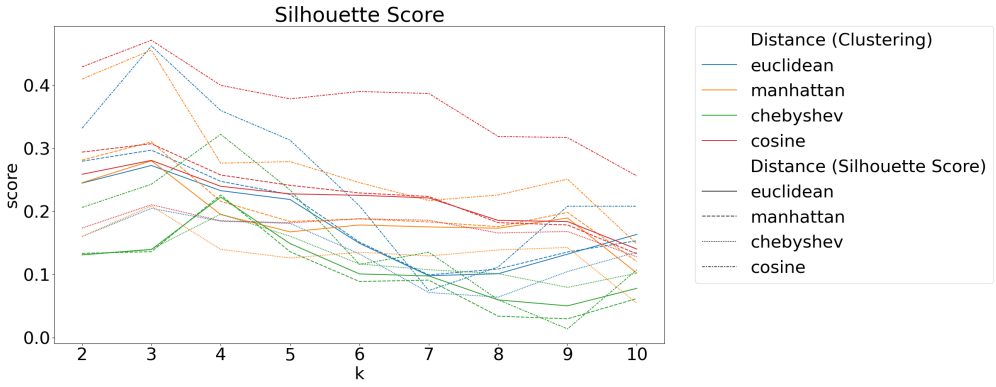
(b) AMI



(c) Completeness Score

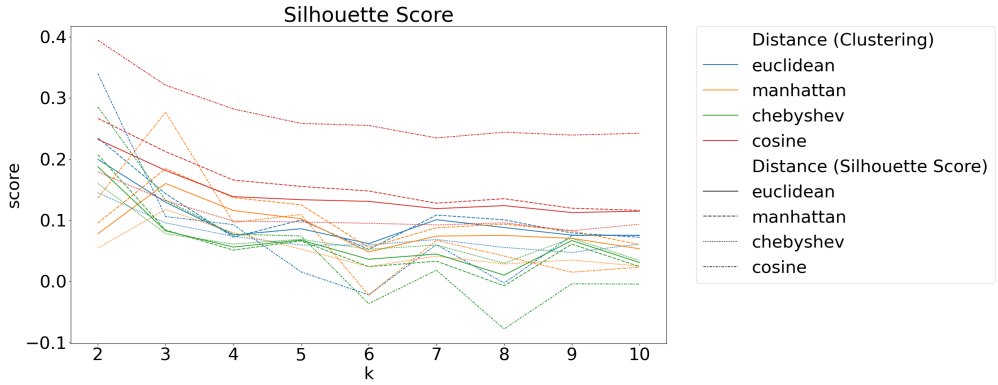


(d) Homogeneity Score



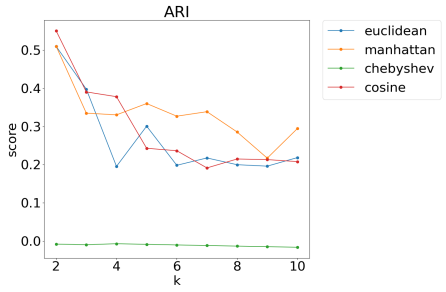
(e) Silhouette Score

Figure 24: Comparison of clustering scores for K-medians-clustering on Wine dataset

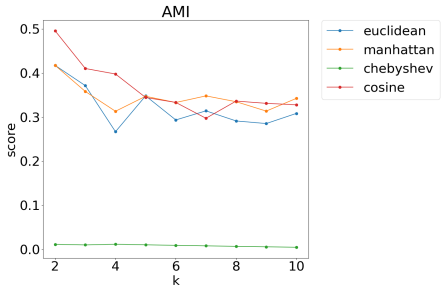


(a) Silhouette Score

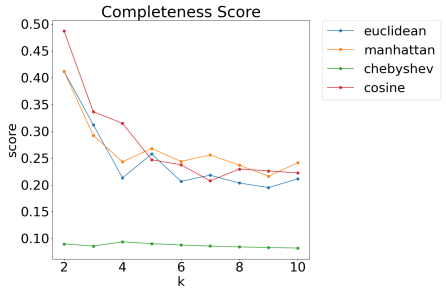
Figure 25: Comparison of clustering scores for K-medians-clustering on Diabetes dataset



