

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, 2nd semester

JULIAN RUMMEL

6673334

s9594673@stud.uni-frankfurt.de

Master Bioinformatics, 2

NIKLAS CONEN

6599913

conen@stud.uni-frankfurt.de

branch of study (Bachelor Computer Science, 8)

Date: June 6, 2021

Chosen Project Topic:

T4 - DISTANCE MEASURES AND CLUSTERING

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-Median, 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. Our results show that

Contents

1	Definition of Distance Measure	5
2	Different Distance Measurements	5
2.1	Manhattan distance	5
2.2	Euclidean Distances	6
2.3	Angular cosine Distance	7
2.4	Chebyshev Distance	8
3	Data Set Description	9
3.1	Housevotes	9
3.2	Wine recognition dataset	9
3.3	Iris dataset	10
3.4	Diabetes dataset	10
4	Clustering Algorithms	10
4.1	K-Means	10
4.2	K-Medoids	11
4.3	K-Median	12
4.4	DBSCAN	13
5	Additional Methods Used	15
5.1	k++-Initialiser	15
5.2	One-Hot-Encoding	15
6	Description of Python libraries used	16
6.1	pyclustering	16
6.2	scikit learn	16
6.3	scikit learn extra	16
6.4	numpy	16
6.5	seaborn	17
6.6	streamlit	17
7	Description of Evaluation Module	17
7.1	t-SNE	17
7.2	Principal Component Analysis (PCA)	18
8	Web Frontend and User Manual	19

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, containing two sets of points. If $d(x, y)$ satisfies the following three axioms the distance measure is classified as a *metric*:

$$d(x, y) = 0 \Leftrightarrow x = y \quad \text{Identity of indiscernibles} \quad (1.1)$$

$$d(x, y) = d(y, x) \quad \text{Symmetry} \quad (1.2)$$

$$d(x, y) \leq d(x, z) + d(z, y) \quad \text{Triangle inequality} \quad (1.3)$$

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.

$$d(x, y) \geq 0 \quad \text{Non Negativity} \quad (1.4)$$

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:

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

where x and y are vectors

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

The Manhattan distance is, like the Euclidean distance, part of the L_p - *metrics* (see 2.2), 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 $\sum_{i=1}^n |x_i - y_i| = \sum_{i=1}^n |y_i - x_i|$.

2.2 Euclidean Distances

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

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

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

$$d(x, y) = \sqrt{\sum_{i=1}^n (|x_i - y_i|)^2} \quad (3)$$

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 clearly 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 triangle 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 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 defined as:

$$\text{cosine similarity} = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2 \sum_{i=1}^n y_i^2}} \quad (4)$$

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 [2]:

$$\text{angular cosine distance} = \frac{\arccos(\text{cosine similarity})}{\pi} \quad (5)$$

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).

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

1. Identity of indiscernibles:

Two vectors can have a 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 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 2). 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 distance between two vectors on one axis. Formally it is defined as:

$$d(x, y) = \max(|x_i - y_i|) \quad (6)$$

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

$$d(x, y) = \lim_{p \rightarrow \infty} \left(\sum_{i=1}^n (|x_i - y_i|^p)^{\frac{1}{p}} \right) \quad (7)$$

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 [3]. The dataset consists of voting results of the U.S. House of Representatives Congressmen 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) documenting this session of Congress. The voting results are split into nine different types by the CQA, which are consolidated into three results used in the dataset.

Voted for, paired for, and announced for count as a yes vote. Voted against, paired against, and announced against count as a no vote. Voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known are denoted as a unknown state.

The set consists of two classes, 267 democrats and 168 republicans.

3.2 Wine recognition dataset

This dataset contains the chemical analysis results of Italian wines from 3 different cultivators. It is also taken from the UCI Machine Learning Repository [3]. The dataset consists of 178 instances, each of them having 13 numeric attributes according to different measurements taken for different constituents (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.

3.3 Iris dataset

The Iris Dataset, introduced by Ronald Fisher in 1936, contains the petal and sepal measurements of three different species of Iris [?]. The considered Irises are Setosa, Versicolour and Virginica. For each Iris, 50 samples are included and for each sample the dataset contains the sepal length, sepal width, petal length and petal width in cm.

3.4 Diabetes dataset

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. [4] with the main goal of constructing a model that predicts the response (column 11) from the covariates (column 1-10).

