Karlsruhe Institute of Technology

Carnegie Mellon University

Data-Driven Learning of Clustering Algorithms for Image and Text Data

Master's Thesis of

Manuel Lang

at the Department of Informatics
Humanoids and Intelligence Systems Lab

Reviewer: Prof. Dr. Rüdiger Dillmann
Second reviewer:
Advisor:

1. January 2019 – June 21, 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Baden-Württemberg
STIPENDIUM®

This thesis was written during an exchange at Carnegie Mellon University in Pittsburgh (Pennsylvania) and was kindly supported by the Baden-Württemberg Stipendium.

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, June 21, 2019

. . . .

(Manuel Lang)

Acknowledgments

First, I would like to thank Prof. Dr. Rüdiger Dillmann for recommending me for the InterACT program and supporting me throughout my work.

In addition, I would like to thank Prof. Dr. Marina-Florina Balcan for supervising me during my stay at Carnegie Mellon University where she gave me the opportunity to work on very interesting research topics. Also, Nina supported me with insightful discussions, very helpful guidance and the option to bring in my own ideas and wishes. After my stay at CMU, Nina helped me to wrap up the project to have everything necessary to write this thesis and also submit this work as a contribution to NeurIPS 2019.

My thanks also go to the Automated Algorithm Reading Group and Nina's Learning Theory Group, where we had a lot of interesting discussions about state-of-the-art research that allowed me to learn a lot during my stay at CMU. Especially, I would like to thank Travis Dick, who supported me a lot during the implementation of the framework, the theoretical part of this work and also by finding interesting ideas to apply the introduced algorithms.

Last but not least, I would like to thank my family who supported me not only during my studies.

Abstract

Unsupervised learning is popular in several domains, such as grouping of images or websites, detecting outliers or providing recommendations. Often there are many algorithms that work similarly, but lead to different results. Depending on the data, one algorithm will work better than another and it is often not trivial to select the right algorithm, as worst-case guarantees for the algorithms have to be assumed, because there mostly is no prior information about the data, e.g. about its distribution. We solve the algorithm selection problem for agglomerative hierarchical clustering algorithms, where we propose a parameterized distance function between two clusters that weights the single, average and complete linkage distance individually and allows to us to interpolate linearly between the different linkage strategies.

With our framework, we outperform all of the discussed linkage strategies for a variety of data sets. In addition, we apply the framework to learn the best metric for a data set (i.e. the best feature representation of the data points), where we again achieve great results for a variety of learning tasks. This thesis summarizes the algorithm decisions we made to come up with the α-linkage algorithm as well as the results we obtained when applying the proposed algorithm to image and text data.

Zusammenfassung

Unüberwachtes Lernen ist in vielen Bereichen sehr populär, bspw. wird es verwendet um Bilder oder Webseiten in ähnliche Bereiche einzuteilen, um Ausreißer in Datensätzen zu erkennen oder um Nutzern Empfehlungen zu geben. Dabei gibt es viele verschiedenen Algorithmen, die zu verschiedenen Ergebnissen führen. Abhängig von den Daten eignen sich Algorithmen mehr oder weniger für verschiedene Aufgaben, weshalb es meist auch nicht trivial ist, welcher Algorithmus der beste für eine gegebene Aufgabe ist. Da oft keine genaueren Informationen über den verwendeten Datensatz zugrundeliegen, können nur schlechtmöglichste Annahmen für die Auswahl von Algorithmen berücksichtigt werden. In dieser Arbeit wird das Problem der Auswahl des passenden Algorithmus für agglomeratives hierarchisches Clustering durch eine parametrisierte Distanz-Funktion gelöst, die es ermöglicht linear zwischen verschiedenen Algorithmen zu implementieren. Dabei werden mit Single Linkage, Average Linkage und Complete Linkage drei mögliche Algorithmen zur Berechnung der Distanz zweier Cluster berücksichtigt.

Der in dieser Arbeit vorgestellte Algorithmus findet garantiert den besten Algorithmus, liefert zusätzlich aber noch bessere Clusterings als alle verwendenten Linkage-Strategien durch eine gewichtete Linearkombination dieser. Die Ergebnisse und das Potential dieses Algorithmus werden mit verschiedenen Bild- und Textdaten aufgezeigt. Zusätlich wird der vorgestellte Algorithmus auch dazu angewandt, die optimale Kombination verschiedener Metriken für Clusteringaufgaben zu finden, was ebenfalls zu deutlichen Verbesserung der Ergebnisse führt.

Contents

List of Figures

List of Tables

1. Introduction

Unsupervised grouping is used in various applications to categorize data observations into similar regions. As an example, similar documents can be combined into clusters so that for a new document or a search query, a list of corresponding documents can be shown [1]. The same procedure can also be applied for different tasks such as grouping products [2], searching images [3] or detecting anomalies [4]. In comparison to supervised learning, the data does not have to be (completely) annotated, i.e. potentially expensive labeling work can be avoided by using clustering algorithms.

As the amount of available data has been increasing in the past [5], data analysis is more often required for some specific use-case that includes a very specific dataset. State-of-the-art algorithms mostly provide general complexity- and runtime-guarantees, thus worst-case guarantees have to be assumed for the given dataset. However, as large datasets do often not adapt much over time, it is very likely that also runtime and complexity of certain algorithms applied on the given data will not change much. On the other hand, it is not trivial which algorithm can then be used to obtain the optimal results, i.e. the optimal clusters of the given data [6].

In addition, data is often split into different natural representations. For instance, images on websites can be seen as a matrix of pixels, but visually impaired people would rather use the image's alternative text description. For machine learning experiments it can be difficult to create a model based on various representation as it does not seem to be natural how to stack different data sources such as pixels and alternative texts [7].

This thesis proposes several algorithms to efficiently use a linear combination of clustering algorithms to overcome the hurdle of selecting the proper algorithm for the given data. In addition, the framework this algorithm is built in[1] will be also be applied on learning a weighted linear combination of feature representations. The proposed clustering algorithms belong to a specific family that will be introduced in chapter 2.

[1] The implementation is published open-source, see `https://github.com/manu183/LearningToLink`.

2. Related Work

2.1. Data-driven Algorithm Design

This increasing amount of data allows us to improve the learning capabilities of machines. We know how well existing algorithms perform in general and which runtime guarantees they have. However, the algorithms' guarantees are general observations and can vary a lot between different data. Also, it is often not trivial to choose the right algorithm for the given data without extensive data engineering. In many real-world applications the data does not vary that much, e.g. the data for clustering websites into different types may vary quite much on a yearly base, but as this task can get executed thousands of times each second for certain search algorithms, the data will not change much. By assuming a static context, it is then possible to leverage the context to improve the algorithmic results, e.g. say you want to cluster person data for different genders. By having this a-priori information, you can use a k-means clustering algorithm with $k = 3$ in order to differentiate between female, male and non-binary people.

However, such observations are mostly not that trivial and often require more effort in order to obtain useful a-priori information. In order to cluster financial standing, one could imagine seeing different clusters depending on the age or the education. But how many clusters would result here? The data has to be processed and evaluated for different values in this case.

Once our algorithm performs well for our data and our tasks, we then want to transfer the gained knowledge to different tasks. Say the algorithm already learned how to differentiate images of the handwritten digits zero, one and two, the same algorithm should then be able to apply the gained knowledge to distinguish between other handwritten digits too. The gained knowledge is some kind of learned data, that can for example be the feature representation of a Convolutional Neural Network, where a potential goal can be to transfer the representation knowledge to another classification task.

For clustering tasks learned knowledge could be a number of clusters, a good feature representation for the input data or other useful information that allows performing similar clustering tasks better by transferring the knowledge.

2.2. Linkage-based hierarchical clustering.

This thesis focuses on agglomerative hierarchical clustering, i.e. clustering algorithms that merge clusters starting from each cluster as its own point until all points belong to the same cluster. In each iteration, the two clusters with the closest distance get merged together. As there are various clustering algorithms, there also are various distance measurements. One way to describe the distance between two clusters, say X and Y, is by defining a linkage between them. There are three main methods to do so [8].

Single Linkage. Single linkage defines a distance between two clusters X and Y as the distance between the two nearest points of these clusters (see equation 2.1).

$$d_{SL}(X, Y) = \min_{x \in X, y \in Y} d(x, y) \tag{2.1}$$

Complete Linkage. Complete linkage defines a distance between two clusters X and Y as the distance between the two farthest points of these clusters (see equation 2.2).

$$d_{CL}(X, Y) = \max_{x \in X, y \in Y} d(x, y) \tag{2.2}$$

Average Linkage. Average linkage defines a distance between two clusters X and Y as the average distance between all points $x \in X$ and all points $y \in Y$ (see equation 2.3).

$$d_{AL}(X, Y) = \frac{1}{|X||Y|} \sum_{x \in X, y \in Y} d(x, y) \tag{2.3}$$

Figure 2.1 demonstrates which points are used for the different linkage strategies using two exemplary clusters.

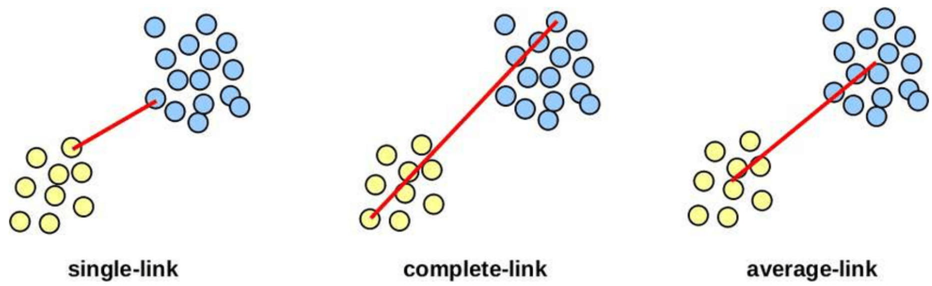


Figure 2.1.: To calculate the distance between two clusters, single linkage calculates the nearest neighbor distance, complete linkage uses the farthest neighbor and average linkage calculates the average distance over all points [9].

Effects of different linkage strategies. Depending on the linkage strategy, the pairwise distances between all N clusters $C_1, \ldots, C_N$ will be different. As the clustering algorithm merges the closest pair of clusters in each iteration, the merging clusters C_i and C_j with $i, j \in 1, \ldots, N$ might vary as shown in figure 2.2, where ten clusters $C_0, \ldots, C_9$ get clustered with bottom-up hierarchical clustering using the Euclidean distance as the pointwise distance $d(x, y)$ to calculate the pairwise distance according to the three mentioned linkage strategies.

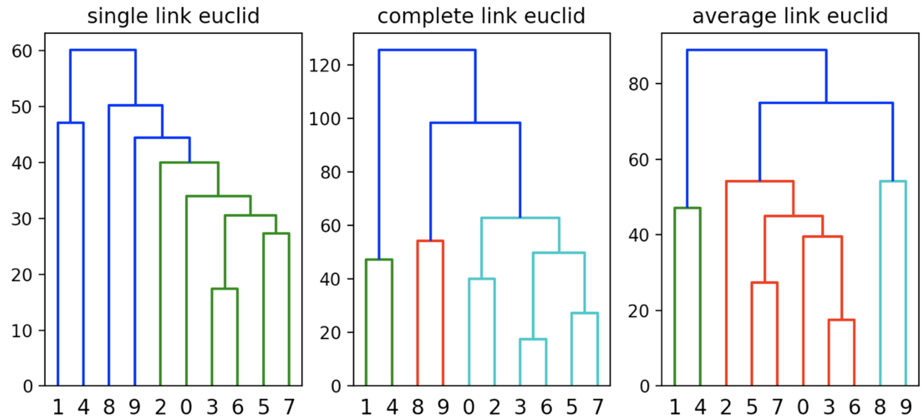


Figure 2.2.: Different distance measurements often result in different merges for bottom-up hierarchical clustering algorithms. The three discussed linkage strategies result in three different clusterings.

As different points are merged together, this also means that the clustering may have a different quality. This thesis compares the clusterings' quality for different data by introducing algorithms to efficiently determine the quality not only for these linkage strategies but also for their linear combinations.

2.3. Generating Feature Representations

In order to improve the overall clustering performance, this work uses several techniques to obtain better feature representations of text and image data.

2.3.1. Text Features

To cluster different words, there are several ways to create a difference measurement of these words. A rather simple approach would be to calculate the edit distance that describes the difference of the characters in the words [10]. However, this approach does not contain any semantic information, i.e. synonyms will have a larger distance than wanted. For example the edit distance between loan and moan is very small ($wed(loan, moan) = 1$) where the edit distance between loan and credit is larger ($wed(loan, credit) = 6$). This motivates to leverage contextual information, where we can use several pre-trained models on different datasets. Stanford's GloVe provides such models that incorporate knowledge from Wikipedia and social networks [11]. Figure 2.3 shows examples for why these embeddings give a helpful feature representation. In addition, CMU's Machine Learning Department provides another way to compare words. Their Never-Ending Language Learner provides information in which contexts certain words are used [12], e.g. by considering that both Pittsburgh and Karlsruhe are used in the one same context "is a city" and Pittsburgh and Philadelphia share additional contexts such as "belongs to the state Pennsylvania", we can conclude that Pittsburgh has a higher correlation to Philadelphia than to Karlsruhe. We can then create a corpus containing all different contexts. Similar to the bag-of-words approach, we can then count the occurrences of the words in the corpus' contexts. However, the resulting data is very sparse and Euclidean distance will not work well to compare the "bag-of-contexts" representations, i.e. other measurements such as the cosine distance are preferred.

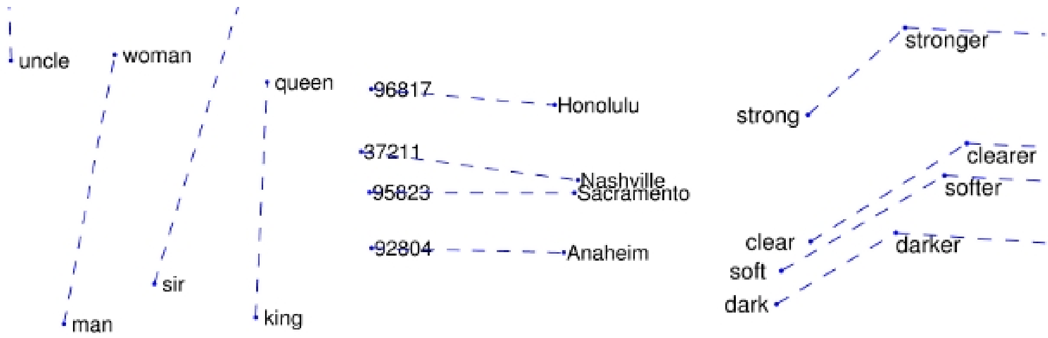


Figure 2.3.: Word embeddings give insightful correlations between similar and different words. For example, we can obtain several relations between female and male people, between zip codes and cities or between comparative and superlative words [11].

2.3.2. Image Features

Similarly, there also exist ways to extract useful features from image data. In particular, this work focuses on neutral networks that learn to represent images in a way that images of different classes can be separated well where images of the same class might share similar features. As a primary example, we use Convolutional Neural Network (CNN) architectures that learn to represent images with convolutional, pooling and activation layers and later map the representations to target classes with fully-connected layers. In this way, we can cut off the fully-connected layers to extract lower-dimensional feature representations for the input image data. Figure 2.4 visualizes such features learned on the ImageNet dataset [13].

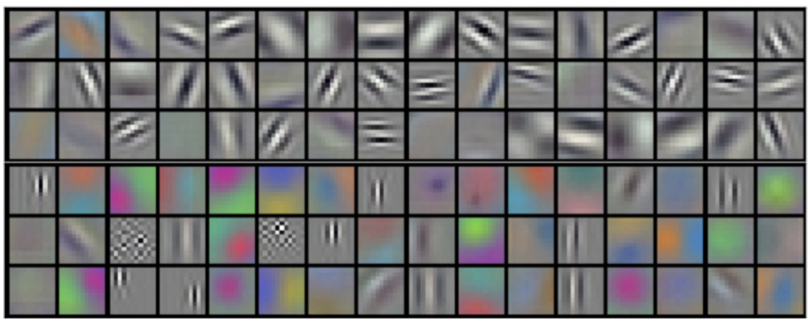


Figure 2.4.: Convolutional Neural Networks learn to represent images in lower-dimensional feature maps by applying convolutions to the image and averaging local neighborhoods (pooling) [13].

