Agglomerative Sequence Clustering in Python

Alexander Grentner

BACHELORARBEIT

eingereicht am
Fachhochschul-Bachelorstudiengang

Medizin- und Bioinformatik

in Hagenberg

im Juli 2021

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere. This printed copy is identical to the submitted electronic version.

Hagenberg, July 15, 2021

Alexander Grentner

Contents

Abstract

Clustering sequences is an important task in biological data analysis. It is possible to find similarities within a given data set. A common approach is clustering sequences hierarchically or agglomerative. This enables the user to evaluate the data by similarity by calculating distances within the data points and linkages. Agglomerative clustering methods need the numerical input values. This is necessary for calculation of the differences between the sequences here strings. The result of a distance calculation is then a numerical value which can be used in different clustering algorithms. In the following the focus is on the Hamming-, Levensthein-, Damerau- and Jaro distance, which differ mainly in their complexity of cluster calculation and therefore implicitly in their runtime. With the use of the mentioned distance metrics, the calculated distance matrix will be used for further clustering calculation. For this, linkages will be used which define the agglomeration within the clustering process. In each iteration the clusters will be merged based on the calculated distance matrix and the defined linkage. The relevance of the linkage and the calculated distance matrix can be visualised in a dendrogram, especially the effect of chaining within the hierarchical clustering. All algorithms are implemented in Python 3.9 and are applied on a data set of 120 000 sequences in a FASTQ file. Especially the range of functionalities in the packages Biopython an scikit-learn are compared. Additionally, the runtime of both packages will be compared with other established frameworks like CD-HIT and online packages. The runtime increases by a larger data set, which is a significant disadvantage, not only for the clustering calculation but also for the calculation of the distance matrix. The results show that the mentioned approach with a data set containing more than 8000 sequences is not suitable in Python and should not be used. Other established frameworks which use a greedy approach during the clustering process are much faster and have therefore a huge advantage compared to agglomerative algorithms implemented in Python. Frameworks such as CD-HIT iterates only once over the data set and do not calculate a distance matrix before. However, the result of an greedy approach is highly dependant on the input order of the sequences, in case of different input lengths the sequences have to be sorted before. In conclusion, sequence clustering in Pyhton by using an agglomerative approach is not recommended for bigger data sets, especially without a previous calculated distance matrix. However, it should be outlined that Biopyhton and scikit-learn provide a variety of methods for clustering and its visualization, which are easy to implement. This can be advantageous especially for smaller data sets with no further insight on the data, where the explained approach is used to gather a first overview of the data.

Kurzfassung

Das Gruppieren bzw. Clustern von Sequenzen spielt eine wichtige Rolle in der biologischen Datenanalyse. Damit ist es möglich Ähnlichkeiten verschiedener Datenpunkte innerhalb eines Datensatzes zu erkennen. Eine beliebte einfache Methode ist es Sequenzen hierarchisch beziehungsweise agglomerativ zu clustern. Hierarchisches Clustering ermöglicht Ähnlichkeiten verschiedener Sequenzen unter Verwendung verschiedener Distanz- und Ähnlichkeitsmaßen zu evaluieren. Diese benötigen numerische Werte um Ähnlichkeiten bestimmen zu können. Daher ist zuvor die Berechnung einer Distanzmatrix zwischen den Sequenzen notwendig. Mit Hilfe von verschiedenen Distanzen können die Unterschiede zwischen verschiedenen Sequenzen berechnet werden. Das Resultat ist dementsprechend ein numerischer Wert, welcher in weiterer Folge den verschiedenen Clustering-Algorithmen als Eingabeparameter dient. Bei den Distanzmaßen wird im Wesentlichen auf die Hamming-, Levensthein-, Damerau- und Jaro-Distanz eingegangen. Diese Distanzen unterscheiden sich in ihrer Komplexität der Berechnung und daher implizit in ihrer effektiven Laufzeit. Die, mit Hilfe der oben erwähnten Distanzmatrix , erstellte Distanzmatrix dient in weiterer Folge als Basis für das weiterführende Clustering. Für den Kern des Clusterings, die Berechnung der Ähnlichkeit der Sequenzen, werden Ähnlichkeitsmetriken, sogenannte Linkages, verwendet. Diese werden für das Zusammenführen der einzelnen Cluster benötigt. Bei jedem Schritt wird über die Datenpunkte der Distanzmatrix iteriert und unter Verwendung des definierten Linkage-Kriteriums zu Clustern zusammengeführt. Dies wird so lange durchgeführt, bis alle Datenpunkte in zu einem Cluster zusammengeführt wurden. Bei der Visualisierung mittels Dendrogramm ist die Bedeutung der Linkage und die Effekte der Distanzmatrix zu erkennen. Es kann dabei beispielsweise zu den Verkettungen bei Single-Linkage Clustering kommen, wodurch Ausreißer innerhalb des Datensatzes gut zu erkennen sind. Die Implementierung wurde mit Python 3.9 umgesetzt. Dabei wurde ein Datensatz von 120 000 Sequenzen im FASTQ-Format als Testdatensatz verwendet. Es wurde dabei speziell auf den Umfang der Funktionalitäten der Packages Biopython und Scikit-learn geachtet, sowie auf die Laufzeit der enthaltenen Clustering-Algorithmen. Die Ergebnisse wurden mit denen von etablierten Frameworks, wie CD-HIT, und Onlinepackages verglichen. Die Laufzeiten, der verwendeten Packages zeigen bei zunehmendem Umfang des Datensatzes einen erheblichen Nachteil. Die schlechte Performance bezüglich der Laufzeit bezieht sich dabei sowohl auf die initialen Berechnungen der Distanzmatrix als auch auf den darauffolgenden Clustering-Prozess. Es kam deutlich hervor, dass ab einem Umfang von circa 8000 Sequenzen der Ansatz mit einer zuvor berechneten Distanzmatrix und einer nachfolgenden agglomerativen Methode in Python nicht sinnvoll ist. Bereits etablierte Frameworks, welche einen Greedy-Algorithmus verwenden sind wesentlich performan-

ter. Der ausschlaggebende Grund ist, dass Frameworks wie CD-HIT oder UClust durch ihren Greedy-Ansatz lediglich einmal über den gesamten Datensatz iterieren, im Gegensatz zu einem notwendigen Vergleich aller Sequenzen bei den Berechnungen einer Distanzmatrix und der agglomerativen Clusteringmethode. Nachteil dieser Frameworks ist jedoch, dass durch einen Greedy-Ansatz die Ergebnisse stark von der Eingabereihenfolge abhängig sind und daher, bei unterschiedlichen Sequenzlängen, vorher sortiert werden müssen. Durch die lange Laufzeit und die daraus resultierende Beschränkung der Größe des Datensatzes ist ein agglomertiver Ansatz mit einer zuvor berechneten Distanzmatrix in Python nur bei einem überschaubaren Datensatz sinnvoll. Hierbei ist dennoch die Flexibilität der Packages Biopython und Scikit-learn von Vorteil, da diese umfangreiche Methoden für Clustering und die Erbegnisvisualisierung anbieten. Dies ist insbesondere bei unbekannten Datensätzen sinnvoll, da die genannten Methoden die Intention haben einen Überblick über die Ausgangssituation zu schaffen.

Chapter 1

Introduction

Clustering biological sequences is a common approach in bioinformatics to derermine similarities of nucleotide sequences or amino acid sequences of different sources. Python provides a range of possibilities in sequence clustering and is therefore widely used for tasks in data analysis. Python libraries such as Biopyhton [23] and scikit-learn [31] already provide clustering algorithms that can be used for various problems, including sequence clustering. Both packages provide similar functionalities and approaches. However, both are not specifically designed for clustering biological sequences on a larger scale. This thesis will determine the handling of biological data within the packages, by comparing the packages in general and evaluating their application on given data sets. The main focus is agglomerative clustering of sequences and the resulting overall performance of each clustering step, in this case the evaluation of a distance matrix of the given sequence data set, linkage methods within the hierarchical clustering process and the visualization with dendrograms.

Packages such as Biopyhton are specifically designed for biological data analysis on a smaller scale, not only for sequence clustering and therefore provide a range of functionality targeting biological sequence data. Thus, it will be compared with the scikit-learn package which is not as application specific designed and offers a more general usage. Furthermore, the process of sequence clustering will be conducted, and each package will be analysed in each clustering step. Starting by calculating the distance for each sequence provided, with different distances using the package jellyfish [29]. Further, results derived from the mentioned packages are compared based on the advantages of their functionality. The process is conducted similar with each package, where a distance matrix has to be previously defined. As well as the subsequent visualization which is not necessarily dependent on a package.

Additionally, the aim of this research is a detailed analysis of commonly used python packages and their clustering algorithms in a biological context. The main evaluation is the runtime analysis of the distance matrix calculation of each clustering algorithm, provided by the packages. Additionally, a theoretical overview of agglomerative clustering with sequence will be given, which then will be used as an approach for the given data set. Further a determination whether an hierarchical clustering approach is useful for the given data set and implicitly evaluate the use-cases of agglomerative sequence clustering using Python. The results collected during the process will be used to determine the advantages and disadvantages of each package as well as commonly used

alternatives such as online packages or packages based on other programming languages.

For clustering algorithm evaluation a data set containing so called unique molecular identifier has been provided. UMIs are short nucleotide sequences that are attached to DNA sequences before sequencing. It can be assumed that all UMI sequences have the same length, which is relevant for some sequence clustering algorithms.

1.1 Relevance in Bioinformatics

Sequence clustering is a task in bioinformatics to cluster certain protein, nucleotide or amino acid sequences. The reason sequences such as nucleotide or amino acid sequences can be clustered is that they share specific nomenclatures. Clustering algorithms can be used to find similarities, within a given data set. The similarities can then be used to determine a representative sequence for the data set or the clustered groups. Therefore, it is possible to abstract larger data sets in groups and their representatives. Building phylogenetic trees is similar to sequence clustering by using the same algorithm but focused on evolutionary steps within the sequences based on their similarity. Therefore, all phylogenetic trees use clustering algorithms but are not necessarily agglomerative clusters based on ancestors and their evolution. Because of the increasing amount of data in biology, especially induced of Next-Generation Sequencing (NGS), clustering sequences with assessable performance, especially on larger scale, is still relevant. Clustering algorithms and tools can find certain features and similarities within the data set, without having much previous insight of the data. [15]

Chapter 2

Methods

In the following the steps for agglomerative clustering will be described. The main aspects of calculating the distance matrix and merging the calculated clusters by each step will be outlined. Agglomerative clustering uses a bottom-up approach by iteratively joining the nearest elements. Clustering for sequences commonly uses the same approach: in a first step is to calculate the distance matrix, which defines the difference between all sequences in a given data set. Next, the provided distances will be used for clustering by finding the smallest distance between the calculated differences of the sequences and merge them to a cluster, so called agglomeration. With each iteration the clustered data set is merged by their minimal distance of the previous calculated cluster. The matrix of the distances gets updated at each iteration providing the distances for the next agglomeration. Additionally, the length of the branches can be determined by different clustering algorithms within the agglomerative clustering step. The clustering algorithms stops at $N-1$ steps, the last agglomeration is therefore at the final iteration only one cluster [11, chpt. 7; 13, chpt. 7].

2.1 Distance Matrix

To analyse sequence data via agglomerative clustering a distance has to be defined. Distance matrices are used to quantify and compare the similarity between samples, in the case of nucleotide sequences represented as strings. In the following, the most common and established distance matrix algorithms are applied and compared. The focus is hereby at the Hamming distance, Levenshtein distance, Damerau distance and Jaro distance in the context of hierarchical sequences clustering. Because of their frequent usage in a biological context many packages provide functionalities for their calculation. The examples are all implemented with the usage of the same packages which is especially designed for a biological context. The mentioned distances are not only designed for sequences clustering of biological sequences but also used in various problem domains, where a string has to be measured by its similarity. The calculation of a distance matrix is the first step of every hierarchical clustering approach. The result is then used as an input for further clustering calculation [2, chpt. 8].

2.1.1 Hamming distance

The Hamming distance is a well known and a commonly used distance metric for string comparison. Two strings with the same size are needed, which limits its application in a biological context, sequences which are compared often have not the same size. For calculating the distance both strings are compared by each character at their index. If the characters of both strings at the given position are not equal a counter will be increased by a certain value. The value can be chosen depending on the usecase and how much the penalty of the difference between two strings is. As a result the calculated distance represents the sum of all necessary substitutions to receive equal strings. As shown in Figure 2.1, the substitution can be weighted depending on the error rate or a certain application. The figure shows a comparison of two input values with a Hamming distance of three, which is resulted by the sum of each value on a position which is different on the same index. The penalty for each difference is one. Although the Hamming distance is defined for strings with the same length, Python packages like jellyfish already provide a Hamming distance calculation where this restriction is not needed [7].

$$x = x_1, \ldots, x_n \tag{2.1}$$

$$y = y_1, \ldots, y_n \tag{2.2}$$

$$\Delta(x, y) := |\left\{j \in \{1, \ldots, n\} \mid x_j \neq y_j\right\}| \tag{2.3}$$

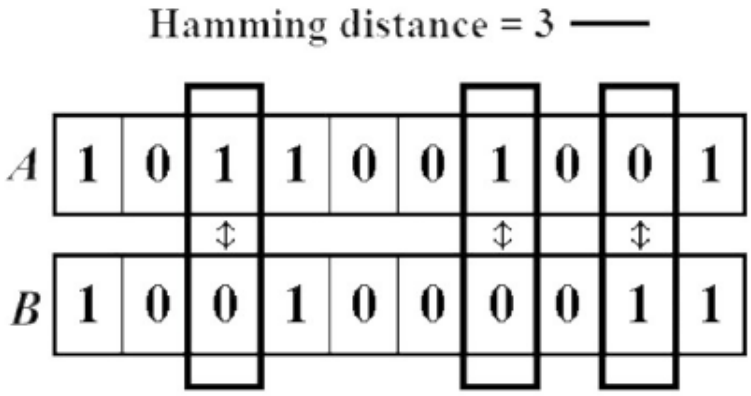


Figure 2.1: Hamming distance [20]

2.1.2 Levenshtein distance

The Levenshtein distance is a commonly used distance metric for biological sequences and was firstly introduced by Wladimir Levenshtein in 1965, as a string comparison algorithm. Unlike the hamming distance, the length of the input strings can be different. The Levenshtein distance hereby defines three states: deletion, insertion and substitution, which are weighted with a specific value. For the distance calculation a $n*m$ matrix is build, where n is the size of the first string and m the size of the string to compare.