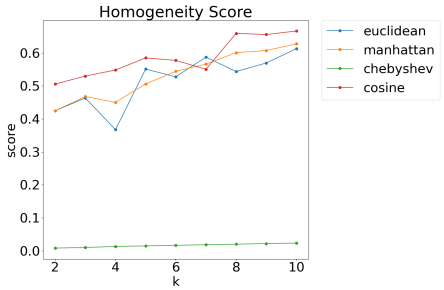
(a) ARI



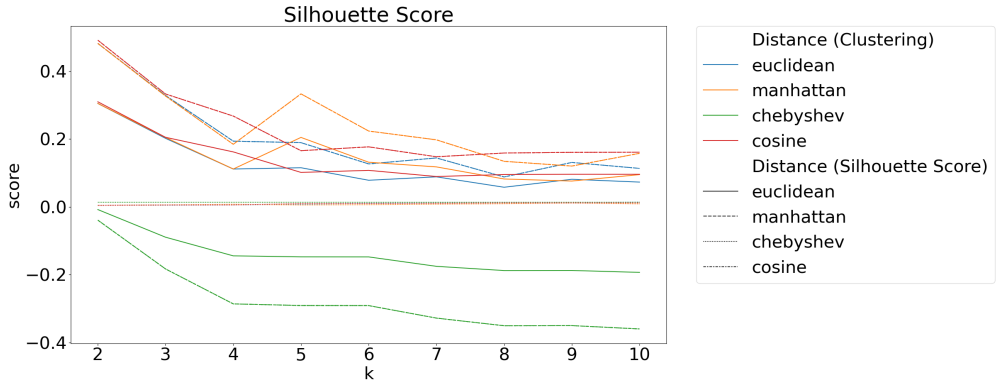
(b) AMI



(c) Completeness Score

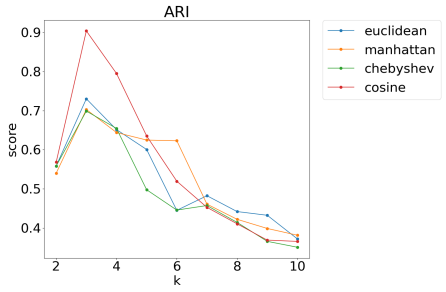


(d) Homogeneity Score

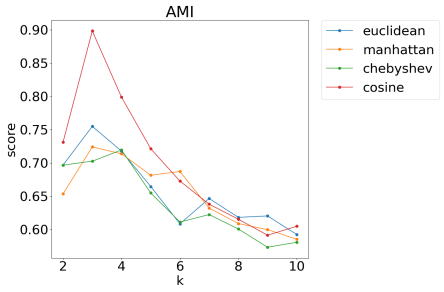


(e) Silhouette Score

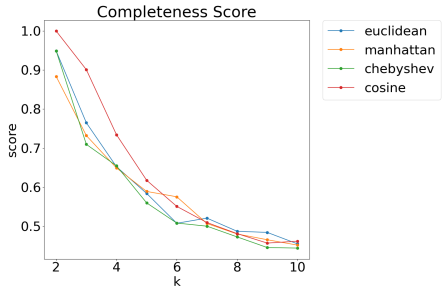
Figure 26: Comparison of clustering scores for K-medians-clustering on Housevotes dataset