2.4. Data-Driven Clustering

As briefly discussed earlier, it is often not trivial to find the best algorithm for a given clustering task. While there already is empirical work in data-driven algorithm selection in certain domains such as choosing the step size in gradient descent [6], this thesis focuses on the earlier discussed bottom-up hierarchical clustering with the three different linkage strategies. In practice, there exists a variety of additional clustering algorithms that often also are parameterized, however data-driven methods only exist to some extent, e.g. for calculating the seed points of k-means efficiently [14].

As this work tries to select from a family of strategies, we first look at formulations that describe the given families. Balcan et al. proposed the two infinite families to interpolate between different linkage strategies [16], such as shown in equations 2.4 and 2.5.

$$\mathcal{A}_1 = \left\{ \left(\min_{u \in A, v \in B} (d(u,v))^\alpha + \max_{u \in A, v \in B} (d(u,v))^\alpha \right)^{1/\alpha} \middle| \alpha \in \mathbb{R} \cup \{\infty, -\infty\} \right\} \tag{2.4}$$

Equation 2.4 shows a distance in the range between single linkage ($\alpha = -\infty$) and complete linkage ($\alpha = \infty$). They also show that $\mathbb{R} \cup \{\infty, -\infty\}$ contains a maximum of $O(n^8)$ different intervals, where each interval $[\alpha_{lo}, \alpha_{hi}]$ represents a different merging behavior.

$$\mathcal{A}_2 = \left\{ \left(\frac{1}{||A|| ||B||} \sum_{u \in A, v \in B} (d(u,v))^\alpha \right)^{1/\alpha} \middle| \alpha \in \mathbb{R} \cup \{\infty, -\infty\} \right\} \tag{2.5}$$

Equation 2.5 will also result in single linkage for $\alpha = -\infty$ and complete linkage for $\alpha = \infty$. In addition to that, the family $\mathcal{A}_2$ also contains the definition of average linkage ($\alpha = 1$). However, the guarantee for maximum $O(n^8)$ intervals does not apply to this family. A formal guarantee will be $O(n^4 2^n)$, but this thesis will show that the empirical results are much better than the actual formal guarantee.

Balcan et. al also provide a solution to calculate all different merges of $\mathcal{A}_1$, however this approach solves the mathematical equations and leads to the same clusters being used for a merge quite often. As our solution only evaluates cases where different pairs of clusters get merged, the algorithm described in the following section has a lower runtime as well as a lower lower complexity.

3. Efficient Algorithm Selection

We define α as the parameter with which the distance of an algorithm is weighted. In this chapter we propose different distance measures depending on the weight parameter α that allows us interpolating between different linkage strategies in a way similar to the proposed by Balcan et al. [16], where a infinite interval was proposed.

To have a real application, we need to have a finite set of intervals. So we interpolate between one algorithm with $\alpha = 0$ and another algorithm with $\alpha = 1$, where for $\alpha = 0$ the result will be the result of algorithm 1 and for $\alpha = 1$ the result of algorithm 2.

3.1. Linear Interpolation between two different linkage strategies

Interpolating between two of the three mentioned linkage strategies results in three different algorithmic settings. In the first setting we are using the single linkage distance $d_{SL}(X, Y)$ and the complete linkage distance $d_{CL}(X, Y)$. By combining the two distances we can create a linear model that ranges from $\alpha = 0$ (single linkage) to $\alpha = 1$ (complete linkage) resulting in equation 3.1.

$$\begin{aligned} \mathcal{D}_{\mathrm{SC}}(X, Y, \alpha) &= (1-\alpha) \cdot d_{SL}(X, Y) + \alpha \cdot d_{CL}(X, Y) \\ &= (1-\alpha) \min_{x \in X, y \in Y} d(x, y) + \alpha \max_{x \in X, y \in Y} d(x, y) \end{aligned} \tag{3.1}$$

Equivalently we can interpolate between the single linkage distance $d_{SL}(X, Y)$ and the average linkage distance $d_{AL}(X, Y)$ instead of the complete linkage distance $d_{CL}(X, Y)$ for $\alpha = 1$ resulting in equation 3.2.

$$\begin{aligned} \mathcal{D}_{\mathrm{SA}}(X, Y, \alpha) &= (1-\alpha) \cdot d_{SL}(X, Y) + \alpha \cdot d_{AL}(X, Y) \\ &= (1-\alpha) \min_{x \in X, y \in Y} d(x, y) + \alpha \frac{1}{|X||Y|} \sum_{x \in X, y \in Y} d(x, y) \end{aligned} \tag{3.2}$$

The last of the three settings describes the interpolation between the average linkage distance $d_{AL}(X, Y)$ and the complete linkage distance $d_{CL}(X, Y)$ resulting in equation 3.3.

$$\begin{aligned} \mathcal{D}_{\mathrm{AC}}(X, Y, \alpha) &= (1-\alpha) \cdot d_{AL}(X, Y) + \alpha \cdot d_{CL}(X, Y) \\ &= (1-\alpha) \frac{1}{|X||Y|} \sum_{x \in X, y \in Y} d(x, y) + \alpha \max_{x \in X, y \in Y} d(x, y) \end{aligned} \tag{3.3}$$

3.2. Proposed Algorithms

Our goal is to find an algorithm that determines all different behaviors depending on the value of α. To do so, we propose different algorithms. First, we start with notations. In general, we evaluate results for a data domain $\mathcal{X}$. Here, we maximize the utility u of a clustering instance with the set of points $S = \{x_1, ..., x_n\} \in \mathcal{X}$ and an (unknown) target clustering $\mathcal{Y} = \{C_1, ..., C_k\}$. The output of the bottom up clustering is a binary cluster tree as shown in figure 2.2 and calculated with algorithm 1 where the top level node contains one cluster with all points and the leaf nodes contain one cluster for each point. We then prune the cluster tree to k clusters in order to compare these clusters to the target distribution. Afterwards, we use the in section 5.3 discussed cost functions as quality criteria for the resulting criteria.

Algorithm 1 α-linkage Clustering

Input: Merge functions D_0 and D_1, parameter $\alpha \in [0, 1]$, and clustering instance $S = \{x_1, . . . , x_n\}$.

1. Let $\mathcal{N} = \{\mathrm{Leaf}(x_1), . . . , \mathrm{Leaf}(x_n)\}$ be the initial set of nodes (one leaf per point).
2. While $|\mathcal{N}| > 1$
   a) Let $A, B \in \mathcal{N}$ be the clusters in $\mathcal{N}$ minimizing $\mathrm{D}_\alpha(A, B) = (1 - \alpha) \cdot \mathrm{D}_0(A, B) + \alpha \cdot \mathrm{D}_1(A, B)$.
   b) Remove nodes A and B from $\mathcal{N}$ and add $\mathrm{Node}(A, B)$ to $\mathcal{N}$.
3. Return the cluster tree (the only element of $\mathcal{N}$).

Next, we show an algorithm to divide the interval of $\alpha \in [\alpha_{lo}, \alpha_{hi}]$ into subintervals where the behavior is consistent within each interval, i.e. we split the interval $\alpha \in [\alpha_{lo}, \alpha_{hi}]$ into several different executions depending on the parameter α. First, we introduce the definition of an execution tree that stores all intervals $\mathcal{I} \in [0, 1]$ that lead to different clusterings. We start with the entire range $[\alpha_{\mathrm{lo}}, \alpha_{\mathrm{hi}}]$ and then iteratively bound the interval depending on the different merges. The result is a tree where each node represents a different merging behavior (see figure 3.1) and in the end, each leaf node corresponds to one cluster tree.

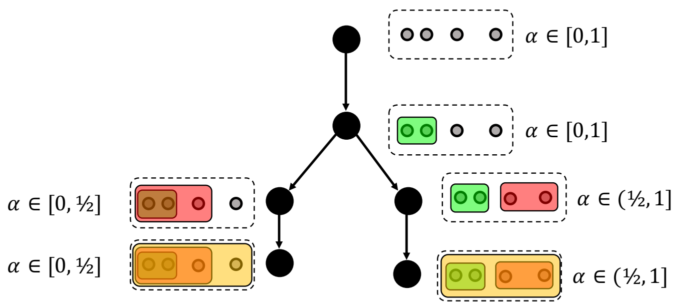


Figure 3.1.: The “tree of executions” stores all different merge behaviors and the resulting α-intervals in a tree.

Starting from an interval $\alpha \in [\alpha_{lo}, \alpha_{hi}]$, algorithm 2 calculates the merging clusters by $\min\limits_{X,Y} d(X, Y, \alpha)$ for both the minimum α_{lo} and the maximum α_{hi} of the interval. In case both

Algorithm 2 Building the Execution Tree

Input: Merge functions D_0 and D_1, clustering instance $S = \{x_1, \ldots, x_n\}$ and initial state st

1. Let $\mathcal{I} = \emptyset$ be the initially empty set of parameter intervals.
2. For iteration $1 : n - 1$
   - For each state $s \in st$
     a) remove state s
     b) Let A, B and C, D be the clusters that get merged for α_{lo} and α_{hi}.
     c) If $(A, B) == (C, D)$
        i. $ms \leftarrow$ merge (A, B)
        ii. add state ms with interval $[\alpha_{lo}, \alpha_{hi})$ to the end of st
     d) Else
        i. $\alpha_{split} \leftarrow$ calculate split $((A, B), (C, D))$
        ii. $s_1 \leftarrow$ merge (A, B)
        iii. $s_2 \leftarrow$ merge (C, D)
        iv. add state s_1 with interval $[\alpha_{lo}, \alpha_{split})$ to the end of st
        v. add state s_2 with interval $[\alpha_{split}, \alpha_{hi})$ to the end of st
3. For output state $s \in st$
   - add interval $i = [st.\alpha_{\text{lo}}, st.\alpha_{\text{hi}})$ to $\mathcal{I}$
4. Return $\mathcal{I}$

values of α return the same pair of merging clusters A and B, we merge A and B. In case the values of α_{lo} and ahi lead to different merges, we can calculate a value α_{split} where we know that for values of $\alpha \in [\alpha_{\text{lo}}, \alpha_{split})$ we merge the clusters found for $\min_{X,Y} d(X, Y, \alpha_{\text{lo}})$ and for values of $\alpha \in [\alpha_{split}, \alpha_{\text{hi}})$ we merge the clusters found for $\min_{X,Y} d(X, Y, \alpha_{\text{hi}})$. In order to calculate the value of α_{split} we can equalize the distance functions of the merges of clusters A, B and the clusters C, D (say α_{lo} leads to merging A and B and α_{hi} leads to merging C and D or vice versa) as seen in equation 3.4.

$$d_\alpha(A, B) = d_\alpha(C, D) \tag{3.4}$$

Applying 3.4 to a concrete example of distance functions leads to a concrete calculation for the value of α_{split}. Equation 3.5 shows the calculation for the in equation 3.1 introduced d_{SC}.

$$
\begin{gathered}
d_{SC}(A,B) = d_{SC}(C,D) \\
(1-\alpha_{split}) \min_{a\in A, b\in B} d(a,b) + \alpha_{split} \max_{a\in A, b\in B} d(a,b) = \\
= (1-\alpha_{split}) \min_{c\in C, d\in D} d(c,d) + \alpha_{split} \max_{c\in C, d\in D} d(c,d) \\
(-\alpha_{split}) \min_{a\in A, b\in B} d(a,b) + \alpha_{split} \max_{a\in A, b\in B} d(a,b) + \alpha_{split} \min_{c\in C, d\in D} d(c,d) - \alpha_{split} \max_{c\in C, d\in D} d(c,d) = \\
= - \min_{a\in A, b\in B} d(a,B) + \min_{C\in C, d\in D} d(c,d) \\
\alpha_{split}(- \min_{a\in A, b\in B} d(a,b) + \max_{a\in A, b\in B} d(a,b) + \min_{c\in C, d\in D} d(c,d) - \max_{c\in C, d\in D} d(c,d)) = \\
= - \min_{a\in A, b\in B} d(a,b) + \min_{c\in C, d\in D} d(c,d) \\
\alpha_{split} = \frac{- \min_{a\in A, b\in B} d(a,b) + \min_{c\in C, d\in D} d(c,d)}{- \min_{a\in A, b\in B} d(a,b) + \max_{a\in A, b\in B} d(a,b) + \min_{c\in C, d\in D} d(c,d) - \max_{c\in C, d\in D} d(c,d)}
\end{gathered}
\tag{3.5}
$$

After knowing the exact consistent range, we split the clusters for the different states and then calculate the merge candidates for the start and the end of the new intervals again. We can show the different possible merges in a tree of executions, where each node represents one interval $[\alpha_{lo}, \alpha_{hi}]$ where the same clusters get merged (see figure 3.1). We perform the described procedure for iterations $i = count(points) - 1$ times until only one cluster containing all points is left. All the leaf nodes in the resulting tree of executions then represent one interval $[\alpha_{lo}, \alpha_{hi}]$ where the clustering is expected to be consistent within the interval and each interval contains a different clustering. Next we will show that all intervals are well-defined, i.e. in each interval, the clustering always is constant. To do so, we prove that each distance function is a linear function depending on the parameter α (see equation 3.6).

$$
\begin{aligned}
d_\alpha(A,B) &= \alpha \cdot d_0(A,B) + (1-\alpha) \cdot d_1(A,B) \\
&= d_1(A,B) + \alpha \cdot (d_0(A,B) + \cdot d_1(A,B))
\end{aligned}
\tag{3.6}
$$

As we now know that all distance functions are linear functions, we can argue that by calculating the intersection of two distance functions, we can say that both distance functions will be optimal for some region. However just calculating the split values in a range $[\alpha_{lo}, \alpha_{hi}]$ does not necessarily yield to the best possible solution. One of these examples is demonstrated in figure 3.2, where the blue line for the constant value of d will not be considered, only the lines $\alpha \in [\alpha_{lo}, \alpha_{split})$ (red) and $\alpha \in [\alpha_{split}, \alpha_{hi})$ (black) will be.

In order to solve this, we can recursively check each resulting interval again if it contains different merging behaviors.

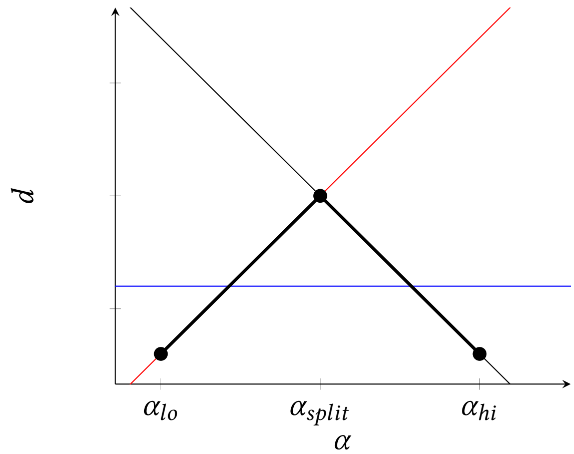


Figure 3.2.: Simply calculating the split values between the start and the end value of the range $[\alpha_{lo}, \alpha_{hi}]$ will not necessarily lead to the optimal values. By doing so, the blue line (constant d value) will not be considered.

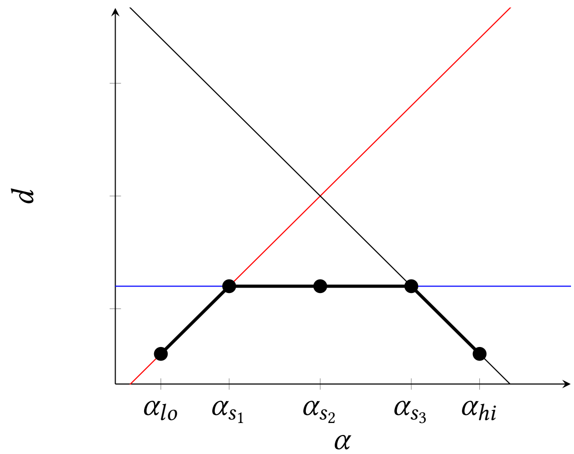


Figure 3.3.: Simply calculating the split values between the start and the end value of the range $[\alpha_{lo}, \alpha_{hi}]$ will not necessarily lead to the optimal values. By doing so, the blue line (constant d value) will not be considered.