This results in a time complexity with $\mathcal{O}(n * m)$. (2.4) Matrix calculation is based on the editing distance, which is the number of modifications needed for equal strings. (To achieve hereby results calculating the distance by building a matrix with the properties in (2.4) dynamic programming is used. The calculation time of each cell of the matrix is constant.) The runtime depends on the string sizes of the two strings which are compared. In comparison to more complex distance algorithms, the Levenshtein distance calculation is quick and is able to calculate a distance of two strings with different size. As seen in the formula the minimal value of a combination of substitution, insertion and deletion is the resulting distance value. If the value at the index is the same the distance will not be increased [12, 16].

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} & +0 \text{ if } u_i = v_j \\ D_{i-1,j-1} & +1(Substitution) \\ D_{i,j-1} & +1(Insertion) \\ D_{i-1,j} & +1(Deletion) \end{cases} \tag{2.4}$$

2.1.3 Comparison Levenshtein and Hamming Distance

As mentioned above, the Hamming distance needs two strings with the same size. The algorithm only compares the characters at the same index of both strings without considering insertions or deletions. The Hamming distance therefore is in some cases, such as the example below (Figure 2.2), not as intuitive and much more strict than other distance calculation approaches, like the Levenshtein distance (Figure 2.3). On the other hand, the Hamming distance is easy to implement and quick by executing [30].

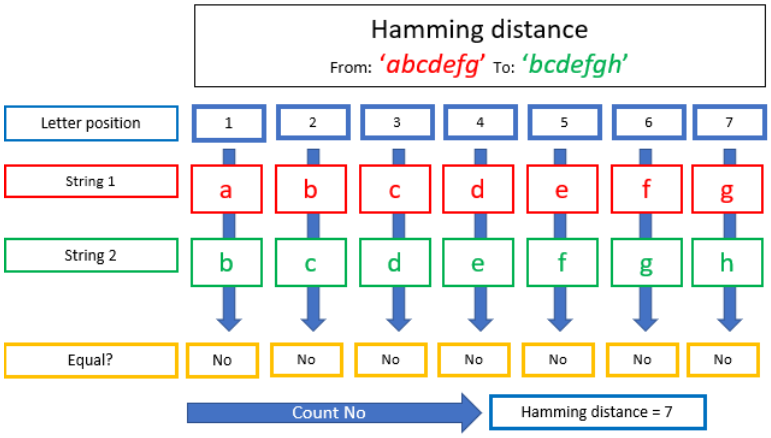


Figure 2.2: Hamming Distance of two strings with shift [21]

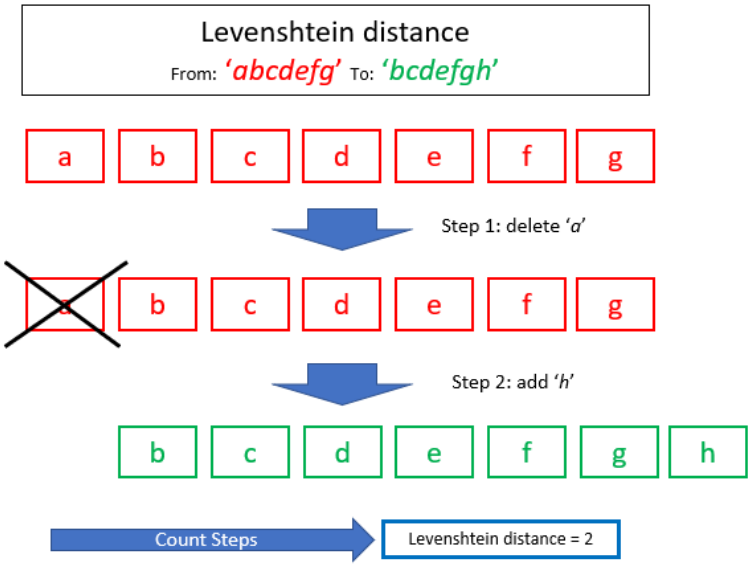


Figure 2.3: Levenshtein Distance of two strings with shift [21]

As seen in the Figures above (2.2, 2.3), the two sequences "abcdefg" and "bcdefgh" have a Hamming distance of seven and a Levenshtein distance of two. In this case, the Levenshtein distance is much less than the Hamming distance. This is because of the consideration of deletion, insertion and substitution. The two given strings only differ by one character which causes a shift in the string, and results by only comparing each character by a much higher score. In cases where shifts can occur often for example in nucleotide sequences the Hamming distance will calculate much higher scores than the Levenshtein or other distances, which consider shifts within the string. Approaches like the Levenshtein distance which considers different positions and scores the minimal number of corrections within a string, results in lower distance scores, which reflects in some cases the wanted threshold better [7, 12, 16, 30].

2.1.4 Damerau distance

The Damerau distance, often also referred as Damerau-Levenshtein distance was invented by Frederick Damerau and Wladimier Levenshtein. Similar to the Levenshtein distance, it measures the edit distance between the given strings. Therefore the minimal number of corrections has to be found in the string. Again, the used operations are substitution, insertion and deletion. Additionally, it is possible to transpose characters, within one of the two strings. These operations are applied on each character of the string. Instead the supplementary operation, the transposition, requires two adjacent characters. Because of the additional option of transposition, the algorithm is often used in a biological context, especially for DNA sequences, which also undergo transpositions, insertions, substitution and deletion. These events occur frequently and are extremely relevant in analysing a variety of nucleotide data sources. Although transposition is an additional operation within the calculation, the runtime complexity is equal to the Lev-

enshtein distance with a complexity of $\mathcal{O}(n * m)$ with n and m as the length of each string and therefore the size of the matrix. As seen in the formula below, similar to the Levenshtein distance, a matrix is build by using dynamic programming. The calculation differs to some extend by adding an additionally calculation step which uses the score at the matrix at position $D_{i-2,j-2}$ as the transposition [17, 18]. (2.5)

$$D_{i,j} = \min \begin{cases} 0 & \text{if } i = j = 0 \\ D_{i-1,j} + 1 & \text{if } i > 0 \text{ (Deletion)} \\ D_{i,j-1} + 1 & \text{if } j > 0 \text{ (Insertion)} \\ D_{i-1,j-1} + 1 & \text{if } i, j > 0 \text{ (Substitution)} \\ D_{i-2,j-2} + 1 & \text{if } i, j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j, \text{(Transposition)} \end{cases} \tag{2.5}$$

2.1.5 Jaro distance and Jaro-Winkler distance

Similar to the previously described distance metrics the Jaro and Jaro-Winkler distance calculates the edit distance between to strings. The Jaro distance provides a value between zero and one. As seen in the formula below the value is normalized by the length of both strings and the number of transpositions within the two strings. The result value between zero and one represents the ratio of similarity of both strings. An exact match is represented as one [26, 27].

$$sim_j = \min \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \tag{2.6}$$

For the calculation of the Jaro-Winkler distance the formula is extended as follows:

$$sim_w = sim_j + \mathrm{lp}(1 - sim_j) \tag{2.7}$$

The Jaro-Winkler distance therefore uses the calculated sim calculated from the Jaro distance. The parameters m is the number of matching characters within the string s. The Jaro-Winkler similarity uses additional parameters l and p. The parameter p represents a scaling factor for adjusting prefixes and l which is the length of the prefix, which matches with the comparing string. Similar to the previous mentioned distance matrices the time complexity is $\mathcal{O}(n*m)$ with n and m the length of strings to compare and is therefore as all previously described distance metrics dependent on the string length [28].

2.1.6 Distance Metric Comparison

Calculating the distances is the first step in hierarchical sequence, clustering. The mentioned distances above are designed for the comparison of two sequences, and therefore the score is based on the character comparison for each sequence in this case nucleotide sequences. The iteration process and the comparison of each sequence pair concludes to

long runtime and increasing memory complexity. Especially for larger data-sets, in this example more than 5000 data entries, this may be problematic for further evaluation which is explained in detail in chapter 5. Packages which already include a distance scoring within the clustering algorithm and are specifically designed for biological sequence clustering commonly use a greedy approach or include domain specific knowledge based on databases or alignment scores. This requires a more detailed information about the data set and its source as well as the target outcome of the clustering itself. The distances mentioned above, except the Hamming distance, use the approach of minimal corrections within the sequences. The calculation of the Damerau and the Jaro-Winkler distance are the most complex, especially concerning the runtime, which will be discussed in further chapters [7, 12, 16–18, 26–28, 30].

2.2 Clustering Algorithms

Clustering is one of the most important approaches and established technique in unsupervised machine learning to identify certain patterns and similarities in not previously labeled data. Therefore, clustering does not require any depth knowledge of possible results. A wide range of clustering algorithms are used depending on the data set and the field of research. In a biological context, especially in sequence clustering, agglomerative clustering and hierarchical clustering approaches are widely established and supported by several frameworks, especially in Python.

The following chapter will focus on hierarchical clustering. Other clustering algorithms such as k-means or density-based clustering algorithms such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) are only described briefly in the following.

The goal of clustering analysis is finding values and clusters to determine similarities of groups, so called clusters. These groups share similarities and patterns and can then be used for further classification. There is a large variety of different clustering algorithms, which mainly differ in their definition of a cluster, called clustering model.

One of the most popular representative is the k-means algorithm. K-means requires a pre-defined number of clusters. Therefore, it is not able to find the number of cluster by itself but rather assigns values to one of the initialized clusters. The algorithm iteratively determines the centroid of the clusters and assigns a value to the nearest centroid. A cluster is thus defined by its centroid, which is, in this case, as the name suggests the mean of the values of a cluster. The process is complete as soon as all elements of a data set are assigned to one of the clusters. As similar approach is used for k-medians or k-medoids, the difference is the calculation of the centroid. Because the cluster is defined by its centroids these approaches are called centroid based clustering.

[2, P.76] In contrast, density-based clustering approaches, define their clusters as areas of densities based on their data set. A popular representative is DBSCAN. Other than k-means, DBSCAN does not need an initial number of clusters. Here the data is categorized in three different categories: core points, border points and outlier [11, P.72–276]. The core points determine the least amount of datapoints to define a cluster, the border points then the point which is reachable within the cluster, if a datapoint can not be defined two one of these categories, which means it is not assignable to a cluster it is defined as an outlier.

The focus of this thesis is hierarchical clustering which is a clustering algorithm, which merges the given data set until all data entries are merged into one cluster. Merging the data set means in this case combining two clusters into one. The merging step within the clustering process is defined by the linkage used between the data entries, which depends on the distance within the elements. Hierarchical clustering differs between divisive and agglomerative clustering. The main difference is the initial cluster which the clustering algorithm starts. Divisive clustering is based on a top-down approach and starts with all elements in one cluster and iteratively splits the clustered elements in smaller clusters. On contrary, agglomerative clustering uses a bottom-up approach, all elements are initially defined as a cluster which are iteratively merged to one cluster. [11, chpt. 7; 13, chpt. 7; 2, chpt. 8].

2.2.1 Hierarchical Clustering

For hierarchical clustering there are two necessary steps: first, a distance matrix has to be provided. The calculation of a distance matrix is explained in Chapter 2.1. Second, a linkage has to be defined which describes the merging step at each iteration and is necessary for the evaluation of the distance between clusters. The linkage is therefore an equally important part in hierarchical clustering and has to be chosen wisely. The resulting dendrogram depends on the distance matrix as well as the chosen linkage calculation. The most common linkage calculations are described in the chapter below. Each approach differs in its calculation of the clusters and the branch lengths, linking the merged clusters torwards the root cluster. The hierarchical clustering process can be explained in general with the following steps shown in Figure 2.4. Initially, each element of the data set is a cluster, which then can be compared based on the defined linkage. The datapoints with the highest similarity then will be identified and in the next step merged. As seen in Figure 2.4 the distances of the merged clusters then are recalculated. The process will be repeated until all clusters are merged into one cluster. [11, chpt. 7.3.2; 13, chpt. 7.1; 2, chpt. 8.2.2.2].