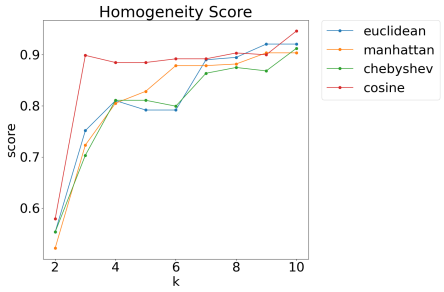
(a) ARI



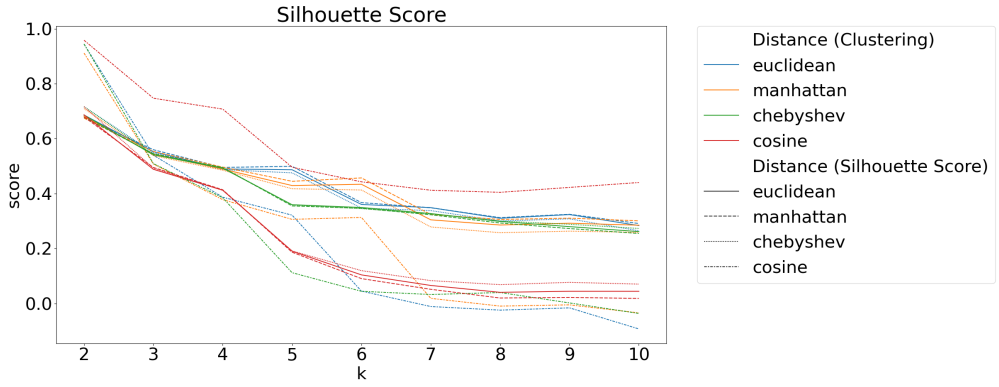
(b) AMI



(c) Completeness Score

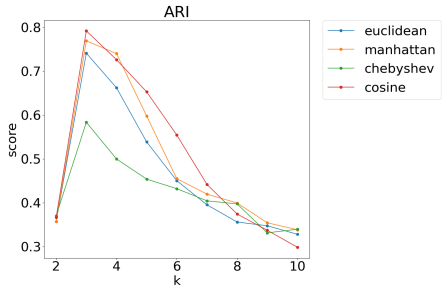


(d) Homogeneity Score

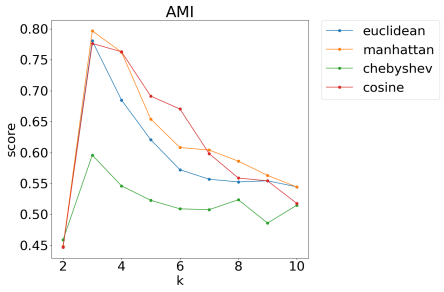


(e) Silhouette Score

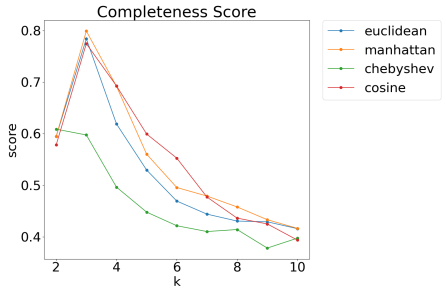
Figure 27: Comparison of clustering scores for K-medoids-clustering on Iris dataset