By calculating the split points recursively, the example in figure 3.3 will result in the intervals $[\alpha_{lo}, \alpha_{s_1}]$, $[\alpha_{s_1}, \alpha_{s_2}]$, $[\alpha_{s_2}, \alpha_{s_3}]$ and $[\alpha_{s_3}, \alpha_{s_{hi}}]$. The optimal distance between α_{s_1} and α_{s_3} is covered now, but the results contain one unncessary interval as α_{s_2} still splits two intervals. The algorithm can check if older splits are still relevant, however the runtime cost to do so will be more expensive than carrying one additional interval with the same distance. We can use this knowledge and adapt algorithm 2.

As experimental results turn out to need a lot of memory (up to $\approx$ 20 GB for 300 points and 20,000 states), we want to adapt algorithm 3 so that it uses less memory. The memory usage scales relative to the amount of currently in-memory stored states, so the goal is to reduce these. As the amount of states is much larger than the amound of iterations, we

Algorithm 3 Recursive Interval Calculation

Input: Merge functions D_0 and D_1, clustering instance $S = \{x_1, \ldots, x_n\}$ and initial state st

1. Let $\mathcal{I} = \emptyset$ be the initially empty set of parameter intervals.
2. For iteration $1 : n - 1$
   - For each state $s \in st$
     a) remove state s
     b) ranges $\leftarrow$ find ranges between $s.\alpha_{\mathrm{lo}}$ and $s.\alpha_{\mathrm{hi}}$
     c) For each range $r \in \mathit{ranges}$
        i. $A, B \leftarrow$ candidate for range
        ii. $ms \leftarrow$ merge A, B
        iii. add state ms with range r to the end of st
3. For output state $s \in st$
   - add interval $i = [st.\alpha_{\mathrm{lo}}, st.\alpha_{\mathrm{hi}})$ to $\mathcal{I}$
4. Return $\mathcal{I}$

calculate and evaluate the leave nodes of the tree and keep the alternative merges stored, i.e. we use a depth-first instead of a breadth-first approach. This results in algorithm 4.

Algorithm 4 Depth-first α-linkage

Input: Merge functions D_0 and D_1, clustering instance $S = \{x_1, \ldots, x_n\}$ and initial state st

1. Let $\mathcal{I} = \emptyset$ be the initially empty set of parameter intervals.
2. While $|st| > 0$
   - For each state $s \in st$
     a) remove state s
     b) If s is final: add interval $i = [s.\alpha_{\mathrm{lo}}, s.\alpha_{\mathrm{hi}})$ to $\mathcal{I}$
     c) Else:
        i. ranges $\leftarrow$ find ranges between $s.\alpha_{lo}$ and $s.\alpha_{hi}$
        ii. For each range $r \in \mathit{ranges}$
           A. $A, B \leftarrow$ candidate for range
           B. $ms \leftarrow$ merge A, B
           C. add state ms with range r to the beginning of st
3. Return $\mathcal{I}$

Using a depth-first implementation instead of a breadth-first implementation leads to huge benefits. We do not need a lot of memory anymore and together with the memory / copying costs, we also improve the runtime. Figure 3.4 gives insights about the memory usage and the required runtime for our MNIST experiments with algorithm 3 (breadth-first) and algorithm 4 (depth-first). We notice that the memory usage for the breadth-first implementation had an exponential growth. Clustering 500 points already needed 8 GB of memory, that was the maximum of the device the experiments were run on, i.e. for larger-scale experiments we would have been forced to use better hardware. Instead, with the depth-first we have a linear growth over the points and it is possible to reduce the

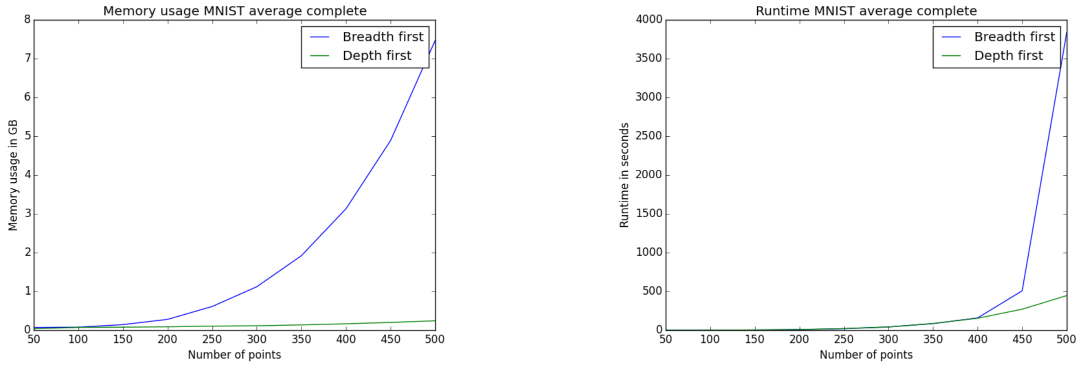


Figure 3.4.: The depth first implementation needs less memory and also has a better runtime compared to the breadth first implementation.

amount of needed memory further if the intervals do not have to be stored in-memory, e.g. when we run an experiment on one dataset, we can directly export resulting intervals. For the breadth-first approach this would also be possible, but not prior to the last iteration, as we have to store each interval in each of the previous iterations. In addition, figure 3.4 shows that the runtime also benefits from using the depth-first implementation. While we needed more than one hour to evaluate 500 points, we now need less than 10 minutes. This allows us to scale up our experiments from so far ≈ 250 points to $\approx 1,000$ points.

As an addition, instead of merging iteratively and steadily shrinking the intervals, we propose a tweaked version of algorithm 4 with a geometric motivation. We are again evaluating an interval $[\alpha_{lo}, \alpha_{hi}]$, but we interpret the different merges as linear functions depending on α. We can start by calculating the merge candidate for the start value α_{lo} and calculate the next intersection that will yield to the next merge. By calculating all the intersections of linear functions, we can also determine all the different intervals for the range $[\alpha_{lo}, \alpha_{hi}]$, where different merging behaviors occure. Algorithm 5 describes this procedure.

This leads to a clean implementation, where we do not store unnecessary intervals (see figure 3.3) any longer. In contrast, we now only store the intervals where different merges occur. By interpreting the geometrics of linear functions, we always find the optimal merges for any value of α leading to well-defined intervals for any clustering instance interpolating within $\mathcal{D}_{\mathrm{SC}}$, $\mathcal{D}_{\mathrm{SA}}$ or $\mathcal{D}_{\mathrm{AC}}$. Figure 3.5 shows the optimized clustering of the exemplary distance functions.

3.3. Performance Optimizations

In order to have real-world applications, the proposed algorithms should run in an efficient way, i.e. it should not take the α-linkage algorithms too much time to run. A first python implementation took days to run, but switching to C++ and using its advantages took down the runtime to hours. However, there are more optimization methods that we used in order to improve the runtime.

Algorithm 5 α-linkage with Geometric Interval Calculation

Input: Merge functions D_0 and D_1, clustering instance $S = \{x_1, \ldots, x_n\}$ and initial state st

1. Let $\mathcal{I} = \emptyset$ be the initially empty set of parameter intervals.
2. While $|st| > 0$
   - For each state $s \in st$
     a) remove state s
     b) If s is final: add interval $i = [s.\alpha_{\text{lo}}, s.\alpha_{\text{hi}})$ to $\mathcal{I}$
     c) Else:
        i. $\alpha \leftarrow \alpha_{lo}$
        ii. linear function $lf \leftarrow$ get lf for α
        iii. While $\alpha < \alpha_{hi}$
           A. $\alpha_{new} \leftarrow$ calculate next split for α
           B. $lf \leftarrow$ get lf for α_{new}
           C. $\alpha \leftarrow \alpha_{new}$
3. Return $\mathcal{I}$

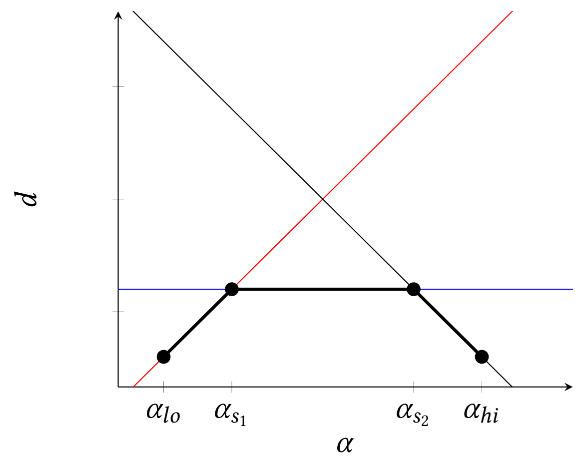


Figure 3.5.: Simply calculating the split values between the start and the end value of the range $[\alpha_{lo}, \alpha_{hi}]$ will not necessarily lead to the optimal values. By doing so, the blue line (constant d value) will not be considered.

3.3.1. Dynamic Programming

One of the most time-consuming parts was the calculation of the distances. For each pair of clusters C_i, C_j the distance had to be calculated for each clustering state. We optimized this by using dynamic programming to store the distance matrices D_{lower} and D_{upper} for each state. There we interpolate from one linkage distance (lower) to another linkage distance (upper), e.g. the $\mathcal{D}_{\mathrm{SC}}$ setting describes the interpolation from single linkage (lower) to complete linkage (upper). In this example we then store the pairwise distances for both single linkage and complete linkage and in order to find the merge candidates we have to iterate over the distance matrices instead of calculating the distances over and over again. When we merge two clusters, we then update the distance matrices for the given state. Table 3.1 shows an example for the pairwise distances of clusters i and j.

j\i	0	1	2	3	4
0	0	1.243	1.512	2.468	5.1243
1	1.243	0	2.443	3.1412	4.443
2	1.512	2.443	0	3.8988	6.827
3	2.468	3.1412	3.8988	0	5.72
4	5.1243	4.443	6.827	5.72	0

Table 3.1.: Storing the pairwise distances of all clusters avoids calculating the distances over and over again.

One observation that we can make is that the matrix has a lot of redundant values, because all our distance functions are symmetric, i.e. $D(i, j) = D(j, i)$. Removing these redundant values will result in a trade-off between copying and indexing costs and will be discussed in the following section. Another optimization we can do is to store the indices of the active clusters, i.e. the clusters that can get merged. Once two clusters got merged, they cannot be merged any further, only the resulting cluster can. So we then do not have to consider the old clusters anymore and can remove them from the set of active indicies. This allows us to find the merge candidates faster as the pool of candidates gets smaller.

3.3.2. Trade-Off between Copying and Indexing Costs

Currently we can access the costs for a pair of clusters C_i and C_j through $D[i, j]$ or $D[i + j * width]$ for flattened matrices. These indices are very easy to determine. In order to remove the redundant values from the distance matrix we remove all values below the diagonal as shown in table 3.2.

In addition to that we can also remove the diagonal values as they represent the distances between the same clusters and are thus always zero. This results in table 3.3.

j\i	0	1	2	3	4
0	0	1.243	1.512	2.468	5.1243
1		0	2.443	3.1412	4.443
2			0	3.8988	6.827
3				0	5.72
4					0

Table 3.2.: Removing the redundant distance values leads to less memory usage, but to more efficient index calculations.

j\i	0	1	2	3	4
0		1.243	1.512	2.468	5.1243
1			2.443	3.1412	4.443
2				3.8988	6.827
3					5.72
4					

Table 3.3.: We also get rid of the distances between the same clusters in the stored distance matrices.

The matrices are now smaller, so they need less memory. In the example, we changed a matrix of the size 25 to a matrix of the size 10. In general a matrix of the size nxn will be compressed to a matrix of the size $\frac{n^2-n}{2}$. The lower amount of needed memory also results in less copying costs that will lead to a better runtime. However, the indexing is not as easy anymore. For easier storage, we again work with flattened matrices, the indexing for the resulting list is shown in equation 3.7.

$$index(i,j) = \frac{width * (width - 1)}{2} - \frac{(width - j) * (width - j - 1)}{2} + i - j - 1 \tag{3.7}$$

Calculating this index in a nested loop is very expensive, however we calculate the part $index(j)$ that does not depend on i in the outer loop and thus only need to add i in the inner loop. This does not only yield to a lower memory usage of $\approx$ 30%, but also increases the runtime by roughly the same factor.

3.3.3. Implementation-specific Optimizations

In order to optimize the implementation even further, we will have a look into the implementation. One optimization that already was briefly described is the flatterning of the matrices, so the resulting list will be one-dimensional and can be iterated easier and faster.

Another observation is that copy operations are computationally expensive, so we avoid them as much as possible. In the described algorithms (2, 3 and 4) we removed a state from the list of states and added other states. In an optimized way, we do not remove the state and just overwrite the state with the resulting state. Once there are splits in the current interval, the state gets overwritten and additional states get added to the list.

We can also optimize the way of updating the distance matrices. Instead of adding new clusters there for a merge of clusters i and j we update the distances of i to all active clusters with the distances of the resulting cluster. The distances of the cluster j will not be considered for merges anymore as the index j gets removed from the active indices. This has the advantage that the size of the distance matrices will not increase after merges.

Also, the data types make an important contribution to the memory usage. Instead of using double precision floating point values, single precision is enough to clearly identify and separate all the resulting intervals. Same goes for the distances as we only need the minimum and maximum distances, that are not effected by loss of precision. To store the indices of the clusters, we know that they will not exceed 2^{16}, so they can be store as half precision values.

4. Optimizing the Metric

In a similar fashion as described in section 3 this sections aims to optimize a metric that is a linear combination of several metrics. For instance, images can have a 2D pixel representation and a text describing the each image. Combining these features for clustering tasks can be problematic as it is not how the optimal weight between these features should be. Does a word describe more than a subset of the image, are the features equally important or does the pixel image lead to better clusterings? With β-linkage we provide a framework based on α-linkage that calculates different merges based on linear combinations of representations and leads to optimized clusterings.

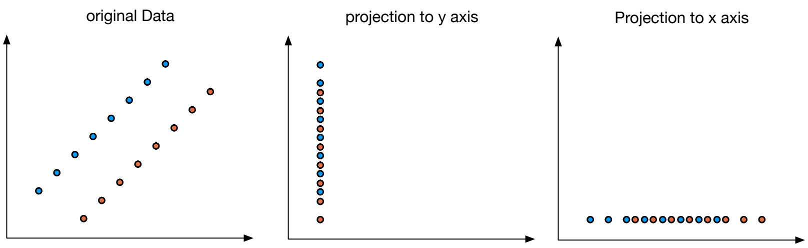


Figure 4.1.: Combining several metrics seems often natural and can lead to improved results as in this example where we project a dataset on both axes.

For instance, figure 4.1 shows a set of points that might be put in clusters easily. However, if you only look at the distance regarding the X_1-axis or the X_2-axis clustering will be very difficult, because each of the axis does not describe the spatial correlation anymore. This example is selected on purpose to motivate the following experiments where we learn optimal combinations of different metrics.

To interpolate between d_0 and d_1, we use the same interpolation as discussed in section 3. We use a parameter $\beta \in [0, 1]$ and weight the metrics as shown in equation 4.1.

$$d_\beta(x, x') = (1 - \beta) \cdot d_0(x, x') + \beta \cdot d_1(x, x') \tag{4.1}$$

$$d_\beta(x, x') = d_0(x, x') + \beta \cdot (d_1(x, x') - d_0(x, x')) \tag{4.2}$$

We can then compete all possible discontinuities by comparing the distances of given clusters (x, x') and (y, y'). As $d_\beta(x, x')$ is a linear function depending on β (see equation 4.2), we can compute all discontinuities by solving the following equation.

$$
\begin{aligned}
&d_\beta(x,x') = d_\beta(y,y') \\
&(1-\beta)\cdot d_0(x,x') + \beta\cdot d_1(x,x') = (1-\beta)\cdot d_0(y,y') + \beta\cdot d_1(y,y') \\
&d_0(x,x') - \beta\cdot d_0(x,x') + \beta\cdot d_1(x,x') = d_0(y,y') - \beta\cdot d_0(y,y') + \beta\cdot d_1(y,y') \\
&\beta\cdot(-d_0(x,x') + d_1(x,x') + d_0(y,y') - d_1(y,y')) = -d_0(x,x') + d_0(y,y') \\
&\beta = \frac{-d_0(x,x') + d_0(y,y')}{-d_0(x,x') + d_1(x,x') + d_0(y,y') - d_1(y,y')}
\end{aligned}
$$

As we know that the function d_β is a linear function depending on β and we showed that all discontinuities depend on four points, we know that there at most $O(n^4)$ well-defined intervals $I_i \in [0,1]$ for any clustering instance S, i.e. in any interval I_i the algorithm will merge the same two points. In order to efficiently calculate and evaluate all resulting cluster trees, we use an algorithm similar to algorithm 5, but adapted to β-linkage.