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. Initially, the algorithm picks k points whose positions each represent one cluster centroid
2. All points are considered in turn:
 - Find the nearest centroid/mean of the considered point (Euclidean distance measure)
 - Assign point to cluster of that centroid
 - Adapt position of this centroid
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 the conducting the following steps:

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

After the K-means algorithm assigned all points, an optional step of reassigning the points with fixed centroids can be conducted. This can be sensible since it is possible that after a point has been assigned to cluster the centroids move so far that the point would now belong to a different cluster.

4.2 K-Medoids

The K-Medoids clustering method is related to the well-known 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. [5] It partitions the dataset by assigning each data point 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 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, that works as following: [6]

1. First initialize the algorithm by selecting k data points to be the medoids and assigning every data point to its closest medoid.
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 re-assign the data points to the new medoids and go back to step 2.

4.3 K-Median

The K-Medians cluster algorithm is 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. [7]

$$Q(\{\pi_j\}_{j=1}^K) = \sum_{j=1}^K \sum_{x \in \pi_j} \|x - c_j\|_1 \quad (8)$$

The geometric median is used for the minimization.

$$\arg \min_{y \in \mathbb{R}^n} \sum_{i=1}^m \|x_i - y\|_2 \quad (9)$$

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. 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. [7] Achieving global optimization

in k-medians is known to be NP-complete [8].

The algorithm works as following: [9]

1. Assign each dataset to a cluster, thus its nearest cluster center using the Manhattan distance as default.
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.

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 [10] or their revisit of DBSCAN [11] and only apply to this algorithm.

Contrary to the aforementioned centroid-based partitioning algorithms (k-means, k-medoids and k-median) 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 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 point*. 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. $|N_{Eps}(q)| \geq \text{MinPts}$

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.

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 .

Now a cluster can be described as:

Definition 4: *cluster*

A cluster is a non empty subset $C \in D$ so that:

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

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

Using these definitions DBSCAN can begin the clustering process with given values for Eps and MinPts . In the beginning all points are not labeled. Beginning with an arbitrary point p all points are iterated in a linear fashion. For each point a **RangeQuery** function is executed finding all density-reachable neighbours of p . If **RangeQuery** finds more than MinPts neighbours then p is a core point and is labeled as such. Otherwise p is marked as Noise.

In the next step every point in the Neighbourhood excluding p is expanded. Unlabeled Points get checked for the core point condition (which equals a **RangeQuery** call). Points that got labeled as Noise before are labeled as core points. When the expansion comes to an end a cluster is yielded and the next

unlabeled point is chosen as p .

Two clusters may be merged if their distance is below Eps . The distance between two clusters C_1 and C_2 is defined as $\|C_1, C_2\| = \min\{\|q, p\| \mid p \in C_1, q \in C_2\}$.

The runtime complexity of DBSCAN heavily depends on the runtime of the `RangeQuery` function and the distance measure. Thus the runtime can exceed $\mathcal{O}(n^2)$ depending on the chosen implementations. A detailed discussion of DBSCANs runtime can be found in [11].

5 Additional Methods Used

5.1 k++-Initialiser

5.2 One-Hot-Encoding

Categorical data is represented by specific discrete values or labels. This is case for the `housevotes` dataset (see section 3.1), where voting results can have one of three values (y, n, ?). The distance measures described in section 2 need numerical data to work. The categorical data therefore needs to be converted (encoded) to numbers which accurately describe their distance to another. Using simple integer encoding where no is encoded as 0, yes is encoded as 1 and the unknown state is encoded as 2 results in a yes vote being classified as closer to the unknown state than a no vote by the distance measures.

For so called non ordinal data (data which has no known order) like the votes, One-Hot-Encoding provides better results when using distance based clustering.

With One-Hot-Encoding every attribute is represented as 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.

The `scikit-learn` library [12] was used to implement One-Hot-Encoding on the

housevotes dataset.

6 Description of Python libraries used

Multiple libraries were used for their implementation of cluster algorithms and other functionality needed for working with results and visualising them. Below follows a brief description of all used libraries.

6.1 pyclustering

pyclustering is a data mining library, focusing on clustering algorithms written in C++ and Python by Andrei Novikov [13]. 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.

We use pyclustering's implementation of K-Means and K-Medians.

6.2 scikit learn

scikit learn is a wide ranging data analysis tool kit for python encompassing algorithms not only for clustering but for classification, regression and more [12].