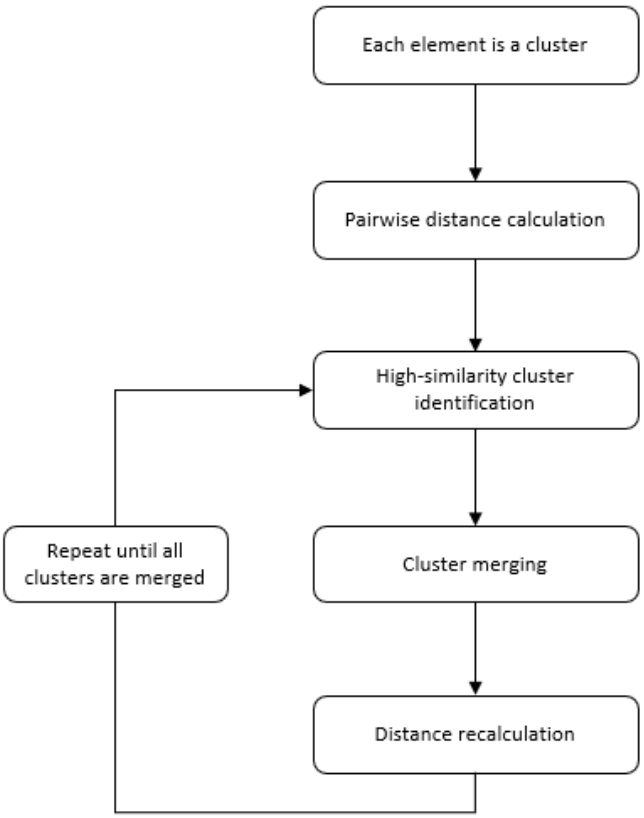


Figure 2.4: Flowchart hierarchical clustering with previous distance calculation

2.2.2 Maximum or complete linkage

The most common and easy linkage is the maximum or complete linkage. Each element starts with its own cluster. Then, each element gets merged with other selected clusters. This will be repeated until each element is in the same cluster, which is comparable with the root of the dendrogram. To do so firstly, the smallest value will be used as a cluster and is estimated for the branch length. The branch length therefore is calculated as the element of it divided by two. The distance matrix then can be updated by reducing it by its already clustered values. Similar to the first step the processing continues by using the updated distance matrix, with the merged clusters. The branch length estimation for the new cluster is the same as before, but it will be further reduced by the branch length of the previous clusters. The following process by finding the smallest number and adding it to an existing cluster will be repeated until all elements in the distance matrix are clustered into one cluster or, in other terms, no values in the distance matrix can be reduced anymore. The linkage calculation is described as:

$$D\left(C_g, C_h\right) = \max\left\{d\left(i,j\right)\right\}, i \in C_g, j \in C_h \tag{2.8}$$

Consequently, dendrogram branch length calculation depends on the branch length of the previous calculation and will be subtracted by every iteration [2, P.83-484; 13, P.24-427; 8].

2.2.3 Minimum or single linkage

Comparable to maximum linkage, minimum linkage is a bottom-up based method, which means it is also an agglomerative clustering algorithm. Each element is represented as its own cluster and will be combined by iterating until only one cluster, containing all elements left. Clusters with the shortest distance will be combined first. After determining clusters with the highest similarity, these clusters are merged.

$$D\left(C_g, C_h\right) = \min\left\{d\left(i,j\right)\right\}, i \in C_g, j \in C_h \tag{2.9}$$

This will be continued until all objects are in one cluster. (as shown in Fig. 2.4) In comparison to complete linkage described before the branches of the dendrogram are much shorter. This is caused by using only the shortest distances between the groups for later clusters. This can lead to the effect of chaining. Chaining is an effect, where certain clusters are merged at an early stage of the clustering process. As a result elements with high dissimilarity are merged into one cluster. However, this has the advantage, to identify outliers, within the clusters easier. This effect can be seen in Figure 2.5, which shows six outliers and three groups chained together in one cluster. [2, P. 80-482; 8].

2.2.4 Average linkage

Another, approach is the average linkage, which uses the arithmetic mean for its distance calculation. The arithmetic mean uses the amount of objects within the two clusters and multiplying it with the distance between each element of their clusters. Similar to the other linkage approaches the clusters are merged depending on the previous calculated distance matrix. The result is then the calculated branch length.

$$D\left(C_g, C_h\right) = \frac{1}{n_g \cdot n_h} \sum_{i \in C_g} \sum_{i \in C_h} d\left(i,j\right) \tag{2.10}$$

By using the formula the distance between the clusters C_g and C_h is calculated by using n_g and n_h, which represent the number of elements within the clusters and the sum of each distances between the clusters, represented by $d(i,j)$. The advantage of average linkage approaches is by compensating greater distances by only using the mean distance, this results in more monotone clusters [2, P. 80-484].

2.2.5 Weighted average linkage or WPGMA

Besides maximum and minimum linkage, other approaches focus on the calculation of the average linkage such as the Weighted Group Method with Arithmetic Mean (WPGMA). Here, in a first step the minimum value of the already processed distance matrix is selected and divided by two to calculate the first level of branches. Then,

the distance matrix is updated by merging the first calculated cluster and reducing its size. Additionally, the value of the matrix will be updated, which means that the distances are recalculated (as shown in Figure. 2.4). Then the process will be repeated by every iteration until all elements of the matrix are clustered into one branch. The final dendrogram has compared to complete linkage much shorter branches because of a different calculation of the branch length, by using the formula above. The WPGMA uses for each step the average of the distances, without weighting them, the term weighted average refers to the result [10].

2.2.6 Unweighted average linkage or UPGMA

While WPGMA works with unweighted link calculation, within each step, the Unweighted Pair Group Method with arithmetic mean, weights within the number of taxa within the cluster each clustering step by. This results to an equal contribution of each distance [32]. The first step is like other linkage function, by finding the minimum, as well as the calculation of the first branch. In comparison to WPGMA the remaining values of the distance matrix will be updated within the next iteration using the reduced matrix. The approach of UPGMA and WPGMA is similar to average linkage clustering. [10].

2.2.7 Ward linkage

As a last exemplary linkage calculation, the ward linkage differs to the minimum linkage and maximum linkage by merging the clusters as well as by calculating the distances to the next merged cluster. The linkage approach is based on the formula:

$$V_g = \sum_{k=1}^{m} \sum_{i \in C_g} \left(z_{ik} - \bar{z}_{gk} \right)^2 \tag{2.11}$$

and not on the nearest or smallest groups, like single or average linkage. The merging depends on the heterogeneity of the groups within the clustering process, the linkage is therefore based on the clusters, which less increase the variance of the overall clustering process. The formula above describes the variance within the clusters and merges them based on the mean within each cluster, described by z_{gk}. As a result a clustering process with homogeneous clusters within the process is build. [2, P.84–488].

2.2.8 Linkage Comparison

The selection of different linkage metrics depends on the and the research question. Additional insight into the data, especially on the variance of particular objects, can be useful by selecting specific linkage algorithms. The clustering result highly depends on the merging step, by choosing maximum, minimum linkage etc, especially for larger data sets. Linkage can be classified whether a chain within the clustering process is build, meaning that single objects will merged into larger clusters within the clustering process. Distances, such as the single linkage clustering, tend to chain certain objects, which means more distinct clusters are merged in an early clustering step. As a result, members

which are more dissimiliar are merged in one cluster. However, it has the advantage that outliers are easier to detect, as shown in Figure 2.5. Complete linkage in contrast tends to create smaller groups within the merging steps by calculating the largest value of each distance, shown in Figure 2.6. The Ward linkage is one of the most established methods, and is proven to have the best portioning in most cases. Nevertheless, before using the Ward method for linkages it should be considered whether the data set has outliers and how it impacts the outcome the clusters, shown in Figure 2.7. Additionally, all variables have to be uncorrelated to each other as well as the application of the variance criteria. Therefore, when using the Ward linkage, more insight into the data is necessary. This can be achieved by preprocessing the data with other clustering methods, such as single linkage. [15]

Comparing the different dendrograms of single, complete and Ward linkage; the single linkage approach chaines many objects during the first clustering step, compared to the ward linkage. This shows the ability to find outliers within the clustering. The complete linkage approach clearly shows three clusters within the dendrogram but only group C is isolated and does not interfere with other groups, which means average linkage is in this example not able to isolate all groups. Clustering with Ward linkage is in this example the only linkage, which is able to identify all expected clusters, nearly without chaining within the clustering steps [22]. In conclusion, the example shows that for data analysis using hierarchical clustering, single linkage clustering approaches are useful for outlier detection and to get deeper insight of the data. For further grouping with Ward linkage for cluster detection without outliers is recommended, because Ward linkage is not as susceptible for chaining as single linkage [11, chpt. 7; 13, chpt. 7; 2, chpt. 8; 8; 10].

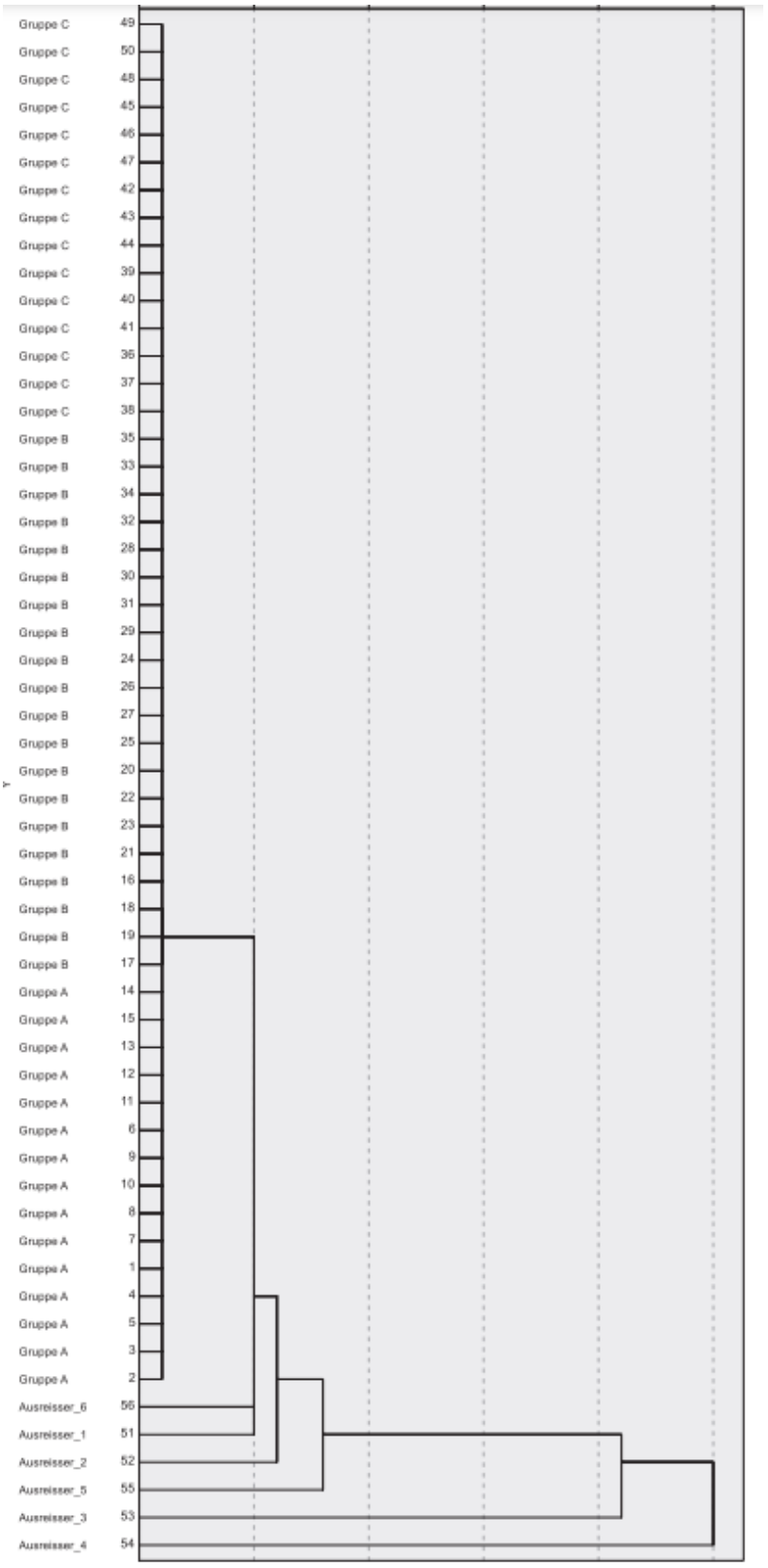


Figure 2.5: Dendrogramm with single linkage hierarchical clustering [19]