Algorithm 6 β-linkage Clustering

Input: Metrics d_0 and d_1, parameter $\beta \in [0,1]$, and clustering instance $S = \{x_1, \ldots, x_n\}$.

1. Let $\mathcal{N} = \{\mathrm{Leaf}(x_1), \ldots, \mathrm{Leaf}(x_n)\}$ be the initial set of nodes (one leaf per point).
2. While $|\mathcal{N}| > 1$
   a) Let $A, B \in \mathcal{N}$ be the clusters in $\mathcal{N}$ minimizing $\max_{a\in A, b\in B} \mathrm{d}_\beta(a,b)$.
   b) Remove nodes A and B from $\mathcal{N}$ and add $\mathrm{Node}(A,B)$ to $\mathcal{N}$.
3. Return the cluster tree (the only element of $\mathcal{N}$).

A slight adaption that we need is to normalize the features such that for $\beta = 0.5$ we equally weight both distance matrices. We achieve this by dividing the features $f_1, \ldots, f_k$ through the maximum value f_{max}. In this way, we scaled the features into $[0,1]$ and did not lose the proportions as it would happen when using a min-max-scaler.

5. Experimental Setup

This work evaluates the proposed algorithms for image and text data. This chapter describes the used datasets and the evaluation methods.

5.1. Datasets

5.1.1. Synthetic Data

To motivate our approach, we manually created a dataset containing disks and rings as shown in figure 5.1. In this case, we know that single linkage performs well clustering the two rings, however it might be problematic to cluster the disks. On the other hand, complete linkage is expected to cluster the disks very well, but it might contact the two rings earlier than wanted. This data motivates our approach of interpolating between different linkage strategies, however the data is not natural and real-world datasets are very likely to have a different structure.

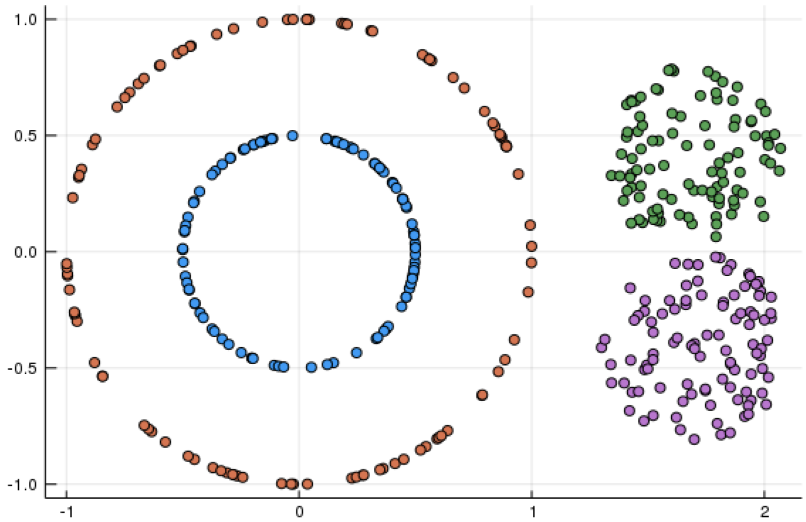


Figure 5.1.: We use disks and rings as a sample dataset to motivate our α-linkage approach. The dataset contains four clusters, two disks and two rings.

5.1.2. Text Data

Never Ending Language Learner data. The Never Ending Language Learner (NELL) is a learning agent that reads the web, extracts data and verifies beliefs [17][18]. NELL for example knows that "Pittsburgh" is located in "Pennsylvania". These beliefs represent different noun-phrases such as "Pittsburgh" and "Pennsylvania". The noun-phrases belong to certain categories. "Pittsburgh" is a "City" and "Pennsylvania" is a "State". These subcategories both belong to the main category "Geopolitical Location". Figure 5.2 shows such a knowledge graph. While there are already different subcategories, the goal for a hierarchical clustering algorithm here is to extract new useful subcategories.

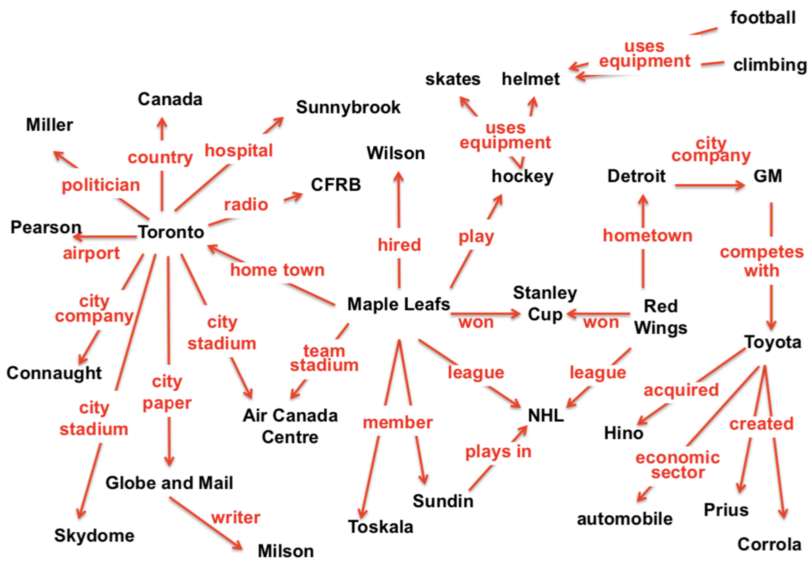


Figure 5.2.: The Never Ending Language Learner represents different entities and their correlation.

The used dataset, extracted web-information by NELL, contains 32 different main categories, such as "Animal", "Location" or "Person". Each of these consists of up to 250 different entities that belong to different subcategories. Exemplary entities for the category "Animal" are "Otter", "Squirrel" or "Wolf".

5.1.3. Image Data

MNIST handwritten digits. The MNIST handwritten digit database contains images of the handwritten digits from zero to nine [19]. Samples of these images are shown in figure 5.3 Its training set contains a total of 60,000 images, where each image is represented as a 784-dimensional vector corresponding to a greyscale image with 28x28 pixels.

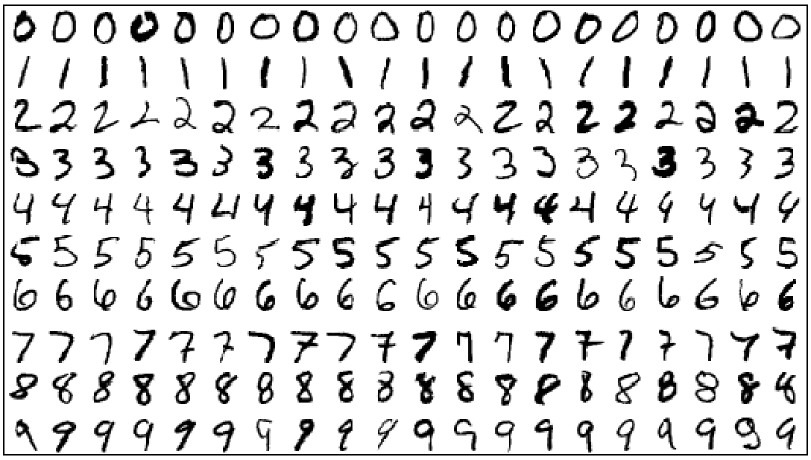


Figure 5.3.: The MNIST handwritten digits database contains 60,000 greyscale images of handwritten digits ranging from zero to nine. These samples show ten randomly drawn samples for each label represented as a 28x28 pixel image [19].

The goal of clustering MNIST images is to find an unsupervised learning method that can distinguish between greyscale images. In addition, we can define various clustering tasks where we pick a subsample of the ten labels and then try to transfer the results to other subsamples. For example, we first cluster images labeled as zero, one, two, three or four and later apply the knowledge the learned gained for clustering images labeled as five, six, seven, eight or nine. Theses types of experiments allow high-level transfer learning if we define several different clustering tasks, e.g. for five different labels there are $\binom{10}{5} = 252$ different combinations of labels.

Another obsevation that results from hierarchical clustering is the similarity of different labels, i.e. which labels are likely to get clustered together.

CIFAR-10. Another image dataset this thesis uses for evaluation is the CIFAR-10 dataset that contains 60,000 RGB images of ten different categories [20]. Each image consists of 32x32 pixels and is thus represented as a 3072-dimensional vector (32x32x3). The categories and ten random images from each are shown in figure 5.4.

As the amount of images and the amount of classes is equal to the ones in the MNIST database, we can also try similar experiments. The main difference is that the images consist of RGB pixels instead of greyscale values.

CIFAR-100. The CIFAR-100 dataset contains similar images, but instead of 6,000 images each for 10 classes, it consists of 600 images each for 100 classes. The classes are di-

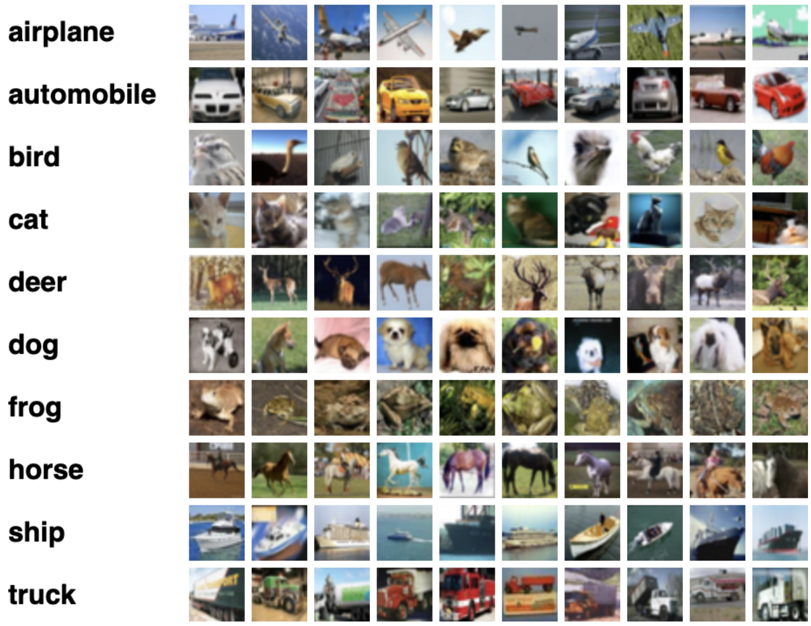


Figure 5.4.: The CIFAR-10 database contains 60,000 RGB images of the ten shown different classes. These samples show ten randomly drawn samples for each label represented as a 32x32 pixel image [20].

vided into 20 superclasses each containing five subclasses. Examples of superclasses and corresponding subclasses are shown in table 5.1.

superclass	subclasses
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle

Table 5.1.: The CIFAR-100 dataset contains 20 different superclasses, each with five different subclasses leading to 100 classes overall. The images are represented in the same way as in the CIFAR-10 dataset, i.e. by a 3072-dimensional vector [20].

Having superclasses and subclasses allows clustering between different subclasses within a superclass and also between different superclasses. This allows more experiments than for the CIFAR10 data.

Omniglot. The omniglot dataset contains 1623 handwritten characters from 50 different alphabets, where each character is represented by 20 different images. Each image is grayscale and represented by 105x105 pixels [21]. Figure 5.5 shows characters of the more well-known Latin, Greek and Hebrew alphabets that are part of the dataset.

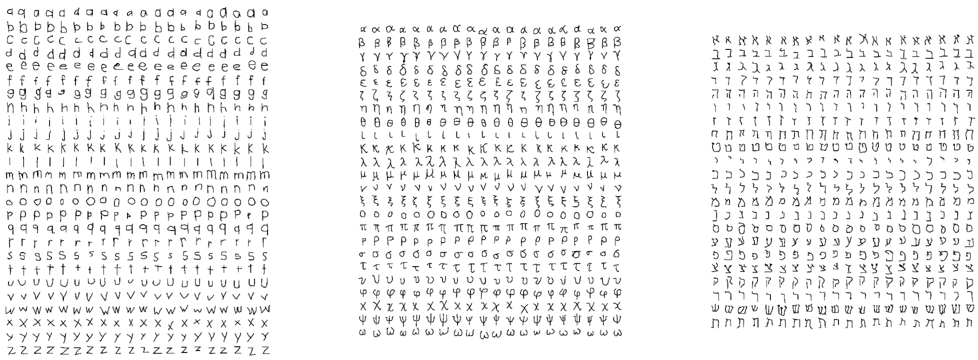


Figure 5.5.: The omniglot dataset contains handwritten characters of different alphabets, such as Latin (left), Greek (middle) and Hebrew (right) [21].

The omniglot dataset is similar to the MNIST dataset as it also contains handwritten characters, however it has more different characters and less images of each of them. This allows us to run more learning tasks.

5.2. Pruning

In order to evaluate a cluster tree with some target labels, we have to prune the tree into the corresponding number of k target labels, i.e. the k different classes of the target data. Therefore, we evaluate the costs of all possible combinations of k clusters and select the best clusters. Figure 5.6 shows an example how we can prune a clustering tree into $k = 3$ clusters by either using the clusters C, D and E or the clusters B, F and G.

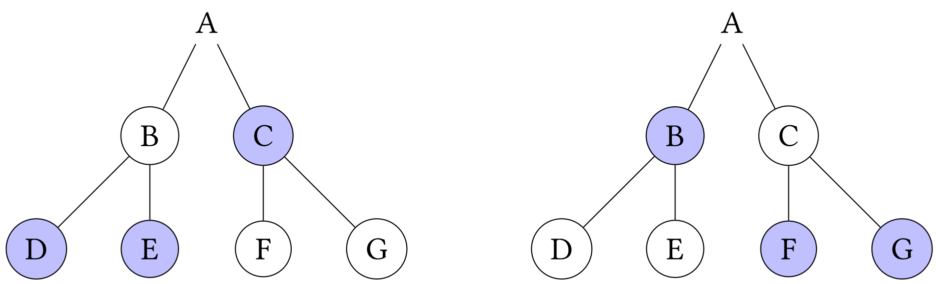


Figure 5.6.: There are multiple ways we can prune a cluster tree into k clusters.

However, we still have to find a way to calculate the preferred pruning of a clustering tree into k clusters, i.e. whether in case of figure 5.6 we would prefer the pruning on the left or the one on the right.

5.3. Cost functions

In order to evaluate the quality of a clustering, we need some kind of cost function that compares the generated clustering $C_1, \ldots, C_k$ with the target clustering $C_1^*, \ldots, C_k^*$.

Majority Cost. One method to compare them is the majority distance as shown in equation 5.1 where n ist the number of sampled points.

$$cost_{majority}(C_{1:k}, C^*_{1:k}) = \frac{1}{n} \sum_{i=1}^{k} \min_{j \in [m]} |C_i - C'_j| \tag{5.1}$$

This cost function is motivated by finding corresponding clusters with the lowest distance, i.e. each generated cluster gets matched with the optimal target cluster. However two generated clusters can be matched with the same target cluster.

Hamming Cost. This motivates the hamming distance as shown in figure 5.2.

$$cost_{hamming}(C_{1:k}, C'_{1:k}) = \frac{1}{n} \min_{\sigma \in \mathbb{S}_k} \sum_{i=1}^{k} |C_i - C'_{\sigma_i}| \tag{5.2}$$

However, the hamming distance consists of an assignment problem to find the optimal matching σ between the generated clusters and the target clusters. Table 5.2 shows how such a matching can look like.

j\i	1	2	3	4	5
1	20	15	30	50	40
2	80	10	15	20	30
3	20	30	50	80	60
4	30	50	40	20	10
5	20	30	40	50	25

Table 5.2.: In order to calculate the hamming distance between two clusterings, we have to calculate the optimal mapping that results in lowest distance for these two clusterings. For random distances between clusterings $C_1^i, \ldots, C_k^i$ and $C_1^j, \ldots, C_k^j$ we can calculate the optimal mapping (blue highlighted cells) in a brute force way or more efficiently with the hungarian method [22][23].