The library is used for its implementations of DBSCAN, One-Hot-Encoding, StandardScaler, TSNE and PCA.

6.3 scikit learn extra

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

6.4 numpy

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

6.5 seaborn

seaborn is a powerful data visualisation library build on top of python's matplotlib. seaborn simplifies plotting predefined templates. We use seaborn's `scatterplot` template for visualising the TSNE and PCA projections of the cluster results.

6.6 streamlit

streamlit is a easy-to-use library for building simple web apps. Our webfrontend is implemented in streamlit and is also hosted on their sharing service.

— TODO — [Link zur gehosteten app](#)

7 Description of Evaluation Module

7.1 t-SNE

t-SNE is a nonlinear dimensionality reduction technique, which was developed by Laurens van der Maaten and Geoffrey Hinton [14]. 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).

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 [15]. 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 .” [14]

$p_{j|i}$ for $i \neq j$ is given as

$$p_{j|i} = \frac{\exp(-d(x_i, x_j)^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-d(x_i, x_k)^2/2\sigma_i^2)} \quad (10)$$

and set $p_{i|i} = 0$.

The joint probability p_{ij} is defined by

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \quad (11)$$

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. 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. 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 $i \neq j$ is defined as followed

$$q_{ji} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}} \quad (12)$$

and set $q_{i|i} = 0$.

Now the Kullback-Leibler divergence can be expressed as is defined as followed

$$KL(P||Q) = \sum_j \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad (13)$$

The optimization procedure is performed by a gradient descent method to find a local minimum.

The final results may heavily depend on the chosen parameters, especially the perplexity value. It is therefore recommended to compare different perplexity values to identify spurious clustering artifacts in the visualization. [16]

7.2 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: 14) of the columns of X with maximum variance (see: 15) are searched for, given by:

$$\sum_{j=1}^p a_j x_j = Xa \quad (14)$$

$$\text{var}(Xa) = a^T Sa \quad (15)$$

with a as vector of constants a_1, \dots, a_p and S as sample covariance matrix associated with the dataset. [17]

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. [18] This produces the equation: [17]

$$Sa - \lambda a = 0 \Leftrightarrow Sa = \lambda a \quad (16)$$

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: [17]

$$\pi_j = \frac{\lambda_j}{\sum_{j=1}^p \lambda_j} \quad (17)$$

This results in a matrix called feature vector, which contains all the remaining components as columns and forms the final dimension of the reduced data set [18]. Usually, the requirements of graphical representation lead to keeping the first two to three principal components [17]. Finally, the original data gets reorientated from the original axes to the axes represented by the principal components.

8 Web Frontend and User Manual

ANMERKUNGEN:

- Describe the implementation and write a brief user manual with screenshots.

9 Conclusion

ANMERKUNGEN:

- Summarize the main points and achievements
- Add your own assessment/criticism on the topic

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] Cosine distance, cosine similarity, angular cosine distance, angular cosine similarity, Jul 2017.
- [3] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017.
- [4] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32:407–499, 2004.
- [5] Xin Jin and Jiawei Han. *K-Medoids Clustering*, pages 564–565. Springer US, Boston, MA, 2010.
- [6] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.
- [7] Christopher Whelan, Greg Harrell, and Jin Wang. Understanding the k-medians problem. *Int’l Conf. Scientific Computing*, pages 219–222, 2015.
- [8] Sirion Vittayakorn, Sanpawat Kantabutra, and Chularat Tanprasert. The parallel complexities of the k-medians related problems. *IEEE Xplore*, 2008.
- [9] K-means and k-medians, 2020.
- [10] 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.
- [11] 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.
- [12] 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. Duch-

- esnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [13] Andrei Novikov. PyClustering: Data Mining Library. *Journal of Open Source Software*, 4(36):1230, apr 2019.
 - [14] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.
 - [15] Mathworks documentation, t-sne, 2021.
 - [16] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016.
 - [17] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *The Royal Societey Publishing*, April 2016.
 - [18] Zakaria Jaadi. A step-by-step explanation of principal component analysis (pca), April 2021.
 - [19] Moshe Lichman. UCI Machine Learning Repository, 2013.
 - [20] Harro Heuser. *Lehrbuch der Analysis : Teil 2*. Mathematische Leitfäden. B.G. Teubner, Stuttgart, 11. auflage edition, 2000.
 - [21] 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.