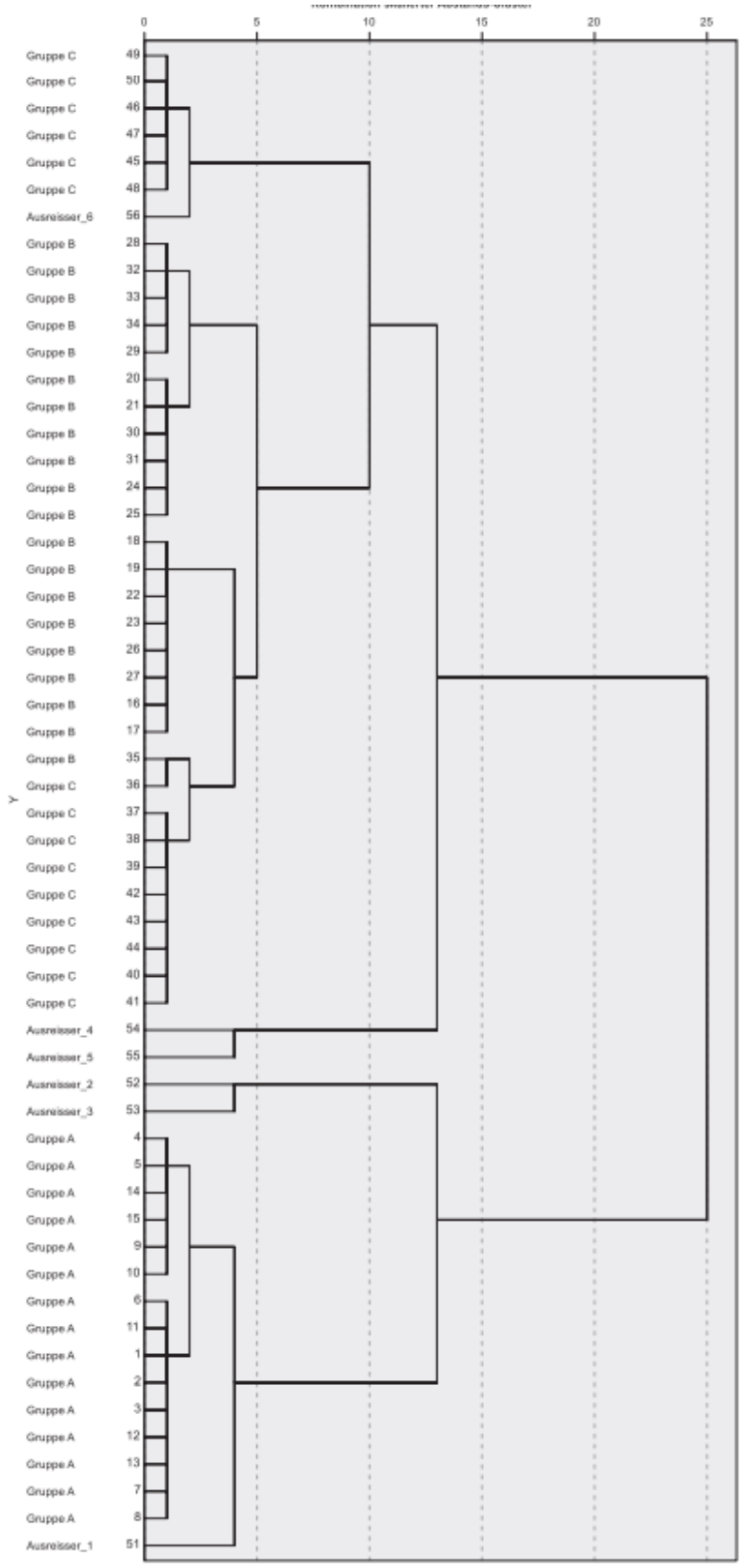


Figure 2.6: Dendrogramm with complete linkage hierarchical clustering [19]

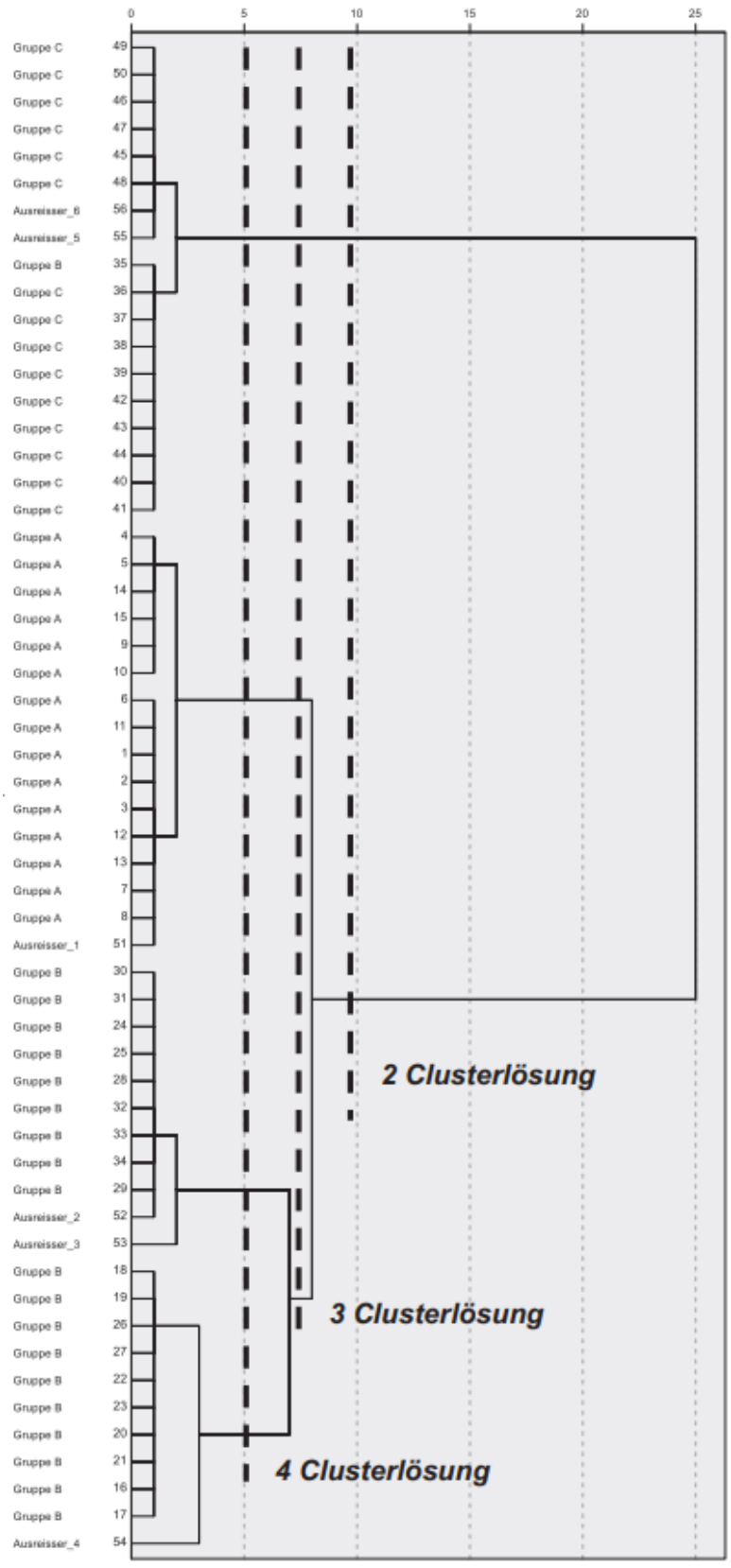


Figure 2.7: Dendrogramm with ward linkage hierarchical clustering [19]

Chapter 3

Packages and Algorithms

The following chapter describes the possibilities of certain packages in python, which are theoretically described in chapter 2. Furthermore some functionalities are implemented in chapter 4 for a given data set.

3.1 Jellyfish

Jellyfish is a python package which provides the most common distances for string comparison. Additionally, phonetic encoding distances, which won't be used in the following implementation. The package is freely available on GitHub. The current version is 0.9.0 and was released on January 7th, 2022. The package provides the distance metric calculation which then can be sued as a function parameter. The following distances for string comparison are implemented and used for the implementation:

- Hamming Distance
- Levenshtein Distance
- Damerau-Levenshtein Distance
- Jaro Distance
- Jaro-Winkler Distance
- Match Rating Approach Comparison

The Match Rating Approach is not described in chapter 2.1, reasoned for not relevant application in a sequence clustering process. The functions provided from the package return a numeric value defining the distance between two strings, for the calculation of the given data set an comparison of each sequence is necessary.

```
from jellyfish import hamming_distance,
d = hamming_distance(s1, s2)
```

In the example the jellyfish package imports the Hamming distance and calculated the distance of two strings, s1 and s2.

3.2 Biopython

Biopyhton is a well established and commonly used package in bioinformatics. The package provides many functionalities besides sequence clustering. In the following chapter

a possible approach of sequence clustering using Biopython is explained, as well as other clustering algorithm provided by the package [4].

3.2.1 Distances

Biopython uses the module `Bio.Cluster` for the most common clustering algorithms and their distance functions. The module is specifically designed and mainly used for biological applications, especially, sequencing data and their gene expression. The input format must be a n x m Numerical Python array. It should be noted that if certain values are missing these values will be ignored for the further analysis [4, chpt. 15].

Biopython provides a variety of already implemented distance operations, which can be used for calculating the distance matrix but are not necessarily required to commit to all functions, which mainly use the Euclidean distance by default:

- Euclidean distance 'e'
- City-block distance 'b'
- Pearson correlation coefficient 'c'
- Absolute value of the Pearson correlation coefficient 'a'
- Uncentered Pearson correlation (equivalent to the cosine of the angle between two data vectors) 'u'
- Absolute uncentered Pearson correlation 'x'
- Spearman's rank correlation 's'
- Kendall's τ 'k' [4, chpt. 15.1]

The mentioned distances will be used for distance calculation within the agglomerative clustering step, not as a distance between the sequence calculation. The reason is that the mentioned distances, which Biopython provides only can be used for numeric values not for string comparison. Additionally, a weight vector can be implemented, which increases the occurrence of a certain value in the data set. By increasing the occurrence of certain values, the relevance of clusters containing these values increases accelerates.

The function `distancematrix` calculates the distance matrix with the following interface:

- data: The argument data is required for running the function and is represented as an array
- mask: Mask can be used to filter missing data, therefore, if the attribute is defined as None all data is presented.
- weight: is an optional weight vector which can be defined
- transpose: The function can determine whether the distances between rows or columns can be calculated. The argument transpose = False determines the distance between rows, the argument transpose = True between columns.
- dist: Defines the distance function using the identifiers mentioned above,

Furthermore `Bio.Cluster` provides functionalities to calculate properties on clustered data itself, such as calculating the centroids of each cluster and the distance between clusters. For example the distance between centroids can be calculated with:

```
cdata, cmask = clustercentroids(data)
```

As implicated in the example above the function returns a tuple, which represents a 2D Numerical Python array.

The function arguments are the obligatory data and the optional parameters mask, clustered, method and transpose. The semantic of transpose and mask is as a user point of view likewise the semantic the semantic of the `distancematrix` function. The focus of the function is hereby the method used to calculate the centre which can be chosen between mean (`method == a`) or median (`method== m`).

Biopython provides hierarchical clustering, k-means/medians/medoids, self-organizing maps and principal component analysis as an approach for partitioning clusters [4, chpt. 15].

3.2.2 Hierarchical clustering

For hierarchical clustering, Biopyhton provides the treecluster method of the Bio.Cluster package with the following interface:

```
Bio.Cluster.treecluster(data, mask=None, weight=None,
transpose=False, method='m', dist='e', distancematrix=None)
```

The parameter data is required either as an array or as an already calculated distancematrix. The latter has to be defined specifically in the interface as distancematrix=distance. Except for the methodparameter, the arguments which passed to the function are semantically equal to other functions in the Bio.Cluster package. However, hierarchical clustering provides four different methods for cluster partitioning:

- pairwise single linkage clustering: `method== s`
- pairwise maximum linkage clustering: `method== m`
- pairwise average linkage clustering: `method== a`
- pairwise centroid linkage clustering: `method== c`

If an already existing distance matrix is used, only pairwise- , single-, maximum-, and average-linkage clustering can be performed but the dist argument lapses. It should be noted that the distance matrix may be deep copied or saved before assigning it to the function, because the clustering algorithm may reorder the values within the distancematrix.

Another approach is the `Phylo` package, which provides several methods for constructing trees and performing distance calculation. This package is also provided in Biopython as a separated package to the previously explained package. One example is the `upgma` method which constructs phylogenetic trees using the UPGMA algorithm. The advantage of using the `Phylo` package and its clustering methods over the treecluster method from the `Bio.Cluster` package is the easy visualization. Visualizing `Tree` objects provided by the `Bio.Cluster` package is not implemented. In comparison to the `upgma` method from the`Phylo` package, visualization in ascii on the console or in matplotlib is possible. The main differences are that the `Phylo` package is only implemented for phylogenetic trees and not for hierarchical clustering in general [4, chpt. 13; 3].

3.2.3 k-means, k-medians and k-medoids

`Bio.Cluster` provides two different functions for calculating k-means, k-medians and k-medoids. The method `kcluster` can either perform k-means or k-medians clustering and returns a tuple, where the clusterid is represented as an integer array of the assigned cluster of each row, an error which is defined as the sum within the cluster and a variable which represents the number of times the optimal solution is found. The parameters `data` , `mask`, `weight`, transpose and dist are defined equal to other function of the `Bio.Cluster` package with a similar interface. It differs by the assignment of the number of clusters, by default two and the method selection, by default k-means. The methods k-means is defined as `method== a` and k-medians as `method== m` . kmediods differs especially in case of the initial input data parameter, as far as it uses a already existing distance matrix. [4, chpt. 15.3]

3.2.4 Example

The above mentioned methods as well as their associated input parameters and are represented as an example as followed:

```
1 from Bio.Cluster import distancematrix, clustercentroids, treecluster, kcluster
2 import numpy as np
3
4 sequence = [ 'GGCGGGGATG','ACGTCTAGAC','CAGTTGAATG','GACGGCCCGG','GGGCGCAAAG']
5 matrix = get_matrix(sequence)    #generates matrix
6
7 distances = distancematrix(matrix, dist='e')
8
9 cdata, cmask = clustercentroids(matrix)
10
11 tree_c = treecluster(matrix)
12
13 clusterid_a, error_a, nfound_a = kcluster(matrix, method='a') #means
14
15 clusterid_m, error_m, nfound_m = kcluster(matrix, method='m') #median
16
17 clusterid_som, celldata_som = somcluster(matrix)
18
19 c_mean, coord, comp, eigenv = pca(matrix)
20
21 pip install grein_loader
22
23
24 geo_accession = "GSE112749"
25 description, metadata, count_matrix = grein_loader.load_dataset(geo_accession)
26
27
28 number_of_datasets = 10
29 overview = loader.load_overview(number_of_datasets)
```

The above code example shows how to use the methods which the Biopyhton packages provdes. It is assumed, that a distance matrix is already provided and can be calculated with the method in line 5.