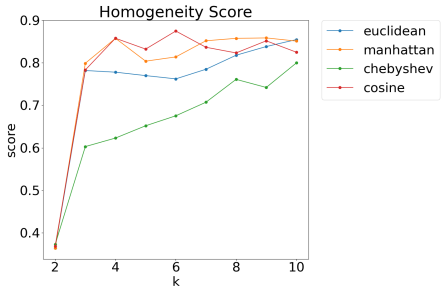
(a) ARI



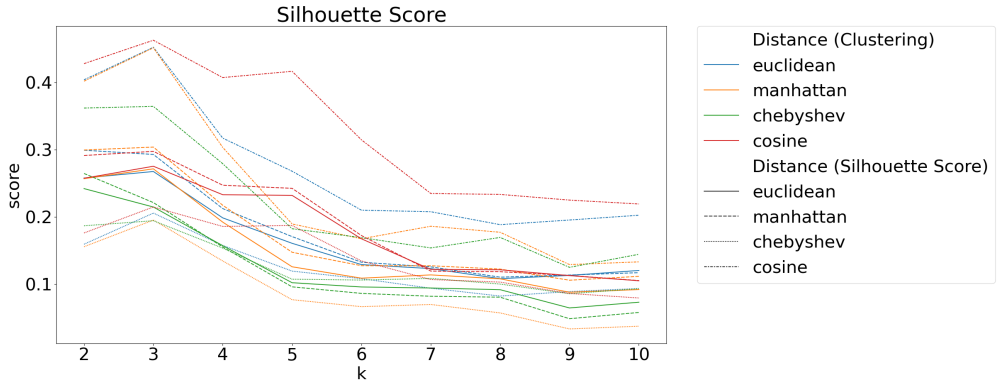
(b) AMI



(c) Completeness Score

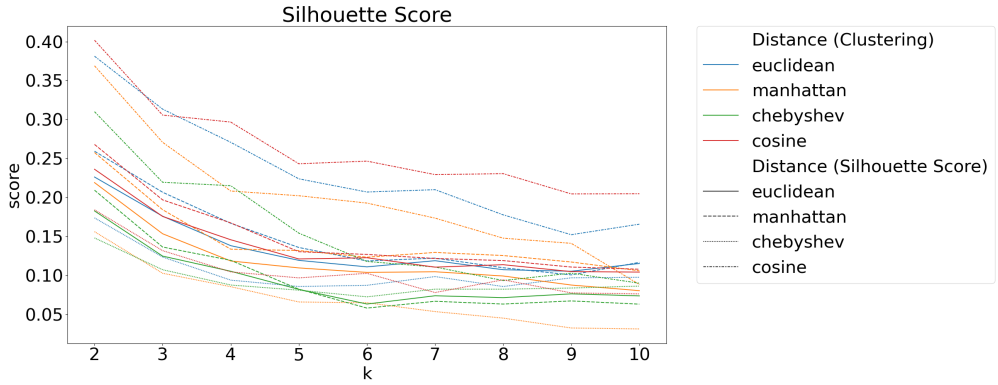


(d) Homogeneity Score



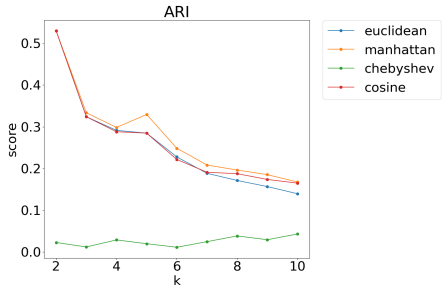
(e) Silhouette Score

Figure 28: Comparison of clustering scores for K-medoids-clustering on Wine dataset

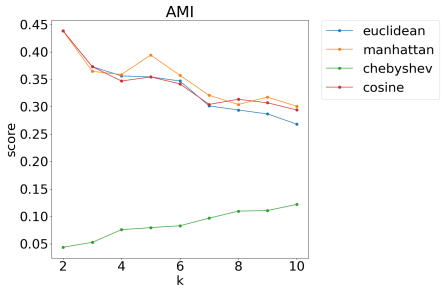


(a) Silhouette Score

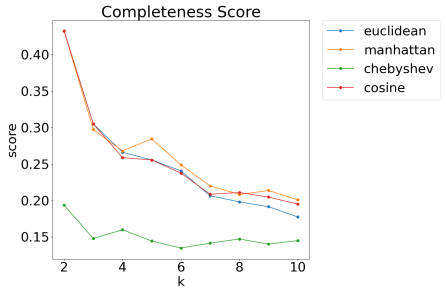
Figure 29: Comparison of clustering scores for K-medoids-clustering on Diabetes dataset