While solving the assignment with a brute force strategy would result in $O(n!)$ complexity, Harold Kuhn introduced the hungarian method to solve the problem in $O(n^4)$ complexity [22]. Later on, James Munkred modified the algorithm to $O(n^3)$ complexity [23]. A detailed explanation of the Hungarian method is included in appendix A.

5.4. Parameter Advising

Our settings average over multiple experiments and show one parameter α that represents the best clustering over all experiments, i.e. the algorithm automatically outputs the best result. For parameter advising, we select the top k values of α for each experiment and calculate the clustering's cost with the best of the k values of α [24]. We select the pool of α-values through the local optima for each experiment. The best k values of α, where k is much smaller than the number of experiments, can then be calculated with an integer optimization problem. A scenario where this setup can be used is by having a domain expert, who can select the best clustering from the k suggested ones.

More formally, we want to to find the optimal parameters $\alpha_1^*, \dots, \alpha_k^*$ such that they optimize the utility u of a clustering instance S and the resulting clustering tree $T(S, \alpha)$ (see equation 5.3). In order to calculate the parameters $\alpha_1^*, \dots, \alpha_k^*$, we first have a look at parameter advising described as facility location problem.

$$\alpha_1^*, \dots, \alpha_k^* = \operatorname*{arg\,max}_{\alpha_1,\dots,\alpha_k} \sum_{i=1}^{N} \max_{j \in [k]} u\big(S, T(S, \alpha_j)\big). \tag{5.3}$$

Facility Location Advising. We create an integer optimization problem for a candidate set $\alpha_1, \dots, \alpha_m$ over the clustering instances $S_1, \dots, S_N$ and introduce the selection parameters $y_1, \dots, y_m \in \{0, 1\}$ that indicate whether α_j is used as one of the k parameters. Also, we denote $x_{ij} \in \{0, 1\}$ as the auxiliary variable with the interpretation that $x_{ij} = 1$ whenever α_j is the best chosen parameter for problem instance S_i. This leads to the following optimization problem that optimizes the overall utility.

$$\begin{aligned}
\operatorname*{arg\,max}_{x_{ij}, y_j} \quad & \sum_{i=1}^{N} \sum_{j=1}^{m} x_{ij} u(S_i, T(S_i, \alpha_j)) \\
\text{subject to} \quad & \sum_{j=1}^{m} y_j = k \\
& \text{for each } i \in [N],\ \sum_{j=1}^{m} x_{ij} = 1 \\
& \text{for each } i \in [N], j \in [M],\ x_{ij} \leq y_j.
\end{aligned}$$

Note that the optimization problem contains three constraints. $\sum_{j=1}^{m} y_j = k$ makes sure that exactly k values are used, the second guarantees that we assign clustering instance to at most one parameter, and the final constraint ensures that we only assign clustering instances to selected parameters. In our experiments, we use IBM ILOG's CPLEX to solve these integer programming problems. However, the computation of the optimal values is very complex, so we also implement a greedy strategy that calculates approximately optimal values more efficiently.

Greedy Parameter Advising. In a convex space, it is very likely that an optimal value α_n^* will also be optimal when calculating $k = n + 1$ optimal values. Leveraging this knowledge, we can iteratively calculate the optimal values $\alpha_1, \ldots, \alpha_m$ step by step, where we first calculate α_1^* that has the largest utility over all clustering instances S and then calculate α_2^* that results in the highest utility combined with α_1^* (see equation 5.4).

$$\sum_{i=1}^{N} \max\{u(S_i, T(S_i, \alpha_1)), u(S_i, T(S_i, \alpha_2))\} \tag{5.4}$$

The results of the experiments with the mentioned datasets are discussed in the following section 6.

6. Experimental Results and Discussion

We evaluated the in chapter 3 and chapter 4 proposed algorithms with the in chapter 5.1 discussed datasets aiming to find new subcategories for the text data and to generate better clusterings overall. The quality of the clusterings was calculated with the in chapter 5.3 explained cost functions.

6.1. Algorithm Selection

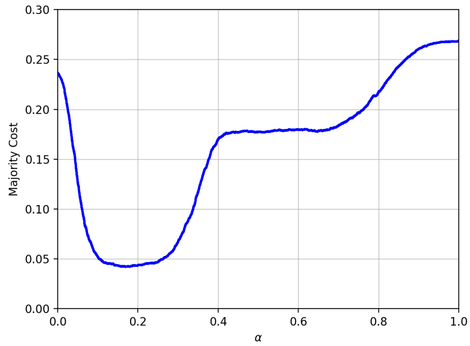
In general we evaluate two different types of experiments that apply for most of the datasets. Only for the synthetic dataset, we evaluate the data distribution shown in figure 5.1.

Batch Data Experiments. In the first one, we evaluate certain data batches, i.e. we subsample the n-th set of points in sorted order for each of the target classes. To generalize the experiments for larger datasets, we average over multiple batches. In our experiments, we evaluate all distinct combinations of k classes, e.g. for multiple datasets we have 10 target classes and use 5 labels for our experiments, i.e. we evaluate all $\binom{10}{5}$ combinations to cover all possible label subsets.

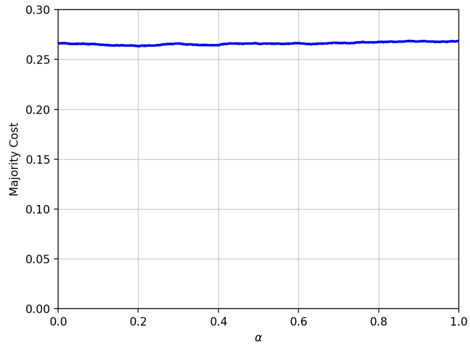
Randomized Experiments. In the other setting, we select the points for certain classes by random. Averaging over a large number of clustering instances allows us to cover a major fraction of the dataset. To clusters a subset of the target classes, we also select the classes by random. Overall, in case both experimental settings agree, we know that the results are generalized well for the underlying data distribution.

6.1.1. Synthetic Data

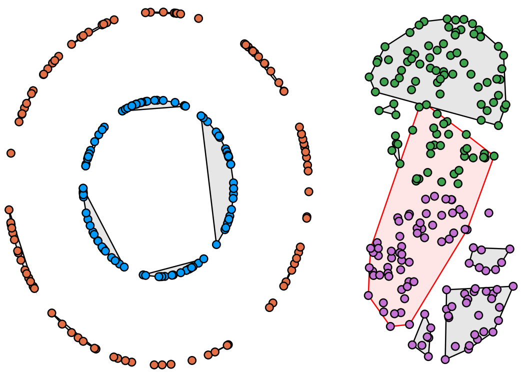
Here we generated 1,000 clustering instances by random given the data distribution shown in section 5.1, i.e. all instances contain four classes, two rings and two disks. In figure 6.1 we observe that all three linkage strategies perform very similarly. Only single linkage does slightly better with an error below 25% while both average and complete linkage are above 25%. Interpolating between single and linkage (a) leads to significantly lower errors (4.2%), where interpolating between average and complete linkage (b) does not lead to improvements. As this example motivates interpolating between single and complete linkage, we elaborate this setting further. Figure 6.1 (c) shows that single linkage does very well identifying the two rings, while it tends to combine the two disk-shaped clusters in a quite early step. On the other hand, complete linkage does very well clustering the two disks, but it tends to combine the two rings (see figure 6.1 (d)).



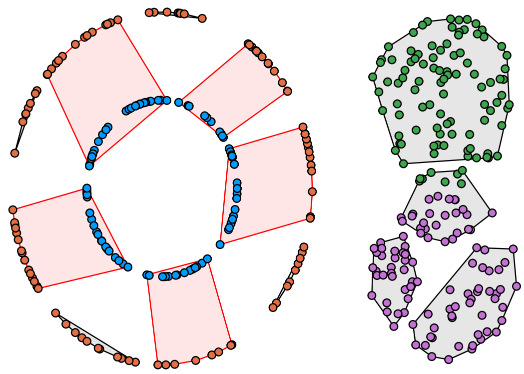
(a) Interpolating between single and complete linkage justifies our motivation for the synthetic experiments as it reduces the error from 23.65% to 4.21%.



(b) In comparison, interpolating between average and complete linkage does not lead to improvements. The error stays mostly constant around 26%.



(c) Single linkage does well for the rings, however it cannot recognize the clusters in the two disks.



(d) Complete linkage does well recognizing the disks, but it fails to correctly identify the rings.

Figure 6.1.: Clustering the synthetic data leads to great improvements when interpolating between single and complete linkage. We observe that single linkage is able to identify the rings very well while complete linkage recognizes the disks. A weighted combination of both is able to plot the overall data very well, while average linkage and complete linkage perform almost identically.

6.1.2. NELL

In order to find new subclusters for the NELL data, we cluster each of the 32 main categories seperately. This results in 32 different clustering tasks, where we compare the results of each clustering task with the target labels using the majority distance function. We will receive a cost function $cost(\alpha)$, that shows us for which value of α the resulting clusterings are good, for each category. By averaging all cost functions, we know for which values of α the α-linkage performs well in general. Beside having a value of α that can be used for other clustering tasks, the experiments also give different representation levels of clusters that are discussed in section. First, we started evaluating all tasks with a maximum of 250 points per task. Figure 6.2 shows the result for all three different interpolation strategies.

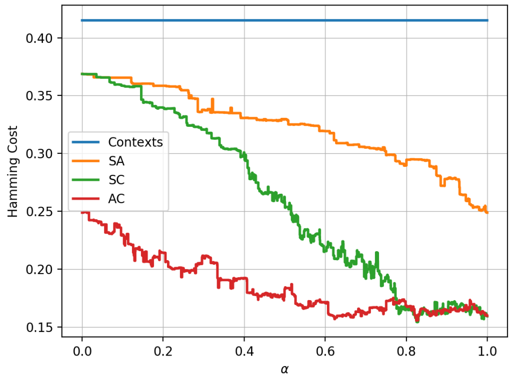


Figure 6.2.: α-linkage using 250 points for each clustering instance gives minor improvements for the NELL data when clustering between single and complete and average and complete linkage. As complete linkage performs best of our input strategies, interpolating between single and average linkage does not lead to improvements.

We see minor improvements when clustering between single and complete and average and complete linkage. On the other hand, interpolating between single and average linkage did not lead to any improvement. In order to evaluate the results further, we have a closer look at the curves and see that the overall improvement we get is 0.53%, a reduction from 15.9725% (complete linkage) to 15.4422% ($\alpha_{SC}(0.826)$) as shown in table 6.1. An interesting observation is that while single linkage performs very poor overall, interpolating between single and complete linkage gives a better improvement than interpolating between average and complete linkage. To evaluate these experiments we are using the Majority Distance, as for such a large number of target clusters calculating the Hamming distance is not efficient.

Strategy	Majority Cost
Single Linkage	0.36871
Average Linkage	0.248913
Complete Linkage	0.159725
$\alpha_{SC}(0.826)$	0.154422
$\alpha_{AC}(0.826)$	0.155697

Table 6.1.: Our proposed algorithm reduces the NELL cost by $\Delta cost = 0.53\%$ when using a maximum of 250 points for each class.

As the algorithm became a lot more efficient during this work, we scaled up the algorithms to use 1,000 instead of 250 points per class. Figure 6.3 shows that in general the error is slightly higher. This is because our experiments contain more different classes. Overall, we again see slight improvements that are shown in table 6.2. Compared to the previous experiments, the improvements were a bit bigger (1.2078% leading to an error of

16.6742%), however the overall curves look very similar. In this setting, we also evaluated the parameter advising for the first 10 parameters α^* (see figure 6.4).

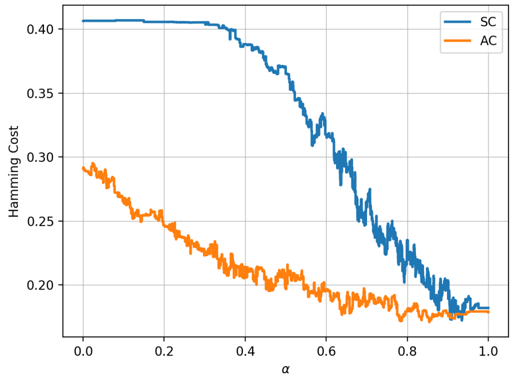


Figure 6.3.: α-linkage using 1000 points for each clustering instance gives minor improvements for the NELL data when clustering between single and complete and average and complete linkage.

Strategy	Majority Cost
Single Linkage	0.36871
Average Linkage	0.291202
Complete Linkage	0.17882
$\alpha_{SC}(0.918)$	0.166742
$\alpha_{AC}(0.855)$	0.171083

Table 6.2.: Our proposed algorithm reduces the NELL cost by $\Delta cost = 1.2078\%$ when using a maximum of 1000 points for each class.

Also, we evaluated the corresponding clusters. As α-linkage uses agglomerative hierarchical clustering, we can extract clusters at different levels starting with each noun phrase as its own cluster. Tables 6.3, 6.4 and 6.5 show some examples for discovered categories.

Luxury Room	**Bathroom**	**Guest Room**	**Suite**
spacious living room	large ensuite bathroom	elegant rooms	luxurious suites
comfortable living room	spacious marble bathroom	three guest rooms	one bedroom suites
guest room	one bathroom	large guest rooms	spacious suites
lounge room	full bathroom	deluxe guest rooms	deluxe suites
living room	upstairs bathroom	guests rooms	guest suites
superior room	large bathroom	spacious air conditioned rooms	bedroom suites
sleeping room	ensuite bathroom	furnished guest rooms	whirlpool suites
main bedroom	elegant bathroom	comfortable guest rooms	three suites

Table 6.3.: Proposed Subcategories for “Office Building Room”.

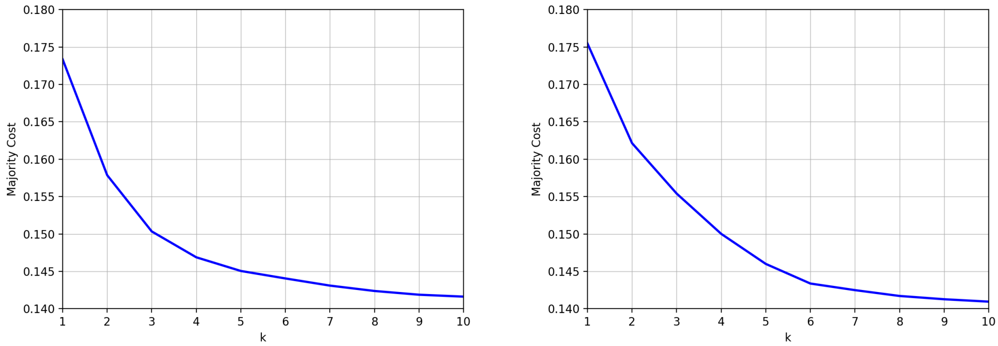


Figure 6.4.: α-linkage using 1000 points for each clustering instance gives minor improvements for the NELL data when clustering between single and complete (left) and average and complete linkage (right).

Shoes	Uniform/Costume	Pants	Casual	Specialized
shoes	costume	kneepants	stocking cap	long stockings
high heel shoes	work uniforms	baggy pants	workout clothes	wide brimmed hat
sensible shoes	outfits	loose fitting pants	casual clothes	casual wear
old shoes	period costume	slacks	baseball caps	black stockings
pointe shoes	folk costumes	black shorts	skull caps	wear socks
dark shoes	halter top	special clothing	ball caps	high heels
spira shoes	period costumes	white shorts	evening clothes	surf wear
mens shoes	costumes	underpants	ball cap	wear gloves

Table 6.4.: Proposed Subcategories for “Clothing”.