3.2.5 Visualization

A common representation of the clustered data and their similarities is with the usage of the `Bio.Phylo` library, which is used to visualize an already existing source independently as a phylogenetic tree. As a method mainly used for visualization and representation, a tree is commonly already constructed and can be read from various file formats. The default output is a simple ASCII-art dendrogram. Better graphical frameworks, such as matplotlib or pylab, can include the quality of the visualization and exploration, as well as colouring methods within the tree can be increased, as shown in Figure 3.1. Like `SeqIO` and `AlignIO`, which are Frameworks in Biopython for file handling which is managed by four basic functions; parse, read, write, and convert. Furthermore, the library provides methods for traversing and searching nodes within the existing tree, as well as adding specific clade attributes. `Bio.Phylo` is mainly used for data visualization and representation and basic traversing. Alternatively, `Bio.Cluster` provides the class `Node` for tree representation, which can define nodes and their subnodes, with basic attributes, left, right and a optional distance, defined as an integer. Therefore, a hierarchical clustering result, represented as Tree can be used to define nodes for the tree representation.

Subsequently, the tree can be drawn with the support of other libraries and packages such as Treeview or Java Treeview. The function `treecluster` also returns a treeobject, containing nodes which can be visualized equally [4, chpt. 5, 13, 15].

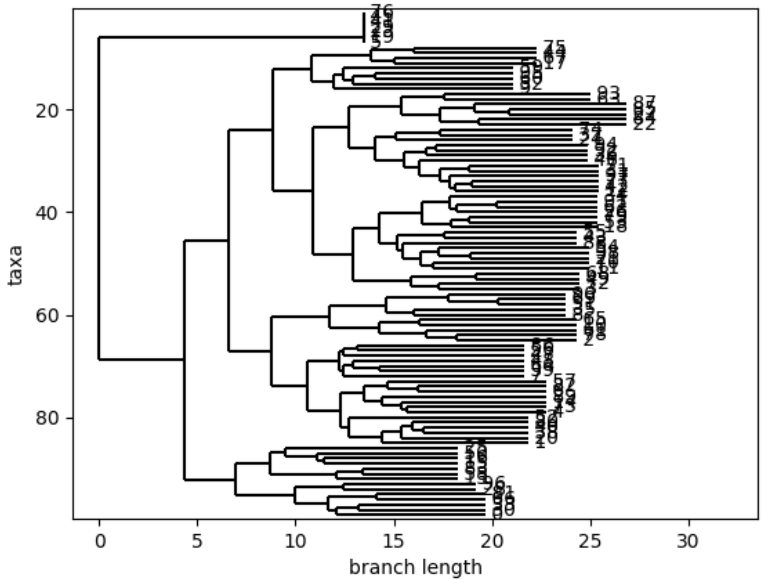


Figure 3.1: Phylogentic tree using UPGMA and Hamming distance with 100 sequences

3.3 scikt-learn

Another package which provides clsutering methods is scikit-learn. Sckit-learn provides with `sklearn.cluster` a module that includes several clustering methods. For each clustering method a distance matrix has to be provided. Therefore, distance matrices and features have to be selected before the method is executed. For feature extraction and generation of the distance matrix, sckit-learn provides further packages such as, `sklearn .feature_extraction` or `sklearn.metrics.pairwise` for preprocessing the data.

Further distance metrics in a more biological context are provided by the `Scikit.bio` package which has to be installed and included separately. For the approaches mentioned in chapter 2.1 a function is provided which uses the algorithm. The theoretical aspect of most of the relevant methods are explained in the chapter 2.2. The following clustering methods shown in Table 1 are already provided by the `sklearn.cluster` package: [22]

Algorithm	Function
k-means	KMeans()
Affinity Propagation	AffinityPropagation()
Mean Shift	MeanShift()
Spectral Clustering	SpectralClustering()
Hierachical Clustering	AgglomerativeClustering()
DBSCAN	DBSCAN()
OPTICS	OPTICS()
BIRCH	Birch()

Table 3.1: Clustering methods scikit.cluster

k-Means clustering algorithm requires several clusters, the function defines 8 initial centroids for clusters by default. The result of k-means clustering is highly depending on the initialization of the clusters, by using `k-means++` the initial centroids are more distant from each other, which is in some cases referable than a random initialization. In other cases, the iteration can be called several times with a random cluster iteration. With the parameter `N_init` the algorithm can be run with different centroid initialization using the initialization method defined before. The default parameter in this case is 10. Furthermore, with `max_iter` a maximal number of iterations within the calculation can be defined. Another important parameter is `algorithm`, the k-means algorithm can be extended with certain implemented features, or is by default `auto`. Other possible parameters are `full` or `elkan`. The elkan k-means is an extension of the classical k-means, by applying the triangle inequality. By using the triangle inequality and defining two centroids as a point within the triangle an element can be assigned to a cluster. Therefore it is defined $d(x, c) \geq d(x, b)$ subsequently the algorithm can use the following approach by assigning an element to a cluster.

3.3.1 Hierarchical clustering

The most notable method in terms of sequence clustering is hierarchical clustering. `Scikit.cluster` provides four different approaches: ward-, maximum/complete- average-

and single linkage. The clustering method will be applied with the AgglomerativeClustering object and uses a bottom up method within the nature of the algorithm. Additionally to a preproceeded data matrix the function `featureAgglomeration` groups features on certain criteria together and therefore can be used for dimensional reduction. The application of different approaches in agglomerated clusters can vary depending on the behaviour of the data set. [**5**] [**6**] The function `skleran.cluster.AgglomerativeClustering` is therefore build as follows:

```
class sklearn.cluster.AgglomerativeClustering(n_clusters=2, *, affinity='euclidean',
memory=None, connectivity=None, compute_full_tree='auto', linkage='ward',
distance_threshold=None, compute_distances=False)
```

The affinity hereby is the metric used for the linkage and also can be precomputed, in this case an already existinc distance matrix is necessary. Additionally to the metric the linkage can be passed as an parameter and will perform the depending algorithm as described above.

The Agglomerative Clustering method only provides a model according to the defined parameters and can then be fitted to a data set, in this case a precalculated distancematrix. The distance matrix is always required. The AgglomerativeClustering function defines the affinity and the necessary paramters for a modell, where the data will be applied. The following .fit function then uses the matrix and returns a finished model, which can be visualized. Scikit does not provide a plotting function by itself, but is referencing to variuous implementations methods provided in the documentation. Here, matplotlib is used for visualization.

3.3.2 Scikit-bio

The package includes numerous features for biological sequences as well as distance functions, which is by default the hamming distance. Furthermore, relative frequencies can be calculated with the `matchfrequency` method, as well as many more distances and sequences within the `skbio.sequence` package. [**7**]

3.4 Sequence Clustering Tools

The high importance of efficient sequence clustering in the field of genomics and proteomics lead to the development of various tools. The development of highly specific algorithms provides various advantages. In the following the most used and established tools will be described.

3.4.1 UCLUST

UCLUST is specifically designed for clustering nucleotide sequences. The algorithm uses a greedy approach, which results in different evaluations depending on the order of the input sequences. Therefore, a sorting of the sequences by length is widely advised using the UCLUST algorithm and package. UCLUST defines two criteria for finding a set of clusters in a data set: first, all centroids have a similarity smaller than T, which is defined by a identity threshold, to each other and second, the member sequences for each cluster have a similarity higher or equal to T. The defined thresholds can be viewed

as the radius of a certain cluster, which centroid is representative for a sequence with highest similarity. The sequences then will be aligned to the representative sequence of the cluster and will be assigned to the cluster with the highest score, defined by T. UCLUST is a widely used algorithm in bioinformatics, especially, in terms of operational taxonomic unit (OTU) clustering. Due to its fast performance it has a big advantage compared to other implementations such as webbased implementations or packages in Python, where distance matrix calculation has to be performed [1, 9].

3.4.2 CD-HIT

Another sequence clustering tool is CD-HIT, a framework for amino acid and nucleotide sequence comparison. It was introduced and developed by the Stanford-Burnham Medical Research institute and first published in 2001. The package includes various scripts for different tasks, but it is mainly focused on finding a representative consensus sequence and not on phylogeny. The package can be easily installed via GitHub and included in a Python library. Additionally, correspondent online tools can be used, with a decline in performance. Compared to hierarchical sequence clustering in Python, CD-HIT is much faster and more performant, because it is, similar to other tools, specifically designed and optimized for sequence clustering of large data sets. CD-HIT uses similar to UCLUST a greedy approach on the sequence comparison, which results in a faster all-by-all comparison. The algorithm therefore sorts the input sequences by length. The longest sequence is then selected as the initial representative sequence, and therefore representative for the first cluster. In the next step, each sequence is compared to the representative sequence and its distance compared with a threshold T. Similar to other clustering algorithms, a new cluster is defined, depending on the distance to the threshold. The distance is then defined as a representative sequence for the new cluster. The comparison is depending on a specific number of identical dipeptides, tripeptides, and continued to a predefined window. These are molecules which consist of a defined number of amino acids.

The sequence similarity then depends on the similarity of the substring of a sequence. Additionally, an index table for string comparison is build. For each comparison, the number of possibilities, in our case nucleotides, with four different letters, exponent with the size of the instance sequence to compare. Therefore, the index table represents short words to compare, which is in terms of performance as well as memory complexity much more efficient. The greedy approach, the comparison of subsequences and the index table are the reason for the resulting performance of CD-HIT compared to other packages [5, 6, 9, 24, 25].

3.4.3 Online Tools

In addition to standalone software frameworks and libraries, various online tools from different companies and research groups are published. One example is Clustal Omega from European Molecular Biology Laboratory (EMBL), which can be used for clustering amino acid sequences, DNA and RNA sequences. It is possible to upload seqeunces in different file formats. Furthermore, it is possible to set different representation formats, for aligning the sequences. The main difference is that the distances are based on the sequence alignment scores and not on a distance matrix by comparing each sequence. On

the contrary, the online tool limits the amount of sequences by 4000 sequences which is far too less for most sequence data sets, especially in a bioinformatic context. Clustering results are visualized as a phylogentic tree and information about the alignment, which makes calculating a consensus sequence easier, than in hierarchical clustering.

3.5 Comparison

Agglomerative clustering methods are already provided in many ways in numerous python packages. The decision of which to choose is highly depending on the usecase and factors such as, the size of the data set, the algorithm for the clustering and the expected result. Scikit-learn provides a wide range of functionalities in agglomerative clustering and is easy to implement. However, it lacks by providing analysis methods which can be applied on the data set after hierarchical clustering. The main advantage over Biopyhton is the easy visualization of the data. Biopython provides a more extensive package with Bio.cluster. Not only by allowing distance matrix calculations, but also by analysing the clusters itself. The disadvantage is the lack of visualization opportunities within the package. In comparison to Pyhton packages, tools such as UCLUST, and CD-HIT are fast in executing and highly specialized on the task, which gives them in some cases an advantage over python packages. Nevertheless, they require already insight of the data for analysing. Functionalities like analysing clusters or visualizing the data are not provided within the packages themselves. Online tools have the advantage of easy usage and relatively quick results, but commonly have limit on data entries which can be processed. Therefore, the application on larger data sets, in this case more than 5000 sequences is not relevant. Online tools therefore can be almost excluded from applying in any form by agglomerative clustering for biological sequences, but might give first insights into smaller data sets. Additionally, online tools commonly provide additional analysis methods or visualization packages [1, 9].

Chapter 4

Implementation

The implementation is mainly focused for analysing the generated results with the python libraries and packages mentioned in the chapter 3. The UML diagram 4.1 shows the implemented process for a given data set. The realization hereby consists mainly on classes which are focused on one specific task or within the problem domain. The clustering process extends to specific libraries within the clustering process which will further conduct the clustering algorithm by itself. The implementation therefore addresses hierarchical clustering approaches on scikit-learn and Biopython, and its metrics. Additionally, a class for distance metrices is implemented, which implements the distances mentioned in chapter 2, by using the jellyfish package and its mentioned distances for biological string comparisons. The implemented classes for sequence clustering, scikit-learn and Biopython then use the calculated model of the distance matrix for further clustering.

Furthermore, the results as well as the runtimes will be saved in a datacollector class and further analysed and visualized, using Matplotlib and certain other tools, like excel spreadsheet, which is enabled by a csv writer within the data collector. A data reader uses filtering features and file readers for FASTA formats and .csv formats to provide the data.