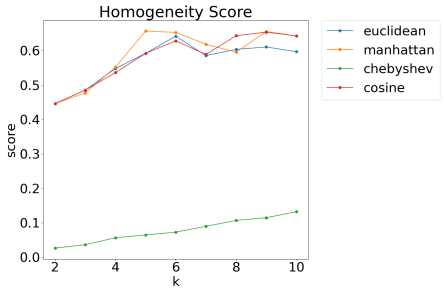
(a) ARI



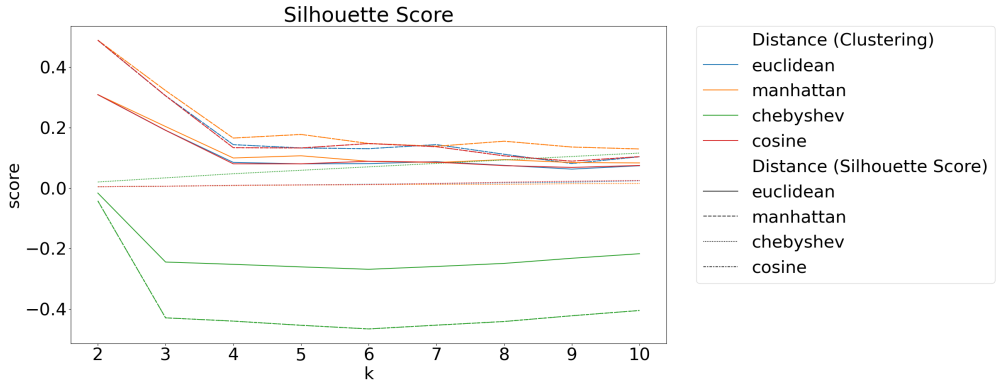
(b) AMI



(c) Completeness Score



(d) Homogeneity Score

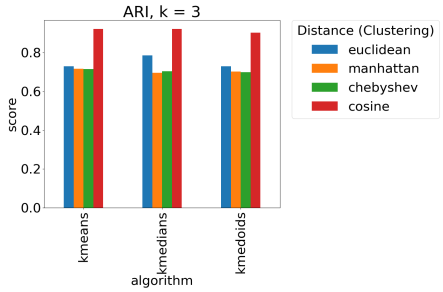


(e) Silhouette Score

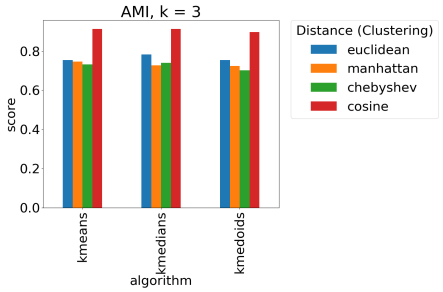
Figure 30: Comparison of clustering scores for K-medoids-clustering on Housevotes dataset

A.2 Comparison for given k value

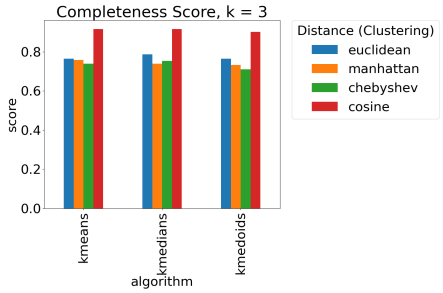
For each dataset we independently compared clustering scores of different algorithms for multiple distances. For the algorithms that require a predefined number of clusters we set the value k to be the number of different classes in the dataset. For the diabetes dataset, which does not have categorical labels we have chosen k to be equal to the k value, that gives the highest silhouette score.
For the silhouette score the mean value and a standard deviation for the score evaluated with all four distance measures is shown.