Stove/Oven	Machines	Bowls	Baking Sheets
full size stove	cookie cutters	large mixing bowl	oiled baking sheet
full size cooker	automatic washing machine	large serving bowl	rimmed baking sheet
red hot stove	washing machine	small bowl	large baking sheet
plastic jug	bread machine	single bowl	small baking sheet
toaster	cookie cutter	separate bowl	prepared baking sheet
greased baking dish	coffee machine	shallow bowl	ungreased baking sheet
wood burning pizza oven	cooking spray	separate mixing bowl	hot plate
ceramic top stove	coffee grinder	large bowl	greased baking sheet

Table 6.5.: Proposed Subcategories for “Kitchen Item”.

Bag-of-Contexts. In addition to using the original features, we also use the bag-of-contexts representations to evaluate these experiments. The features are represented in a sparse very high-dimensional dataset. However, the feature representation lead to a constant error of $\approx$ 41% as shown in figure 6.2, i.e. this feature representation did not lead to improvements over the existing feature representation that was created by domain experts. Also, the error stayed constant for all three discussed linkage strategies as well as the interpolation between them. This means that the feature representation did not lead to good clusterings and α-linkage could not overcome the problem in this case.

6.1.3. MNIST

Pixel Representation. For the MNIST images, we evaluate both described experimental settings with combinations of five out of the ten target classes. In addition to using the raw pixel features. To cluster the data, we set up $\binom{10}{5} = 252$ different experiments by selecting all combinations of five out of the ten labels. In order to do so in efficient time, we subsample the dataset to 200 points for each label, so one experiment will cluster 1000 points. First, we evaluated the results for both average to complete and single to complete linkage for several batches. Note that we do not discuss the interpolation between single to average linkage in this and the following paragraphs, as experiments did not lead to improvements. First, we show the experiments for the six first batches $b_i, i \in \{0, 1, 2, 3, 4, 5\}$ for interpolating between single and complete linkage.

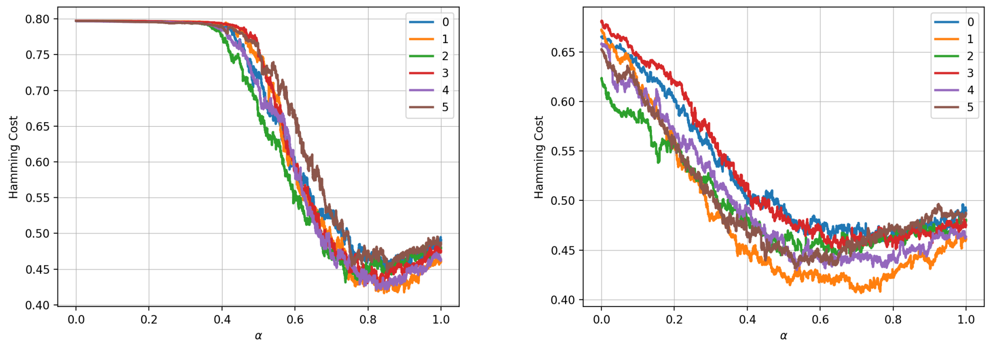


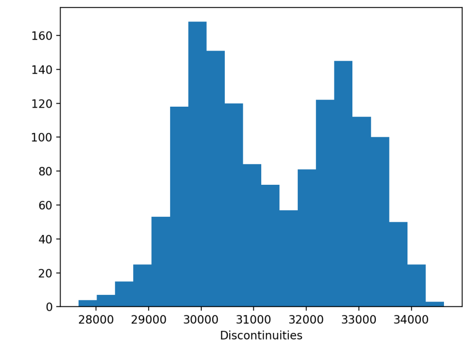
Figure 6.5.: Over the first six batches of the MNIST data, interpolating between single and complete linkage shows a similar behavior (left) while interpolating between average and complete linkage leads to bigger differences (right).

As shown in figure 6.5 (left), the clustering over the first six batches leads to very similar curves with slightly different errors. Table 6.6 evaluates the results in more detail.

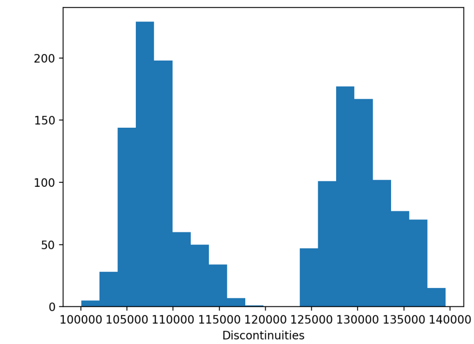
Strategy	Batch 0	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5
Single Linkage	0.796901	0.797345	0.797171	0.797405	0.796766	0.797024
Complete Linkage	0.490468	0.461063	0.479825	0.475329	0.463321	0.487111
α_{opt}	0.87228	0.84419	0.778498	0.83199	0.82338	0.852251
$cost_{opt}$	0.450012	0.416433	0.431143	0.423786	0.421103	0.446032
$\Delta cost$	4.0456%	4.463%	4.8682%	5.1543%	4.2218%	4.1079%

Table 6.6.: α-linkage reduces the cost of the MNIST dataset by up to $\Delta_{max} cost = 5.1543\%$ when interpolating between single and complete linkage.

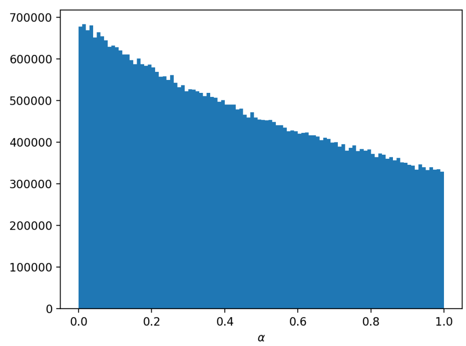
Table 6.6 leads to several observations. Clustering points of five classes with a random guess will result in an error of 80%. As for all batches single linkage results in an error between 79% and 80%, we note that single linkage performs similar than a random guess would. Thus, single linkage is not suitable for the MNIST data. In comparison, complete linkage results in errors below 50% on just using the pixel data. It is not necessarily a great result, but it indicates that grouping high-dimensional pixel features with unsupervised learning can work. Also, we note that the parameter α_{opt} doesn't vary that much and also we notice in figure 6.5 that for $\alpha \in [0.75, 1.0)$ we outperform complete linkage in all cases. As the results are very similar for the used batches, we also average over the batches in figure 6.8. After we evaluated six batches with 252 experiments each, we have a solid base to compare how the different interpolation strategies behave. In figure 6.6 we show the overall amount of discontinuities in a histogram for interpolating between single and complete and between average and complete linkage. Also, we show a histogram that indicates how many splits happen in given regions for the value α.



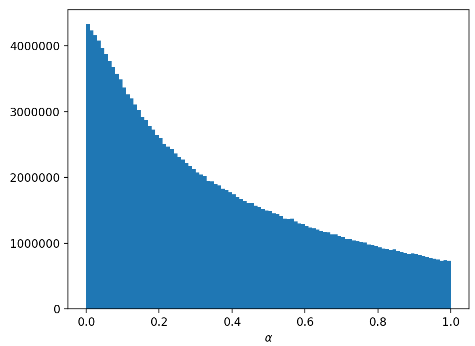
(a) Interpolating between average and complete linkage results in 27,000 - 35,000 discontinuities.



(b) Interpolating between single and complete linkage results in at least 100,000 discontinuities.



(c) The amount of discontinuities is linearly decreasing when interpolating between average and complete linkage.



(d) The amount of discontinuities is exponentially decreasing when interpolating between single and complete linkage.

Figure 6.6.: Interpolating between single and complete linkage leads to more than three times as many discontinuities for the MNIST batch experiments than interpolating between average and complete linkage. More discontinuities happen close to single linkage.

In figure 6.6 we observe that interpolating between single and complete linkage leads to a larger amount of discontinuities than interpolating between average and complete linkage, evaluating the MNIST batch experiments leads to more than three times as many discontinuities. We also observe that a large amount of discontinuities occur close to single linkage, i.e. we observe a huge variety of different clustering trees. Precisely, around single linkage ($\alpha \longrightarrow 0$) we observe four times as many different clusterings as for complete linkage ($\alpha \longrightarrow 1$). An interesting observation is that for $\alpha \approx 0.25$ we have half as many discontinuities as for single linkage and twice as many as complete linkage, i.e. we have an approximately exponential decrease during the interpolation.

Strategy	Hamming Cost
Single Linkage	0.797102
Complete Linkage	0.476186
α_{opt}	0.857
$cost_{opt}$	0.439207
$\Delta cost$	3.6979%

Table 6.7.: Over the first 12,000 points of the MNIST dataset interpolating between single and complete linkage improves hamming cost by 3.7%

Figure 6.8 and table 6.7 show that by applying α-linkage interpolating between single and complete linkage we improve the hamming cost by 3.7% over the first six data batches, i.e. the first 12,000 points of the dataset. Next, we evaluate the randomized experiments for the same interpolation method, where we average over 512 experiments that are run with random label subsets and randomly selected points for each of the selected labels. Figure 6.7 shows that in this setting we obtain a very similar curve as in the other setting.

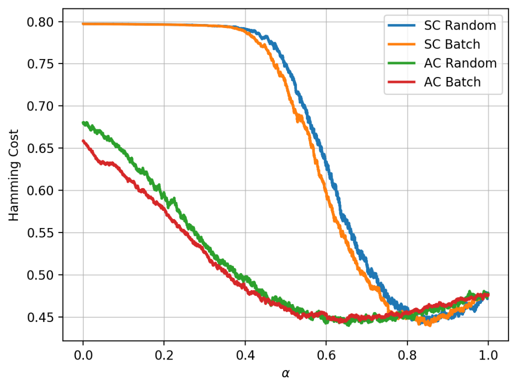


Figure 6.7.: Comparing the averaged batch and the random settings leads to very similar curves for interpolating between single and complete and between average and complete linkage.

Strategy	Hamming Cost (Batch)	Hamming Cost (Random)
Single Linkage	0.797102	0.797215
Complete Linkage	0.476186	0.476355
α_{opt}	0.857	0.857
$cost_{opt}$	0.439207	0.440932
$\Delta cost$	3.6979%	3.5423%

Table 6.8.: Evaluating the randomized setting leads to exactly the same parameter α_{opt} and a similar cost improvement as in the batch setting for the MNIST data.

Table 6.8 compares the results for both settings when interpolating between single and complete linkage. We obtain very similar results for single and complete linkage. Also, the optimal parameter α_{opt} is the same in both settings leading to similar improvements in the hamming cost. This means that α-linkage is robust over the entire MNIST distribution and with an improvement of more than 3% towards complete linkage it outperforms both used linkage strategies by a major difference. In addition, we also evaluate the greedy parameter advising for the previous experiments. By using $k = 3$ parameters α^* the cost drops more than 5% in addition to less than 38%. In comparison to the best linkage strategy, i.e. complete linkage, this is an improvement of $\approx$ 10%.

Similar to that, we also interpolate between average and complete linkage and evaluate both the batch and the random setting. Figure 6.5 and table 6.9 show that the results of the different batches vary much. On the one hand, the parameters α_{opt} have a wider range ($\alpha_{opt} \in [0.53, 0.81]$), but on the other hand, we get slightly larger improvements for the hamming cost in comparison to complete linkage.

Strategy	Batch 0	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5
Average Linkage	0.664952	0.672583	0.623325	0.679929	0.657857	0.652774
Complete Linkage	0.490468	0.461063	0.479825	0.475329	0.463321	0.487111
α_{opt}	0.7869	0.7124	0.634	0.807697	0.536073	0.5305
$cost_{opt}$	0.458167	0.406563	0.440964	0.451063	0.429849	0.431631
$\Delta cost$	3.2301%	5.45%	3.8861%	2.4266%	3.3472%	5.548%

Table 6.9.: α-linkage reduces the cost of the MNIST dataset by up to $\Delta_{max}cost = 5.548\%$ when interpolating between average and complete linkage.

Figure 6.7 shows the comparison between the batch (left) and the random experiments (right). In general, we obtain similarly looking curves, but elaborate the results further in table 6.10.

Summarizing, we also obtained very positive results for d_{AC}, however the results were not as stable as for d_{SC}. In our experiments, we notice that d_{SC} results in more discontinuities (factor $\approx$ 3) than d_{AC}. This may be because the distance $d(\alpha)$ is wider spread for d_{SC}, i.e. $|d_{SC}(\alpha = 1) - d_{SC}(\alpha = 0)| > |d_{AC}(\alpha = 1) - d_{AC}(\alpha = 0)|$. However d_{AC} is dependant on more points, so it may be an indicator for this observation, but not a proof. Parameter advising is useful with a small value k already and reduces the costs for $k = 3$ by $\approx$ 5% in addition.

Strategy	Hamming Cost (Batch)	Hamming Cost (Random)
Average Linkage	0.65857	0.679936
Complete Linkage	0.476187	0.476328
α_{opt}	0.633	0.656
$cost_{opt}$	0.44314	0.439632
$\Delta cost$	3.3047%	3.6696%

Table 6.10.: Comparing between average and complete linkage for the MNIST data leads to slight differences between the batch and the random setting.

Learning MNIST features. Differently to just using the raw pixel features, we here apply preprocessing techniques with the intention to generate more accurate clusterings. As in section 2.3.2 described, we use a Convolutional Neural Network to learn a more robust and lower-dimensional feature representation. Therefore, we use the in appendix B described architecture, train the network with all data and then extract the features by cutting off the last three layers of the network. This then results in a learned 128-dimensional representation for each image.

Figure 6.8 (a) shows that single linkage still performs poorly, however the error for both complete linkage and the interval in between are much lower. Also, we note that the improvement using α-linkage is large over both settings. However, a Convolutional Neural Network aims at recognizing the characters, so training on all images might be the sole cause of our improvements. Thus, it is more relevant for our experiments to either train the network on a subset of the data or to train the network on a different task in order to transfer the knowledge to unseen data or to a different task.

Learning Subsets of the MNIST Data. As our goal is also to cluster unseen data, we evaluated another setup, where a CNN was trained on a subset of the dataset. In a first attempt, we trained it on the labels $\{0, 1, 2, 3, 4\}$ that are represented with 30,000 of the 60,000 points in the dataset. Figure 6.8 (a) shows that clustering unseen points (i.e. the CNN did not use these points for training) still results in a lower error than using the raw pixel features where combining seen and unseen points leads to results that are comparable to clusterings with features extracted from a neural network that was trained with all digits. In average, complete linkage resulted in an error of 22.1%. The cost for $\alpha_{opt} = 0.67$ is 20.7% and makes an improvement of 1.4%. Interesting especially in this setting are the different results of seen and unseen data. In machine learning, the task of applying knowledge to unseen data is commonly known as few-shot learning [25]. While the error was 0.2% for large parts of the seen data (i.e. clustering the digits $\{0, 1, 2, 3, 4\}$), the optimal cost for the unseen data (i.e. clustering the digits $\{5, 6, 7, 8, 9\}$) was 24.7% for $\alpha_{opt} = 0.76$.

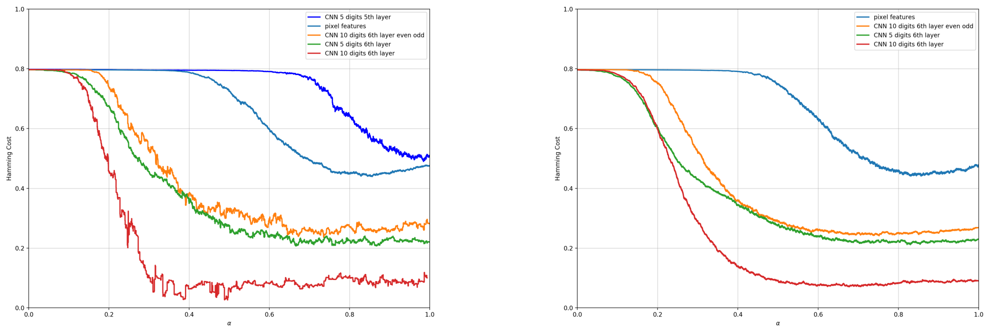
In the random setting, we again used the same feature vectors. Figure 6.8 (b) shows that we again obtain major improvements. While for using pixel features when interpolating between single and complete linkage the error was 47.6%, these feature representations lead to 22.0% for complete linkage, i.e. an improvement of 27.6%. For using $\alpha_{opt} = 0.666$, the error is 20.7%, so another improvement of 1.3%. On the other hand, in this experimental setting, we are not able to compare between seen and unseen data samples as the experiments are completely randomized. Nonetheless, the results are almost identical to the ones shown for the batch setting leading us to the assumption that the clusterings are very stable over the MNIST dataset.