The provided data set consists of NGS data derived from blood of patients suffering from chronic myeloid leukemia (CML). All sequences are tagged with unique molecular identifiers (UMIs). UMIs are short (here: 10 nucleotides long) oligonucleotides used as barcodes while sequencing. These short nucleotide sequences are attached to DNA or RNA strands before polymerase chain reaction (PCR). While PCR, all reads are amplified, which means that (in theory) every DNA strand is copied in every PCR cycle. Technically, not every strand is amplified in every cycle, which means that the PCR product which is sequenced afterwards does not consist of the same amount of copies for each initial DNA template. UMIs are used to trace back to the initial DNA input template. Therefore, UMIs are used to identify true variants/mutations and allow for correcting errors that occurred while PCR and NGS [14].

The data set provides the extracted UMIs from 120 000 sequenced DNA strands of one patient. In this thesis, these UMIs serve as input data for the various clustering algorithms and distance calculations. The implementation as well as some test cases are available on Github
(https://github.com/alexgrent/AgglomerativeSequenceClustering)

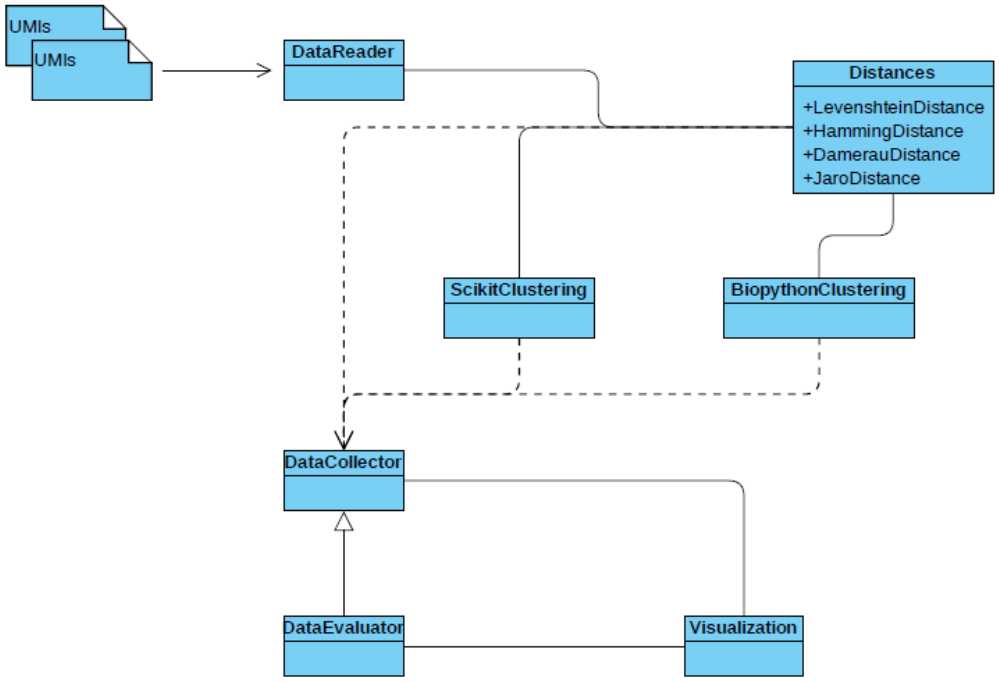


Figure 4.1: UML diagram for sequence clustering in Pyhton using UMIs as input value

Chapter 5

Results and Discussion

The results section includes the runtime of the distance calculation, the overall performance of each step within the sequence clustering process, distance calculation and linkage calculation, and the overall clustering process, as well as the result of hierarchical clustering, with different packages, distance metrics and linkages. Based on the results the packages can be evaluated whether the packages are suitable for the specific data set. The results will be comapared with other established methods and packages beside sequence clustering in python.

5.1 Distances

Distance calculation is the first step of hierarchical sequence clustering the relevance. Is already mentioned in chapter 2.1 and is therefore necessary for further analysis and clustering. As shown in Figure 5.1 the runtime of the distances is highly depending on the amount of sequences which are compared. The runtime increases by larger datasets. All distances, except the hamming distance have a clear peak at 5000 data entries, where the runtime increases sharply, which makes a calculation in Python with a larger data sets, in this case with more than 5000 sequences, in terms of runtime less suitable.

As shown in Figure 5.2 the runtime of Hamming distance increases in comparison to the Levenshtein significantly more after 8000 sequences. The overall runtime is therefore dependend on the distance matrix, it is shown that the usage of different distance metrics affects the runtime, especially on larger data sets.

Furthermore, the distance calculations each sequence has an effect on the runtime but not as dramatically as the amount of sequences. Here, the Damerau, Jaro and Jaro-Winkler distance which use different methods but do not differ as much in terms of runtime as the Levenshtein or Hamming distance. Shown in table 5.1

The following figures 5.3 show a prediction of the runtime based on the calculated data, it is shown that the runtime increases on at least a quadratic bases. It should be noted that the following graph is only an estimated prediction of the runtime, which is based on a polynomial function based on the tested data set. The extrapolation is not verified or tested in any way, thus the prediction should only be interpreted under the reservation that a few samples are used as a representation of the polynomial function.

Although, the graph looks similar for each distance calculation the difference of the

runtime differs greatly, as seen on the scaling of the y-axis. The table below shows the predicted runtime for 120 000 sequences which is the size of the given data set.

As shown, the runtimes, the predicted as well as the actual runtimes from the test cases above, are in terms of runtime extremely high. The methodical approach of calculating the distance of sequences by comparing each sequence in the data set is for this specific data set not suitable. The reason for the long runtimes is the comparison of each sequence in the data set, which is a huge disadvantage of non-greedy approaches. Other packages, explained in chapter 3.4, which use an greedy approach only use one iteration of the data set which results in much better performance, not only for the distance calculation but also for the overall calculation.

Distance	Seconds
Hammming	$3.0034*10^9$
Levenshtein	$6.24*10^8$
Damerau	$1.14*10^9$
Jaro	$2.47*10^6$
Jaro-Winkler	$4.84*10^6$

Table 5.1: Approximated runtimes for 120 000 sequences

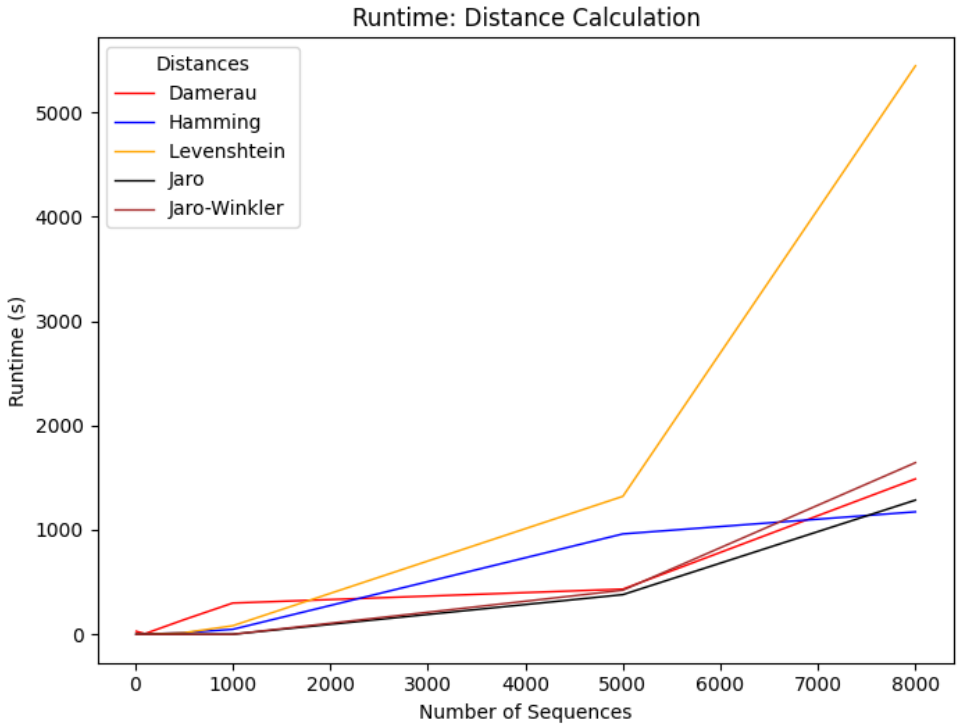


Figure 5.1: Resulting runtime for five different distance algorithms applied to increasing number of sequences

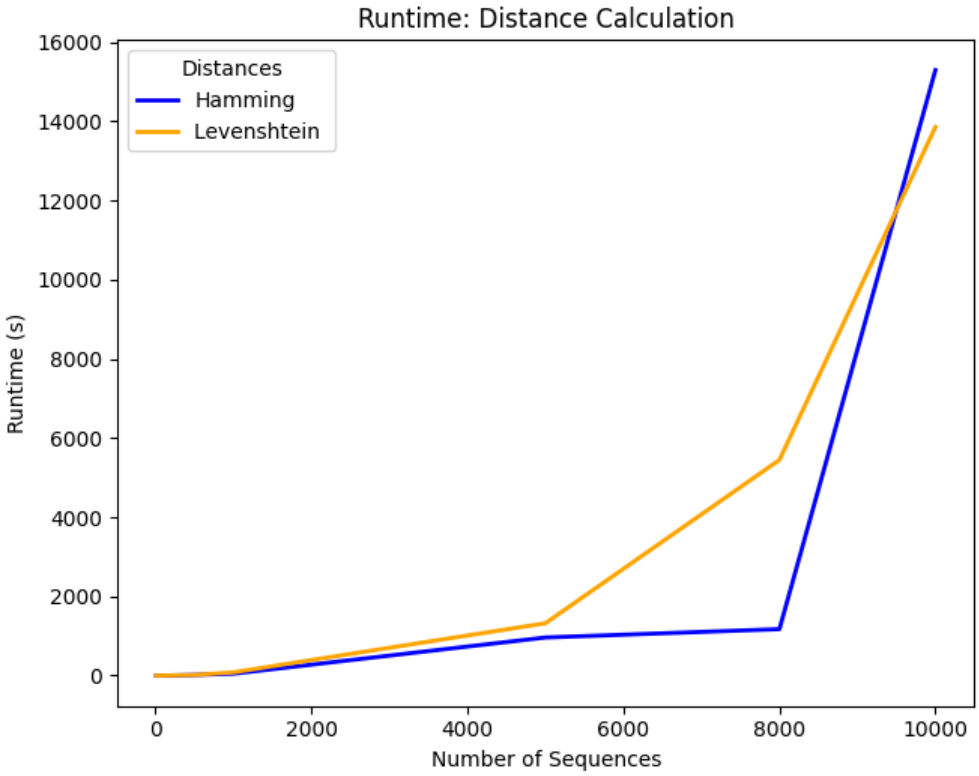


Figure 5.2: Comparison of the runtime of calculation Hamming and Levenshtein distance depending on the number of sequences

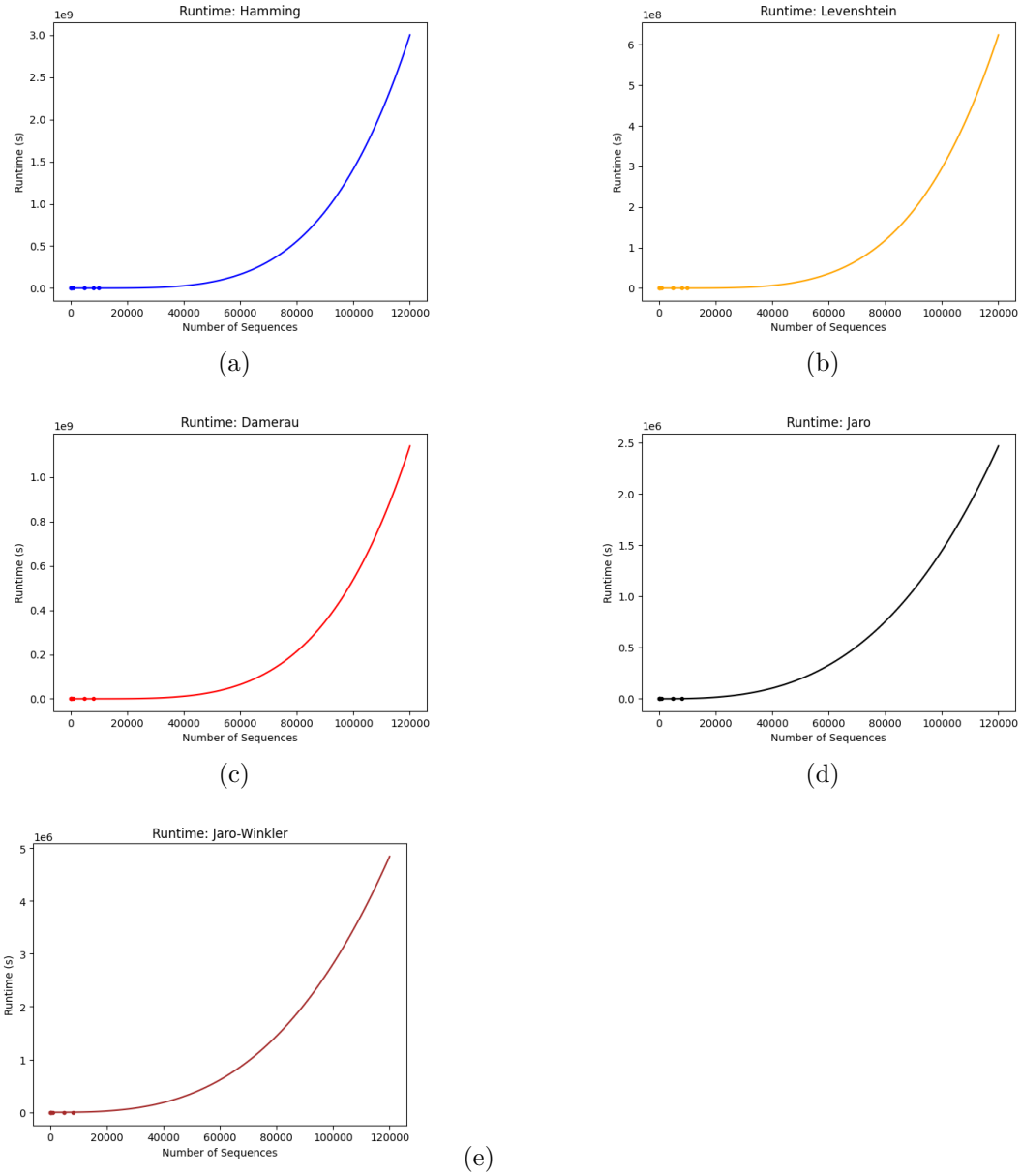


Figure 5.3: Extrapolation of possible runtimes for 120 000 sequences. *Hamming distance* (a), *Levenshtein distance* (b), *Damerau distance* (c), *Jaro distance* (d), *Jaro-Winkler distance* (e).