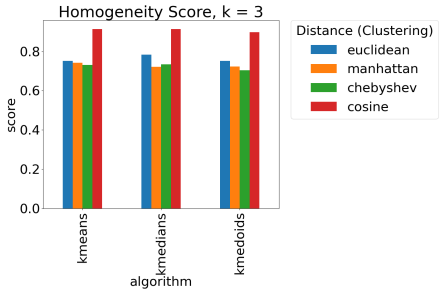
(a) ARI



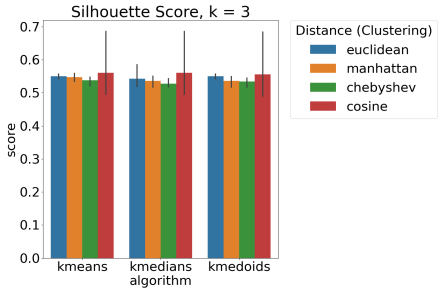
(b) AMI



(c) Completeness Score

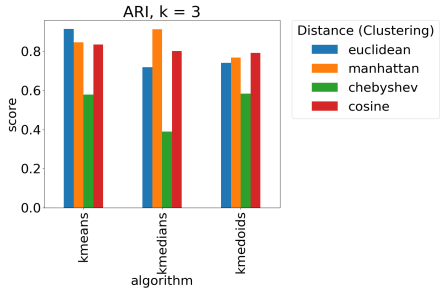


(d) Homogeneity Score

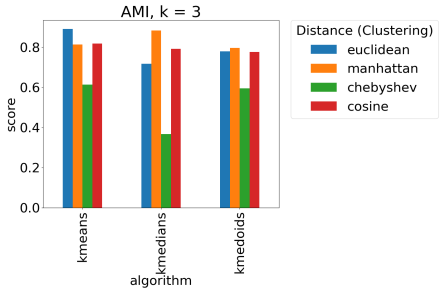


(e) Silhouette Score

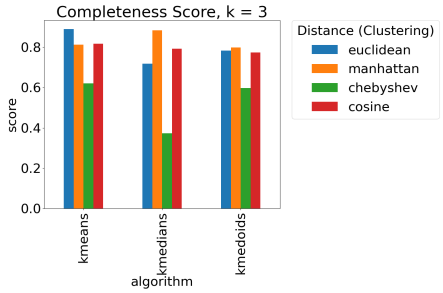
Figure 31: Comparison of clustering scores for Iris dataset (given k=3)



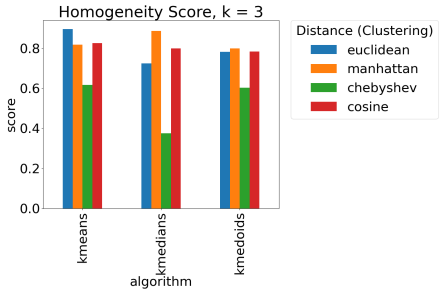
(a) ARI



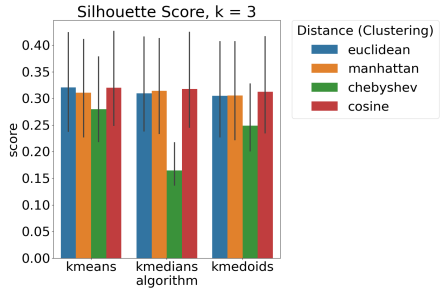
(b) AMI



(c) Completeness Score

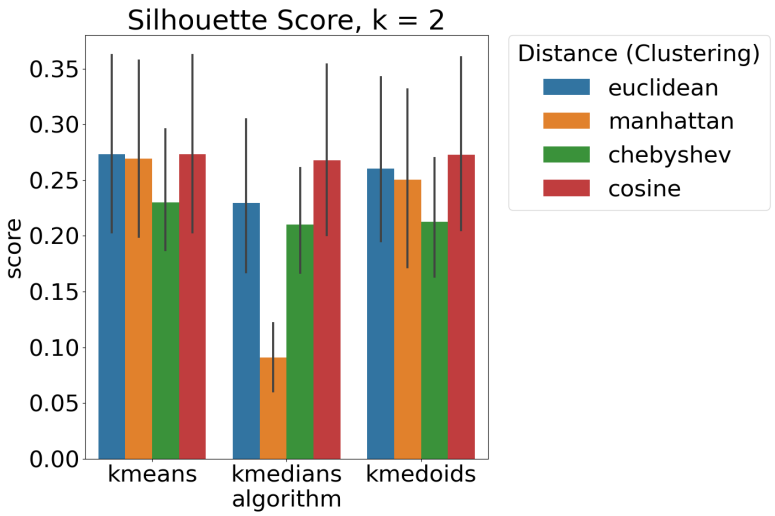


(d) Homogeneity Score



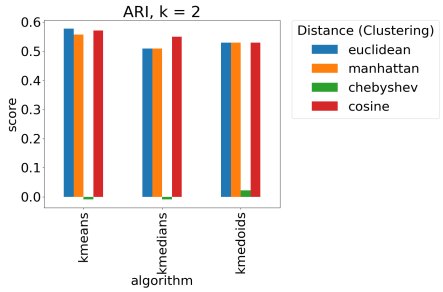
(e) Silhouette Score

Figure 32: Comparison of clustering scores for Wine dataset (given k=3)

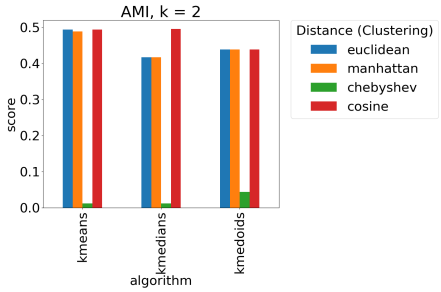


(a) Silhouette Score

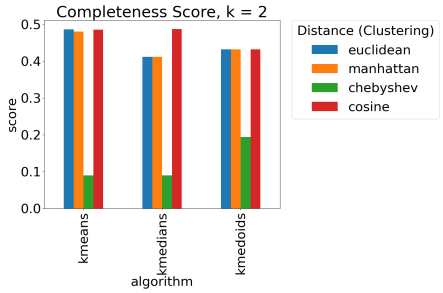
Figure 33: Comparison of clustering scores for Diabetes dataset (given k=2)