Learning Broader Features. In comparison to extracting the feature vectors from the 6th layer, we also evaluated feature vectors before the dense layer that compromises the feature dimensionality from 9216 to 128 dimensions. The higher dimensionality might represent more information that however does necessarily have to be important for the clustering tasks. In this setting, the neural network was agained trained with the images showing the digits $\{0, 1, 2, 3, 4\}$. Figure 6.8 (a) shows that even for the trained digits, the clusterings were not accurate on the 9216-dimensional features. More precisely, clustering the labels the network was trained on results in an optimal cost of 40.0% that is still slightly better than using the raw pixel features, however when averaging over all instances, the optimal cost was 49.0% for $\alpha_{opt} = 0.98$. This means that the results are worse than using the raw pixel features. Another interesting observation is that the cost stays mostly constant at 0.8 for $\alpha \in [0.0, 0.6]$. As this matches a random guess for distinguishing between five classes, this observation leads to the result that the broader features extracted from an earlier layer of the neural network do not give a helpful feature representation for clustering tasks. We did not consider these features for further usage.

Learning Even and Odd Numbers. Beside training a neural network on recognizing all digits separately, another learning task to generate feature representations that we used is to learn if an image shows an even or an odd digit. In this setting, we trained the CNN on all images and extracted the feature vectors from the sixth layer. The used network had the same architecture as the one used in the earlier experiments with the only difference of two neurons in the output layer. Figure 6.8 (a) shows that with features trained on a different learning task we still can improve the overall clustering. Complete linkage resulted in a cost of 28.1% for the first data batch, where the optimal alpha $\alpha_{opt} = 0.74$ led to 23.6%, an improvement of 4.5%. The results are only slightly worse than the ones of a network trained to distinguish the digits $\{0, 1, 2, 3, 4\}$. Parameter advising lowers the cost another 5.2% for $n = 3$ values $\alpha^* \in \{0.74, 0.65, 0.76\}$.

In figure 6.8 (b), we see the results for clustering features extracted from a neural network that was trained on separating even and odd numbers in the random setting. In general, complete linkage results in an error of 26.8%. $\alpha_{opt} = 0.75$ improved the error by 2.7% to a cost of 24.1%. We notice that the parameter α_{opt} is almost identical. The improvement differs in both settings, however it is still a major improvement that α-linkage achieves.

Summarized MNIST Results. Different experimental setups were discussed in this section. First, raw pixel features were used for clustering. Later on, features extracted from Convolutional Neural Networks were used. There, we trained a network on all digits and extracted the feature vectors from the 6th layer of the network that represents each image encoded in a 128-dimensional vector. We used the same representation coming from a network trained on a subset of the images. In addition, we extracted feature vectors from the 9216-dimensional 5th layer of the network that was trained on a subset of the characters. Figure 6.8 gives an overview about the results of the different settings for both the 252 experiments evaluating all different combinations of five labels within the first data batch as well as the randomized experiments where we evaluated 512 experiments with radomized digits and points from the entire data.



(a) Evaluating the experiments of all combinations of five labels within the first batch shows strong discontinuities.

(b) Evaluating 512 experiments with randomized digits and points shows similar results with smoother curves.

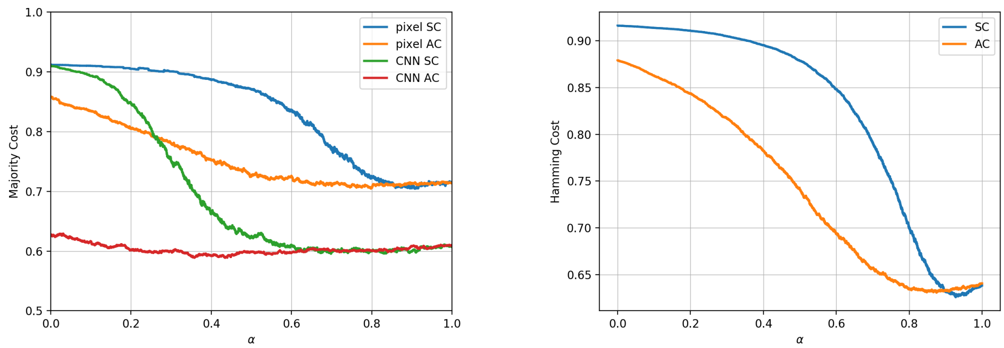
Figure 6.8.: The previously discussed experiments led to different results. While using the features extracted from the fifth layer of the neural network did not lead to good results, features extracted from the sixth layer led to huge improvements.

6.1.4. Omniglot

Intra-Alphabet Experiments. The omniglot dataset contains 30 different alphabets that clustered independently. We cluster all alphabets independently and average over all clustering instances. To see the results for specific alphabets, please see appendix D. Here, we average the results over all alphabets for $\mathcal{D}_{\mathrm{SC}}$ and $\mathcal{D}_{\mathrm{AC}}$. Figure 6.9 shows in both settings complete linkage performs best and we slightly outperform complete linkage. Interpolating between single and complete linkage leads to an error of 71.5% for complete linkage while the optimal parameter $\alpha = 0.908$ leads to an error of 70.5% that makes an improvement of 1.0%. When interpolating between average and complete linkage, complete linkage has an error of 71.5%, while $\alpha_{opt} = 0.798$ results in an error of 70.5% that makes an improvement of 1.0%. We notice that even with the slight improvements we obtain, the error is still very high. This may be caused by the large number of classes (up to 22) used in the omniglot dataset, where a random guess will result in an error of up to $\approx 95.5\%$ depending on the selected alphabet. Again, we observe that single linkage does not perform well at all, however complete linkage leads to significant improvements over a random guess.

CNN Features. In addition to clustering the raw pixels, we also used the previously discussed CNN architecture trained on all MNIST images to create a better feature representation. We again clustered all alphabets separately, where we show the results for all alphabets in appendix D. Figure 6.9 shows that we the feature representations lower the error a lot. When interpolating between single and complete linkage, we note that single linkage does not perform better with the new features. However, complete linkage leads to an error of 61.0% that is an improvement of 10.5% over the raw pixels. In addition for $\alpha_{opt} = 0.835$, the error is 59.6%, i.e. an improvement of 1.4% over complete linkage. When interpolating between average and complete linkage, complete linkage leads to an error of 61.0% and $\alpha_{opt} = 0.435$ reduces the cost by another 2.1% to 58.9%.

Inter-Alphabet Experiments. In addition, we tried to cluster characters taken from different alphabets. In our setting, we selected one character from each alphabet randomly and ran multiple repetitions of this setting where each run contains 30 classes, i.e. 600 points. One advantage of this setting is that each run has the same amount of target cluster that made it easier to average the results over all experiments. Figure 6.9 shows that also for clustering characters of different alphabets the improvements are rather small. Averaged over 250 runs the improvement shown above was 1.0%.



(a) Running experiments within individual alphabets (intra-alphabet), we obtain major improvements over complete linkage for different strategies.

(b) α-linkage again leads to noticeable improvements when clustering characters combined from different alphabets (inter-alphabet).

Figure 6.9.: For the Omniglot data, we evaluated pixel features for interpolating within each individual alphabet (a) and across different alphabets (b). Also, we evaluated CNN features of a network trained on the MNIST data that led to major improvements in the intra-alphabet setting (a).

6.1.5. CIFAR

In addition to these experiments, we will try to cluster as diverse as possible superclasses of the CIFAR100 dataset by manually picking the five superclasses fish, flowers, household furniture, people and vehicles 1. For each superclass we pick one subclass and evaluate the results for all $5 \cdot \binom{5}{1} = 25$ different combinations of subclasses. In addition to the experiments with $k = 5$ clusters, we compare these results to the results for picking two different subclasses of each superclass ($5 \cdot \binom{5}{2} = 50$ different experiments) resulting in $k = 10$ clusters and also for picking three different subclasses ($5 \cdot \binom{5}{3} = 50$ different experiments) resulting in $k = 15$ clusters.

In comparison to picking as diverse as possible superclasses, we also evaluate the performance for as similar as possible subclasses. Similar subclasses are already given in the dataset through the subclasses within one superclass. We then evaluate the majority and the hamming cost for each superclass and again average the cost over all 20 superclasses to evaluate an optimal value for the parameter α.

6.2. Metric Learning

Omniglot. First we present results on the omniglot dataset [21].

Instance distributions. We conduct experiments for two distributions over clustering instances on the Omniglot dataset, both inspired by prior work on few-shot meta-learning. Following [26] and [27], our first instance distribution selects $k = 5$ random characters (independently of their alphabet) and takes the 20 examples of those 5 characters resulting in a dataset with $n = 100$ examples. The target clustering is given by the 5 character labels. We refer to this instance distribution as the MN/MAML distribution. Second, following [28], our second instance distribution generates clustering instances that have a variable number of target clusters and each clustering task involves related characters. Specifically, to generate an instance, we pick one alphabet from Omniglot uniformly at random, choose the number of classes k uniformly between 5 and 10, and then choose k characters from that alphabet uniformly at random. As before, the clustering instance consists of all $20k$ examples for the chosen characters, and the target clustering is given by the k character labels. We refer to this instance distribution as the MD distribution.

Distance metrics. We present results for mixing three different distance metrics on the Omniglot data. This dataset provides two different representations for each example: a 105×105 black and white image of the character, and stroke data describing the path that the pen took when writing that character (i.e., a time series of (x, y) coordinates). We use a hand-designed distance metric based on the stroke data, as well as features derived from a convolutional neural network trained on MNIST.

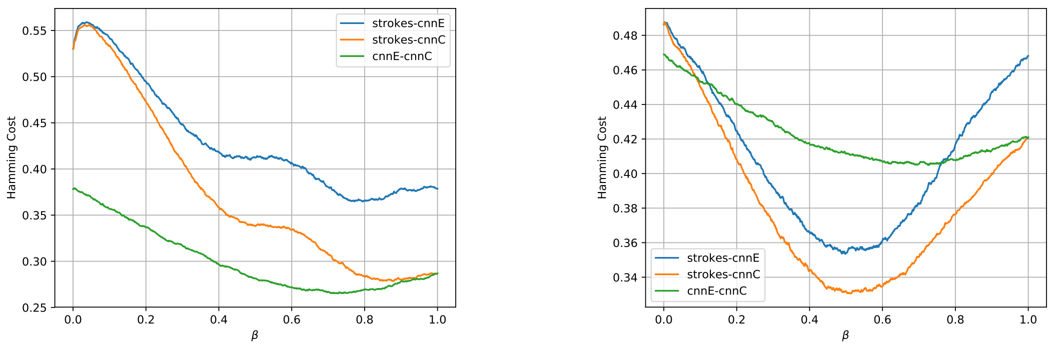
- (Stroke distance) Given two pen stroke trajectories $s = (x_t, y_t)_{t=1}^{T}$ and $s' = (x'_t, y'_t)_{t=1}^{T}$, we define the distance between them by

$$d(s, s') = \frac{1}{T + T'} \left(\sum_{t=1}^{T} d\big((x_t, y_t), s'\big) + \sum_{t=1}^{T'} d\big((x'_t, y'_t), s\big) \right),$$

where $d\big((x_t, y_t), s'\big)$ denotes the Euclidean distance from the point (x_t, y_t) to the closest point in s'. This is the average distance from any point from either trajectory to the nearest point on the other trajectory.

- (CNN-C) Next we construct a distance metric using the image representation of each example. In particular, we train a convolutional neural network for classifying the 10 digits of MNIST. Then we use this network to obtain embeddings of each omniglot digit. Finally, to measure the distance between two examples, we use the cosine-distance between them (that is, the angle between the two feature embeddings).
- (CNN-E) The final metric uses the same neural network embedding as above, except measures distances between two examples using the Euclidean distance.

Results. Figure 6.10 shows our empirical results for learning the best combinations of the above metrics on both instance distributions over the Omniglot dataset. For each pair of metrics and each instance distribution, we plot the average Hamming error of the cluster tree produced by the algorithm as a function of the mixing parameter β averaged over $N = 2000$ clustering instances sampled from the underlying distribution. For both distributions, the best mixture of two metrics performs better than the best fixed single metric. On the MN/MAML distribution, the best average performance is obtained when mixing the Euclidean and cosine distances for the MNIST CNN features with $\beta = 0.727$ achieving an average Hamming error of 0.263. In contrast, using the cosine distance on the MNIST CNN features is the best single metric and has an average Hamming error of 0.289, yielding an improvement of 0.026. For the MD instance distribution, the stroke distance appears to be more useful. The best performance is achieved when mixing the stroke distance and the cosine distance on the MNIST CNN features with $\beta = 0.514$ and achieves error 0.33, while the best fixed metric has error 0.42, leading to an improvement of 0.09.



(a) MN/MAML Instance Distribution (b) MD Instance Distribution

Figure 6.10.: Learning the best distance metric for Omniglot.

7. Conclusion

In this work, we developed a data-driven solution to find the best algorithm (here: linkage strategy) for clustering tasks on a given data set. In section 3 we introduced an algorithm to efficiently interpolate between the in section **??** described linkage strategies. The final α-linkage algorithm uses a line sweep approach to find the pairwise distance function that leads to merges with the lowest resulting cost, i.e. the optimal clusterings.

While the in section **??** described approaches are less efficient, our approach is able to analyze large parts of the in section 5 discussed real-world data sets using cloud computing. We take multiple key observations from the experiments in section 6:

- Interpolating between single and complete linkage overcomes the hurdles of clustering data where both single and complete linkage perform better for some certain parts of the data. While single and complete linkage lead to an error of approximately 25% for our synthetic data, in some experiments α-linkage leads to optimal clusterings.
- For all our real-world data sets, we can sort the linkage strategies according to the quality in following order (best first):
    1. Complete Linkage
    2. Average Linkage
    3. Single Linkage
- Interpolating between single and average linkage does in general not lead to significant improvements.
- When interpolating between single and complete or average and complete linkage, α-linkage outperforms the used linkage strategies in **all** of our experiments.
- Parameter Advising is a very powerful tool to provide more accurate clusterings. Our greedy implementation allows to calculate the optimal parameter in an approximately optimal way very efficiently.

To be more precise, we show the results for the different datasets. In table 7.1 we compare the single to complete linkage interpolation for all the discussed data sets in the batch setting. We notice that we achieve major improvements for all data sets and, excluding the synthetic data, receive similar values for α_{opt}. This knowledge may be useful for transfer learning between different data sets in feature work.

	Synthetic	NELL	MNIST	Omniglot	CIFAR10	CIFAR100
α_{opt}	0.169	0.918	0.857	0.908	TODO	TODO
Δ_{cost}	22.70%	1.20%	3.70%	1.0%	TODO	TODO

Table 7.1.: α-linkage reduces the cost of the MNIST dataset by up to $\Delta_{max}cost = 5.1543\%$ when interpolating between single and complete linkage.

As we had implemented a framework to efficiently interpolate between inter-cluster distances, we also adapted it to interpolate between point distances in section 4. We generated image features with Convolutional Neural Networks and text features with Word Embeddings and showed in section 6 that we outperform all used feature representations.

Future Work. Nevertheless we can think of further additions to this work. The proposed algorithms only work for linear interpolation between two algorithms. In this setting, proving that in each interval we perform the same optimal merge is mostly trivial. As an adaption of our algorithms, we can think of interpolating between more algorithms, e.g. we could interpolate between single, average and complete linkage with a distance such as the following:

$$\mathcal{D}_{\mathrm{SAC}}(X,Y) = \alpha_1 \cdot \min_{x\in X, y\in Y} d(x,y) + \alpha_2 \cdot \frac{1}{|X||Y|} \sum_{x\in X, y\in Y} d(x,y) + (1-\alpha_1-\alpha_2) \cdot \max_{x\in X, y\in Y} d(x,y)$$

s. t. $\{\alpha_1, \alpha_2\} \in [0,1]$ and $\alpha_1 + \alpha_2 \leq 1$

However, the distance functions $d_{(\alpha_1,\alpha_2)}$ now are not linear anymore, as they depend on the two parameters α_1 and α_2. This means that our suggested line sweep approach will not work anymore, as the distance depending on the two parameter spans a three-dimensional space, where each split is a convex hull instead of a linear subspace. Figure 7.1 shows the interval split depending on α on the left, and the split depending on α_1 and α_2 on the right. Our example shows a very optimistic example where the splitting linear functions are parallel, however this might often not be the case, i.e. we will receive more different regions where it will be less intuitive to find the borders where and when one merge is preferred over another (e.g. see figure 7.2).