5.2 Clustering

The distance metrics have an effect on the clustering which is shown in the dendrograms in Figure 7.1, 7.2, 7.3, 7.4, 7.5. The distances explained in chapter 2.2 are calculated in the following dendrograms. The dendrograms clearly show a correlation between distance metric within the agglomerative clustering step and the clusters which are

build. In the following the Ward linkage, which is the default setting for the scikit-learn package, is used, as well as the dendrogram plotting from scikit-learn. The clustering has been tested with a data sets smaller than 1000 UMIs. All visualized dendrograms can be find in the chapter 7.

5.2.1 Influence of Distance on Clustering

As shown in Figures 7.2(c) and 7.4(c) the distance calculation method has an effect on the clustering of the sequences. Another good example is the comparison of the distances using the Ward linkage, especially Figure 7.1(d) and 7.3(d). The Hamming distance separates the clusters, with the Ward linkage differently to other more complex distances. The separated cluster in this case the green cluster in Figure 7.3(d), is in comparison much smaller compared to other distance metric results. For the Levenshtein, the green cluster contains more sequences, approximately 2/3 more than for the Hamming distance. It is clearly evident that the complexity of the calculation is relevant for the assignment to the clusters. As an example the Damerau distance, Jaro distance and Jaro-Winkler distance, show more details by their identification of their clusters. The analysis of clusters gets more complex by an increasing amount of data in the data set. A detailed analysis of data sets with more than 1000 sequences with dendrograms is not the preferred way.

In summary, the distances have an impact on the clustering process. Cluster definition in further steps is only possible by choosing the right distance for t he given data set. The linkage in the clustering process then limits the cluster representation further, but is more depending on the distances itself. Generally, many distances provide similar results, especially by larger data sets, nevertheless choosing the right distance calculation can be relevant for the result.

5.2.2 Influence of Linkage on Clustering

In addition to the distance calculation of the sequences, the linkage has impact on the clustering process. In comparison, the calculation of the linkage has not that strong effect on the runtime like the calculation of the distance matrix. Additionally, the outcome of a certain linkage is much more predictable as the outcome of distance matrix. Effects like chaining, in single linkage clustering, for example Figure 7.1(a) is already well researched and a well-known issue. The effect of chaining in this data set is shown in almost all figures using single linkage clustering independently of the distance matrix. The single linkage suggests an outlier within the data set and therefore can be used especially to target these. Other linkages provide more insight of the data itself. The complete linkage is not able to identify the outlier as a new cluster. The Ward linkage, which aims to minimize the variance of the clusters finds, compared to single, complete and average linkage main, clusters within the data set. As well as many subclusters with almost no outliers within the data set. Nevertheless, the oulier is still represented in the cluster itself, but is rather included, by using the Ward linkage. In conclusion, single linkage is useful by identifying outliers within the data set. Outlier might not be detected by using Ward linkage, reasoned by including almost all data entries to a cluster. On the other hand the ward linkage is able to identify clusters more concise, and provides more detailed information about the data distribution itself.

Chapter 6

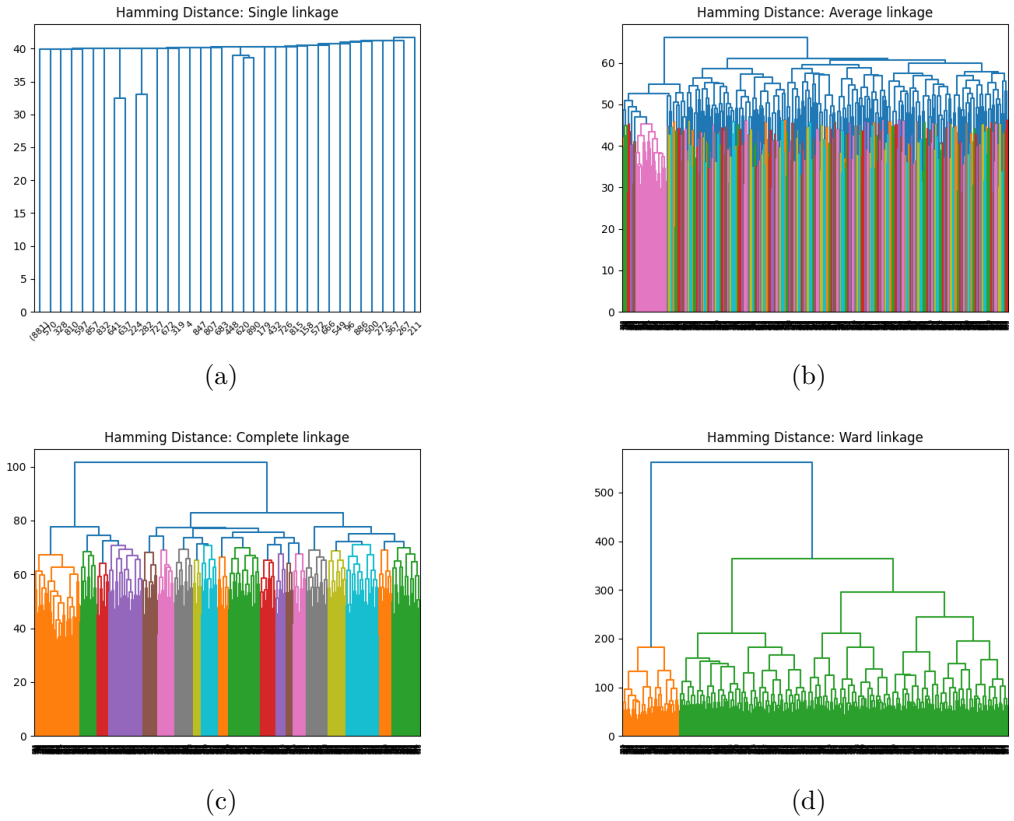
Conclusion

As shown in the runtime analysis of the distance calculation as well as the runtime of the clustering algorithms for each packages, none of the packages provide results for the given data set of more than 120 000 sequences with a convenient runtime. As a result for larger data sets all tested packages, Biopython and scikit-learn, do not provide a result on a realistic time scale and should therefore not be applied for this task. The increasing time complexity is caused by the the calculation of the distance matrix which is only possible by comparing all data entries, which results in an quadratic time complexity for the whole data set, although the time complexity for each comparison is in most cases O(m*n). By separating the distance calculation and the clustering algorithm a new iteration through the whole data set is necessary which results again in a quadratic time complexity. Although, the used packages are highly performant in terms of runtime, they are not suited for data sets with more than 10 000 data entries. Additionally, the usage of the scikit-learn or the Biopython package for hierarchical clustering in terms of finding a master sequence are compared to other packages or tools, not based on Python, not recommended.

In conclusion hierarchical sequence clustering using scikit-learn or Biopython has the advantages to get more insight into the data, with fewer data entries. It provides easy to use features for a first approach in terms of data analysis and getting an overview of the distribution of the clusters or certain outliers. It is also useful in terms of visualization or pyhlogenetic analysis of smaller data sets. The use of Biopython or scikit-learn in terms of finding a representative sequence is in comparison to other packages, explained in chapter 3.4, not recommended due to the long runtime.

Chapter 7

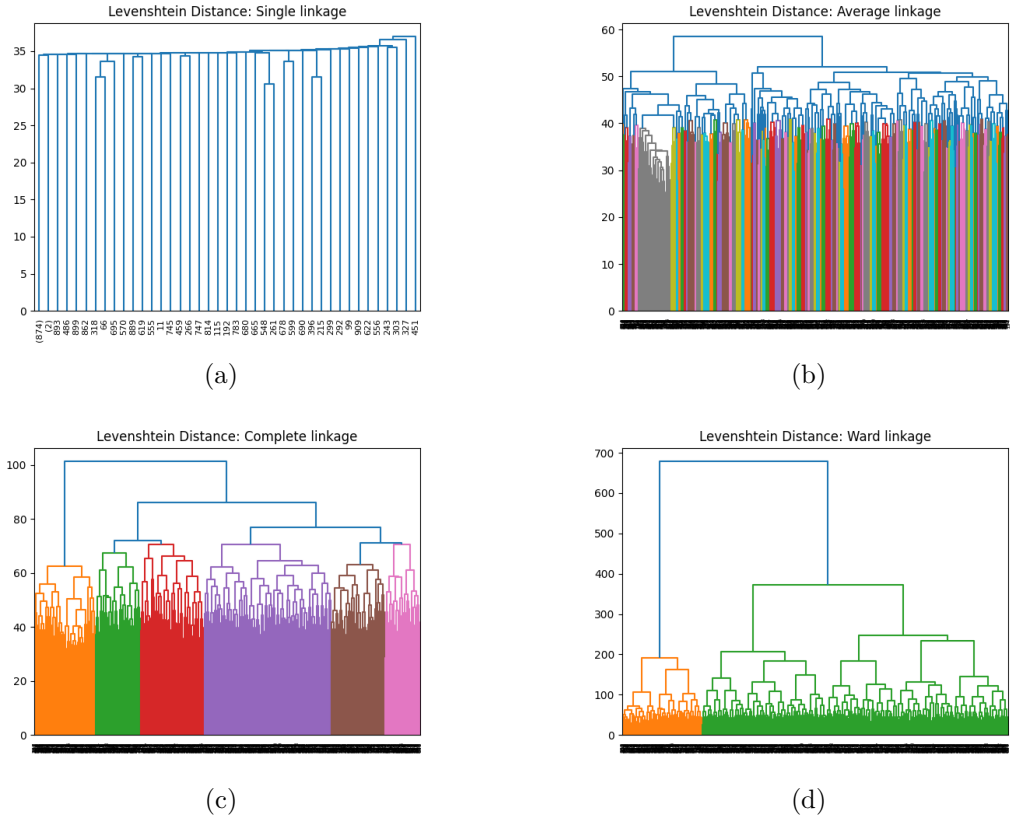
Supplementary Information



(a) (b)

(c) (d)

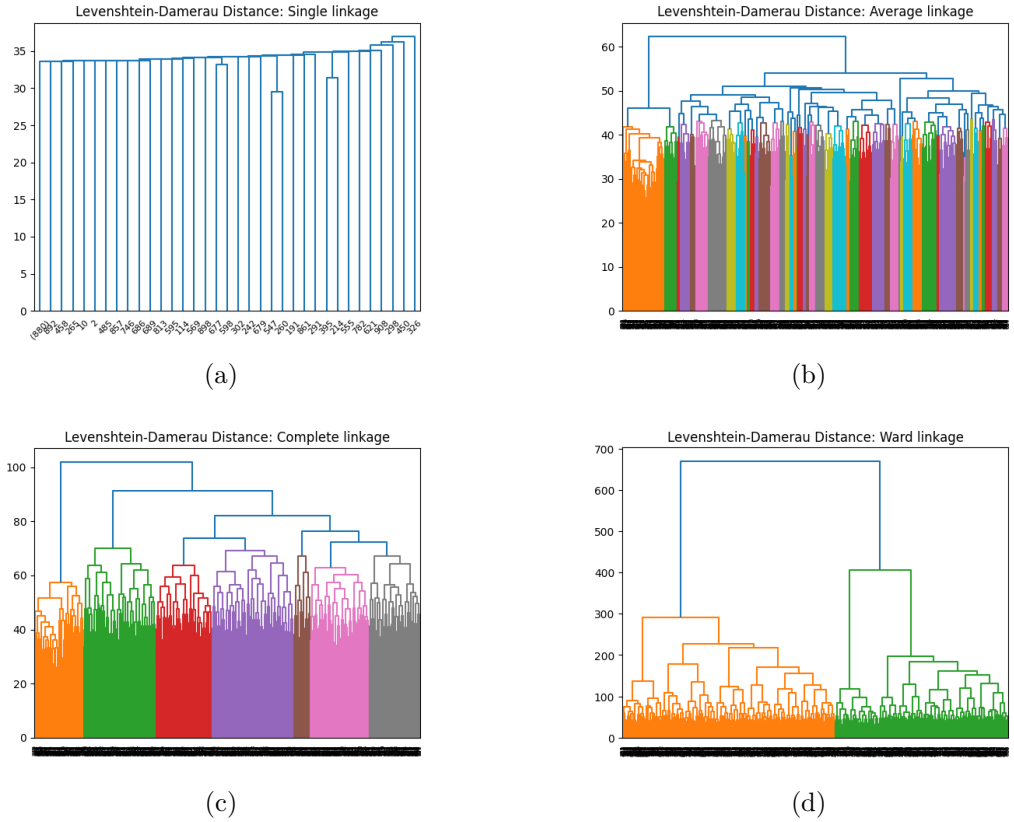
Figure 7.1: Different clustering linkages using the Hamming distance of 1000 Sequences *Single linkage* (a), *Average linkage* (b), *Complete linkage* (c), *Ward linkage* (d)