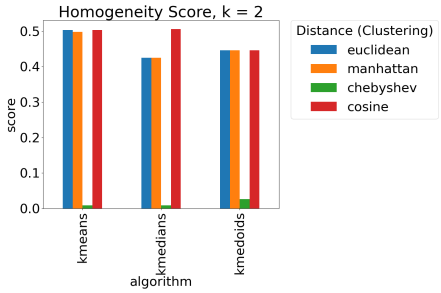
(a) ARI



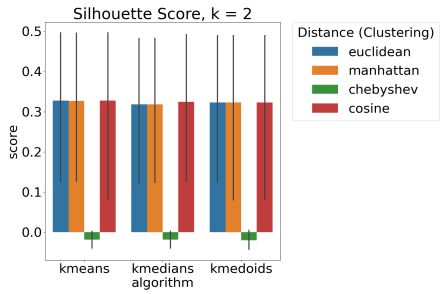
(b) AMI



(c) Completeness Score



(d) Homogeneity Score



(e) Silhouette Score

Figure 34: Comparison of clustering scores for Housevotes dataset (given k=2)

B Comparison of runtimes

The runtimes were calculated on AMD Ryzen 3600X, 16GB 3200MHz DDR4 RAM. We took the average runtime of ten trials. The value k was set to be the number of different classes in the dataset for Iris, Wine and Housevotes dataset. For the diabetes dataset, which does not have categorical labels we have chosen $k = 2$ to be equal to the k value, that gives the highest silhouette score. For DBSCAN the parameters were used that give the highest ARI score for Iris, Wine and Housevotes and the highest Silhouette score for Diabetes dataset.

	Euclidean	Manhattan	Chebyshev	Cosine	Total
K-Means	0.001300	0.001143	0.001214	0.057548	0.061206
K-Medians	0.001824	0.002235	0.005253	0.873826	0.883138
K-Medoids	0.032768	0.026929	0.030192	0.269327	0.359216
DBSCAN	0.000683	0.000736	0.000640	0.000805	0.002864
Total	0.036575	0.031044	0.037299	1.201506	1.306424

Table 2: Runtimes for Iris dataset (in seconds)

	Euclidean	Manhattan	Chebyshev	Cosine	Total
K-Means	0.001686	0.001692	0.001632	0.112143	0.117153
K-Medians	0.003722	0.004837	0.005135	0.074670	0.088364
K-Medoids	0.112080	0.091926	0.102531	0.462057	0.768594
DBSCAN	0.001123	0.001168	0.000848	0.000895	0.004034
Total	0.118611	0.099622	0.110146	0.649764	0.978144

Table 3: Runtimes for Wine dataset (in seconds)

	Euclidean	Manhattan	Chebyshev	Cosine	Total
K-Means	0.005803	0.003188	0.003208	0.136975	0.149174
K-Medians	0.012511	0.014224	0.014343	0.134298	0.175376
K-Medoids	0.558538	0.455667	0.481983	2.495598	3.991786
DBSCAN	0.003577	0.004333	0.003563	0.002348	0.013822
Total	0.580429	0.477412	0.503097	2.769219	4.330157

Table 4: Runtimes for Diabetes dataset (in seconds)

	Euclidean	Manhattan	Chebyshev	Cosine	Total
K-Means	0.005589	0.005560	0.008237	0.341636	0.361021
K-Medians	0.281766	0.285803	0.011203	0.220306	0.799078
K-Medoids	2.093371	1.815214	1.911084	6.242655	12.062324
DBSCAN	0.011879	0.012103	0.006456	0.004406	0.034844
Total	2.392605	2.118680	1.936980	6.809003	13.257267

Table 5: Runtimes for Housevotes dataset (in seconds)