(a) (b)

(c) (d)

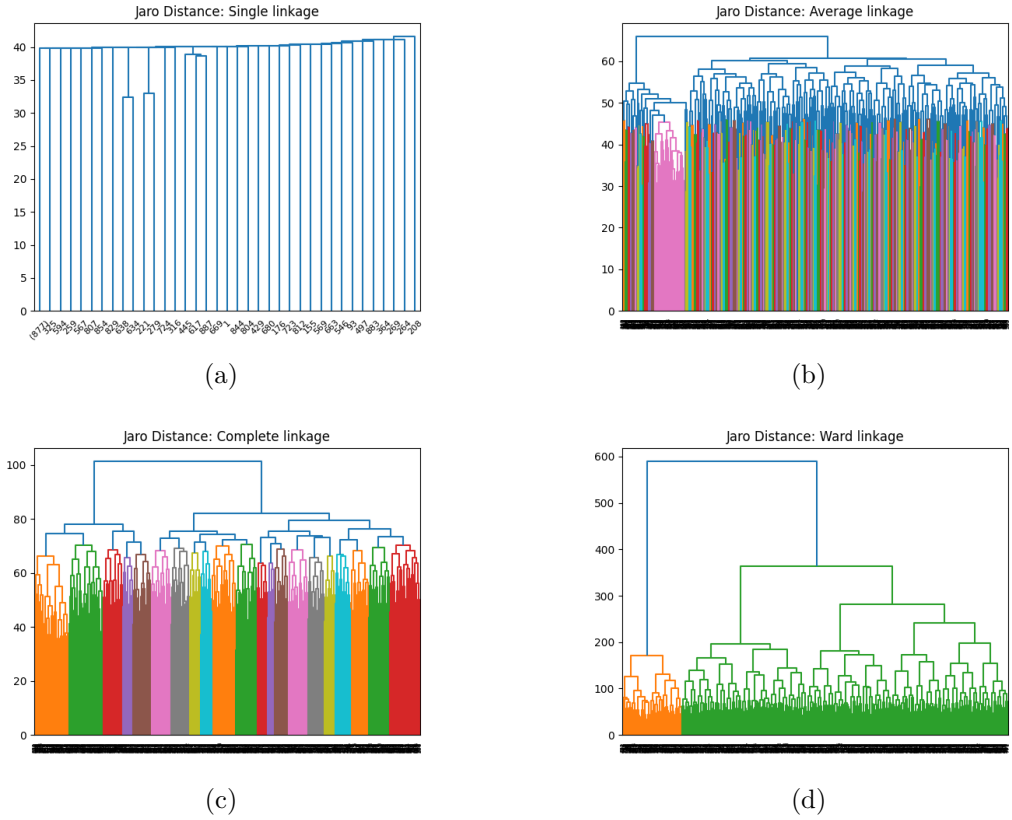
Figure 7.2: Different clustering linkages using the Levenshtein distance of 1000 Sequences *Single linkage* (a), *Average linkage* (b), *Complete linkage* (c), *Ward linkage* (d)



(a) (b)

(c) (d)

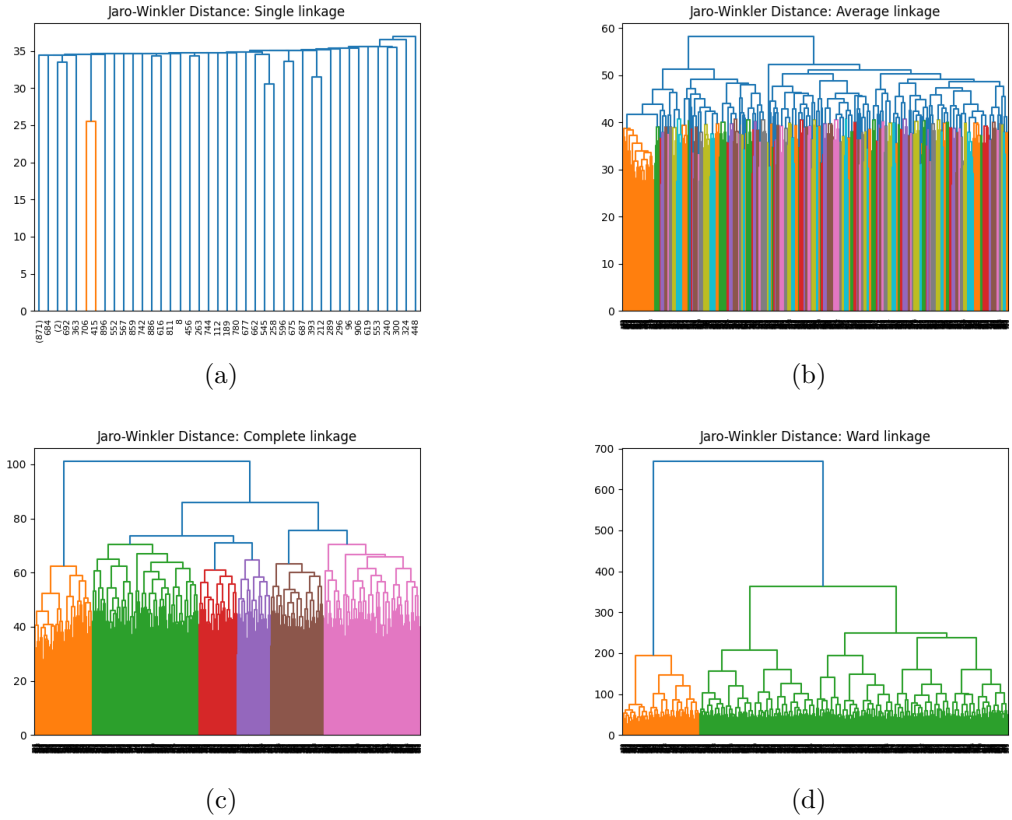
Figure 7.3: Different clustering linkages using the Damerau distance of 1000 Sequences *Single linkage* (a), *Average linkage* (b), *Complete linkage* (c), *Ward linkage* (d)



(a) (b)

(c) (d)

Figure 7.4: Different clustering linkages using the Jaro distance of 1000 Sequences *Single linkage* (a), *Average linkage* (b), *Complete linkage* (c), *Ward linkage* (d)



(a) (b) (c) (d)

Figure 7.5: Different clustering linkages using the Jaro-Winkler distance of 1000 Sequences *Single linkage* (a), *Average linkage* (b), *Complete linkage* (c), *Ward linkage* (d)

References

Literature

[1] Georgios A Pavlopoulos. “How to cluster protein sequences: tools, tips and commands” (Apr. 2017) (cit. on pp. 24, 25).

[2] Klaus Backhaus et al. *Multivariate Analysemethoden. Eine anwendungsorientierte Einführung*. 14th ed. Berlin and Heidelberg: Springer, 2015 (cit. on pp. 3, 8, 9, 11–13).

[3] Metin Balaban et al. “TreeCluster: Clustering biological sequences using phylogenetic trees” (Aug. 2019) (cit. on p. 19).

[4] Je Chang et al. *biopython. Biopython Tutorial and Cookbook*. Berlin and Heidelberg: http://www.biopython.org, 2021 (cit. on pp. 18–21).

[5] Limin Fu et al. “CD-HIT: accelerated for clustering the next-generation sequencing data” (Dec. 2012) (cit. on p. 24).

[6] Adam Godzik and Li Weizhong. “Cd-hit: a fast program for clustering and comparing large set of protein or nucleotide sequences” (July 2006) (cit. on p. 24).

[7] Fredrick Ishengoma. “Authentication System for Smart Homes Based on ARM7TDMI-S and IRIS-Fingerprint Recognition Technologies”. *CiiT International Journal of Programmable Device Circuits and Systems* 6 (Aug. 2014), pp. .162–167 (cit. on pp. 4, 6, 8).

[8] Angur Jarman. “Hierarchical Cluster Analysis: Comparison of Single linkage,Complete linkage, Average linkage and Centroid Linkage Method” (Feb. 2020) (cit. on pp. 11, 13).

[9] Evguenia Kopylova et al. “Open-Source Sequence Clustering Methods Improve the State Of the Art” (Oct. 2015) (cit. on pp. 24, 25).

[10] Daniel Müllner. “Modern hierarchical, agglomerative clustering algorithms” (Sept. 2011) (cit. on pp. 12, 13).

[11] Matthias Plaue. *Data Science. Grundlagen, Statistik und maschinelles Lernen*. Berlin: Springer, 2021 (cit. on pp. 3, 8, 9, 13).

[12] Daw Khin Po. “Similarity Based Information Retrieval Using Levenshtein Distance Algorithm”. *Int. J. Adv. Sci. Res. Eng* 6.04 (2020), pp. 06–10 (cit. on pp. 5, 6, 8).

[13] Werner Timischl. *Angewandte Statistik. Eine Einführung für Biologen und Mediziner*. 3rd ed. Wien, Heidlberg, Dordrecht, London, New York: Springer, 2013 (cit. on pp. 3, 9, 11, 13).

[14] Maria Tsagiopoulou et al. "UMIc: A Preprocessing Method for UMI Deduplication and Reads Correction". *Frontiers in Genetics* 12 (2021). URL: https://www.frontiersin.org/article/10.3389/fgene.2021.660366 (cit. on p. 26).

[15] Rui Xu and Donald Wunsch. "Survey of Clustering Algorithms". *Neural Networks, IEEE Transactions on* 16 () (cit. on pp. 2, 13).

[16] Shengnan Zhang, Yan Hu, and Guangrong Bian. "Research on string similarity algorithm based on Levenshtein Distance". In: *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE. 2017, pp. 2247–2251 (cit. on pp. 5, 6, 8).

[17] Chunchun Zhao and Sartaj Sahni. "Linear space string correction algorithm using the Damerau-Levenshtein distance". *BMC bioinformatics* 21.1 (2020), pp. 1–21 (cit. on pp. 7, 8).

[18] Chunchun Zhao and Sartaj Sahni. "String correction using the Damerau-Levenshtein distance". *BMC bioinformatics* 20.11 (2019), pp. 1–28 (cit. on pp. 7, 8).

Media

[19] Klaus Backhaus et al. *Multivariate Analysemethoden*. Wiesbaden, 2021 (cit. on pp. 14–16).

[20] Fredrick Ishengoma. *Authentication System for Smart Homes Based on ARM7TDMI-S and IRIS-Fingerprint Recognition Technologies*. Hamming distance (cit. on p. 4).

[21] Joos Korstanje. *3 text distances that every data scientist should know*. Hamming distance, Levenshtein Distance. URL: https://towardsdatascience.com/3-text-distances-that-every-data-scientist-should-know-7fcdf850e510 (cit. on pp. 5, 6).

Online sources

[22] *2.3. Clustering*. 2021. URL: https://scikit-learn.org/stable/modules/clustering.html (visited on 02/01/2022) (cit. on pp. 13, 22).

[23] Biopython. *Biopython*. June 3, 2021. URL: https://biopython.org (visited on 05/20/2021) (cit. on p. 1).

[24] *CD-HIT Official Website*. 2001-2022. URL: http://weizhong-lab.ucsd.edu/cd-hit/ (visited on 05/04/2022) (cit. on p. 24).

[25] *CD-HIT User's Guide*. 2006. URL: http://www.bioinformatics.org/cd-hit/cd-hit-user-guide (visited on 05/06/2022) (cit. on p. 24).

[26] Rosetta code. *Jaro similarity*. 2021. URL: https://rosettacode.org/wiki/Jaro_similarity (visited on 10/01/2021) (cit. on pp. 7, 8).

[27] Informatica. *Jaro Distance*. 2019. URL: https://docs.informatica.com/data-integration/data-services/10-1/developer-transformation-guide/comparison-transformation/field-matching-strategies/jaro-distance.html (visited on 10/01/2021) (cit. on pp. 7, 8).

[28] *Jaro and Jaro-Winkler similarity*. 2022. URL: https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/ (visited on 02/01/2022) (cit. on pp. 7, 8).

[29] *jellyfish*. URL: https://pypi.org/project/jellyfish/ (visited on 05/20/2021) (cit. on p. 1).

[30] Joos Korstanje. *3 text distances that every data scientist should know*. URL: https://towardsdatascience.com/3-text-distances-that-every-data-scientist-should-know-7fcdf850e510 (cit. on pp. 5, 6, 8).

[31] *scikit-learn. Machine Learning in Python*. URL: https://scikit-learn.org/stable/ (visited on 05/20/2021) (cit. on p. 1).

[32] *UPGMA and WPGMA trees*. 2021. URL: https://www.cs.rice.edu/~ogilvie/comp571/2018/10/25/upgma-and-wpgma.html (visited on 05/28/2022) (cit. on p. 12).