In addition, we can apply the introduced algorithm to more data sets and data domains, such as the sets contained in the UCI Machine Learning Repository [29]. For instance, we could compare different data sets within one specific data domain. Say we evaluate a variety of different image data sets and compare the optimal values of α. An interesting observation would be, if we could reuse the optimal parameter from different data sets within the same or even across other domains. In our experiments, we often found similar optimal values in the range $[0.7, 0.9]$ while other ranges mostly did not lead to good results (e.g. $[0.0, 0.5]$). We could use this knowledge for further experiments by e.g. evaluating only smaller regions or by putting more emphasis on given parameters in general. We can then also evaluate other domains, such as (partially) labeled voice data sets and compare the results across different data domains.

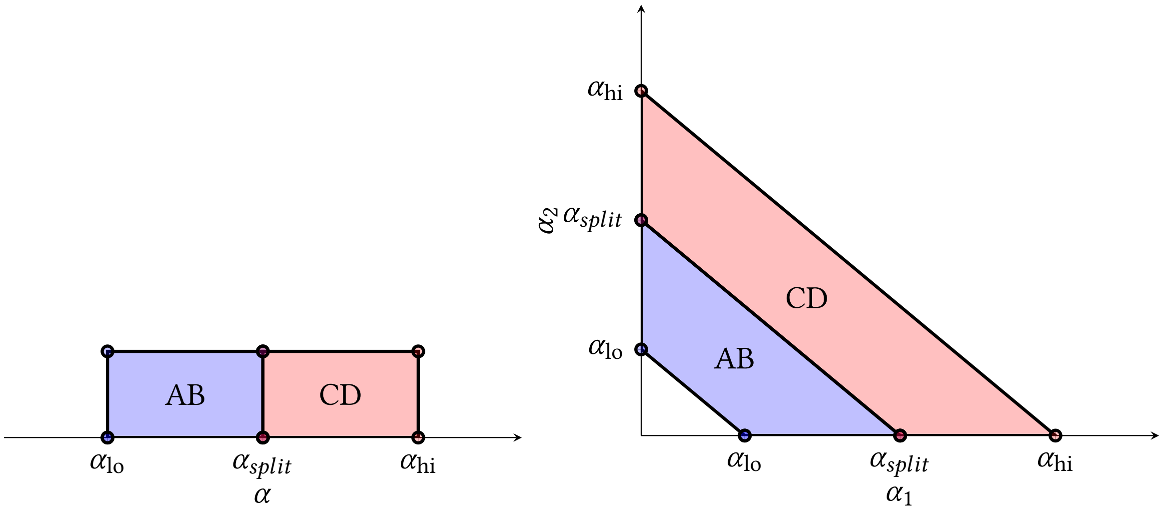


Figure 7.1.: While in this work, the split between different merges was based on a linear function $d(\alpha)$ (left), it will be more difficult to evaluate the merges when interpolating with two weight parameters α_1 and α_2, where the merges will be represented as a convex hull in the α_1-α_2-space.

Also, we can imagine applying this framework to (a) other clustering algorithms than agglomerative hierarchical clustering and (b) other tasks than clustering.

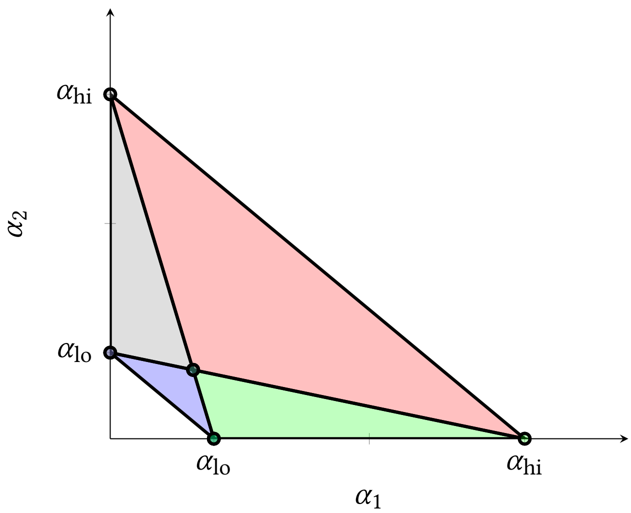


Figure 7.2.: Finding the merges in the different regions can be more challenging when interpolating with two weight parameters α_1 and α_2.

Bibliography

[1] Oren Zamir and Oren Etzioni. "Web document clustering: A feasibility demonstration". In: *SIGIR*. Vol. 98. Citeseer. 1998, pp. 46–54.

[2] Jaydeep Balakrishnan et al. "Product recommendation algorithms in the age of omnichannel retailing–An intuitive clustering approach". In: *Computers & Industrial Engineering* 115 (2018), pp. 459–470.

[3] Wen-Yan Lin et al. "Dimensionality's Blessing: Clustering Images by Underlying Distribution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5784–5793.

[4] Zengyou He, Xiaofei Xu, and Shengchun Deng. "Discovering cluster-based local outliers". In: *Pattern Recognition Letters* 24.9-10 (2003), pp. 1641–1650.

[5] Samuel Fosso Wamba et al. "How 'big data'can make big impact: Findings from a systematic review and a longitudinal case study". In: *International Journal of Production Economics* 165 (2015), pp. 234–246.

[6] Rishi Gupta and Tim Roughgarden. "A PAC Approach to Application-Specific Algorithm Selection". In: *CoRR* abs/1511.07147 (2015). arXiv: `1511.07147`. URL: `http://arxiv.org/abs/1511.07147`.

[7] Juan R Cebral et al. "Combining data from multiple sources to study mechanisms of aneurysm disease: tools and techniques". In: *International journal for numerical methods in biomedical engineering* 34.11 (2018), e3133.

[8] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[9] Pamela Guevara. "Inference of a human brain fiber bundle atlas from high angular resolution diffusion imaging". In: (Oct. 2011).

[10] Eric Sven Ristad and Peter N Yianilos. "Learning string-edit distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.5 (1998), pp. 522–532.

[11] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: `http://www.aclweb.org/anthology/D14-1162`.

[12] Carnegie Mellon University Machine Learning Department. *CMU NELL all-pairs data, version 02-Feb-2012*. `http://rtw.ml.cmu.edu/rtw/allpairs`. Accessed: 2019-06-04.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[14] David Arthur and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding". In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.

[15] Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. "Dispersion for Data-Driven Algorithm Design, Online Learning, and Private Optimization". In: *CoRR* abs/1711.03091 (2017). arXiv: `1711.03091`. URL: `http://arxiv.org/abs/1711.03091`.

[16] Maria-Florina Balcan et al. "Learning-Theoretic Foundations of Algorithm Configuration for Combinatorial Partitioning Problems". In: *CoRR* abs/1611.04535 (2016). arXiv: `1611.04535`. URL: `http://arxiv.org/abs/1611.04535`.

[17] T. Mitchell et al. "Never-ending Learning". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 2302–2310. ISBN: 0-262-51129-0. URL: `http://dl.acm.org/citation.cfm?id=2886521.2886641`.

[18] T. Mitchell et al. "Never-ending Learning". In: *Commun. ACM* 61.5 (Apr. 2018), pp. 103–115. ISSN: 0001-0782. DOI: `10.1145/3191513`. URL: `http://doi.acm.org/10.1145/3191513`.

[19] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: `http://yann.lecun.com/exdb/mnist/`.

[20] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009.

[21] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338. ISSN: 0036-8075. DOI: `10.1126/science.aab3050`. eprint: `http://science.sciencemag.org/content/350/6266/1332.full.pdf`. URL: `http://science.sciencemag.org/content/350/6266/1332`.

[22] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[23] James Munkres. "Algorithms for the assignment and transportation problems". In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38.

[24] Dan DeBlasio and John Kececioglu. "Parameter advising for multiple sequence alignment". In: *BMC bioinformatics*. Vol. 16. 2. BioMed Central. 2015, A3.

[25] Mengye Ren et al. "Meta-learning for semi-supervised few-shot classification". In: *arXiv preprint arXiv:1803.00676* (2018).

[26] Oriol Vinyals et al. "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 3630–3638. URL: `http://papers.nips.cc/paper/6385-matching-networks-for-one-shot-learning.pdf`.

[27] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 1126–1135. URL: http://proceedings.mlr.press/v70/finn17a.html.

[28] Eleni Triantafillou et al. "Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples". In: *CoRR* abs/1903.03096 (2019). arXiv: 1903.03096. URL: http://arxiv.org/abs/1903.03096.

[29] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

A. The Hungarian Method

Our goal is to find the best possible matching between two clusterings $C_1^i, ..., C_k^i$ and $C_1^j, ..., C_k^j$. In order to do so, we calculate the cost of matching each possible pair of clusters within the two clusterings.

To find the optimal matching in a brute force way, we have to look at each possible matching. Say we want to match each i to one j. For $i = 1$ we can pick from 5 different values of j, for $i = 2$ there are 4 potential values of j. This will overall result in $k! = 5! = 120$ different combinations, thus the complexity of the brute force approach is $O(k!)$. A more efficient algorithm (especially for higher values of k) was introduced by Kuhn and Munkres [22][23]. It consists of three major steps. In the first one, we subtract the row minima from each row. This step is performed in table A.1.

j\i	1	2	3	4	5	
1	5	0	15	35	25	(-15)
2	70	0	5	10	20	(-10)
3	0	10	30	60	40	(-20)
4	20	40	30	10	0	(-10)
5	0	10	20	30	5	(-20)

Table A.1.: Hungarian method step 1: Subtract the row minima from each row.

After subtracting the row minima, we now also subtract the column minima from each column as shown in table A.2.

j\i	1	2	3	4	5
1	5	0	10	25	25
2	70	0	0	0	20
3	0	10	25	50	40
4	20	40	25	0	0
5	0	10	15	20	5
	-	-	(-5)	(-10)	-

Table A.2.: Hungarian method step 2: Subtract the column minima from each column.

Now we try to find the optimal matching. To do so, we cover all zeros with lines and count the minumum needed lines to do so. Table A.3 shows that we need four lines.

After covering the zeros and counting the lines, we found the optimal matching in case the number of lines equals the number of rows (or columns) in the matrix. As we need four lines and the matrix has five rows in this example, we have to add more zeros. To do

j\i	1	2	3	4	5
1	5	0	10	25	25
2	70	0	0	0	20
3	0	10	25	50	40
4	20	40	25	0	0
5	0	10	15	20	5

Table A.3.: Hungarian method step 3: Cover all zeros with as few lines as possible.

that, we subtract the minimum value of the matrix (which is 5 here) from all uncovered values that are not zero and add it to all values that are not zero and covered twice. Now we can again check the needed lines as in table A.4.

j\i	1	2	3	4	5
1	5	0	5	20	20
2	75	0	0	0	20
3	0	10	20	45	35
4	25	45	25	0	0
5	0	10	10	15	0

j\i	1	2	3	4	5
1	5	0	5	20	20
2	75	0	0	0	20
3	0	10	20	45	35
4	25	45	25	0	0
5	0	10	10	15	0

Table A.4.: Hungarian method additional step: Create more zeroes until the number of minimal needed lines to cover all zeros matches the number of rows.

This will then result in the assignment seen in table A.5. Applying the matching to the input matrix then gives the optimal cost by summing the optimal values. For this example the optimal cost is then 95.

j\i	1	2	3	4	5
1	5	0	5	20	20
2	75	0	0	0	20
3	0	10	20	45	35
4	25	45	25	0	0
5	0	10	10	15	0

j\i	1	2	3	4	5
1	20	15	30	50	40
2	80	10	15	20	30
3	20	30	50	80	60
4	30	50	40	20	10
5	20	30	40	50	25

Table A.5.: Result of the hungarian method: The optimal matching between two clusterings.

B. Convolutional Neural Network Architecture for Feature Extraction

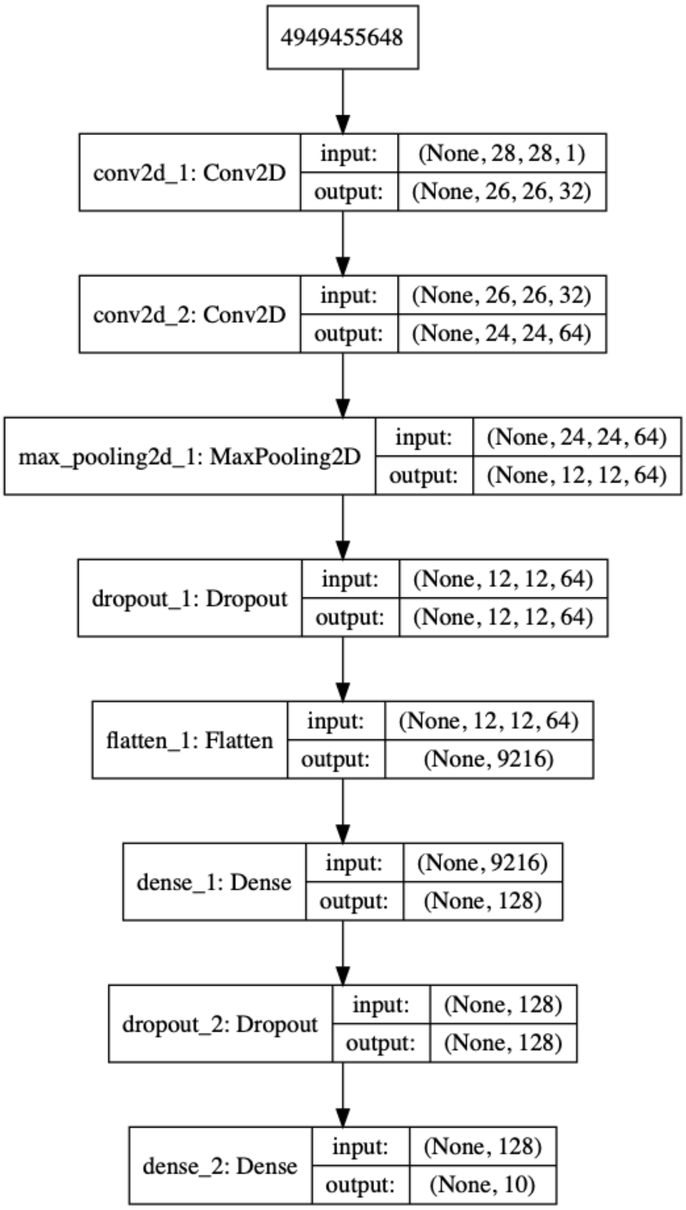
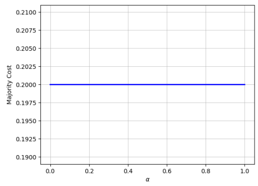
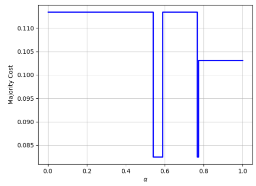
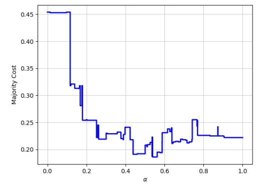


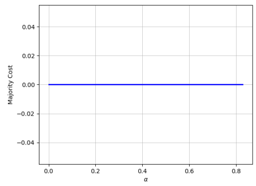
Figure B.1.: We use a small Convolutional Neural Network (CNN) architecture to create meaningful features for the MNIST and the Omniglot data.

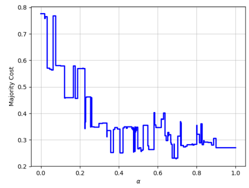
C. NELL Results

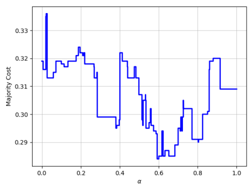

(a) Animal

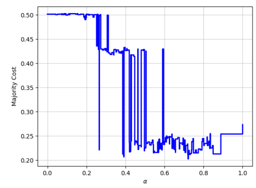

(b) Arthropod

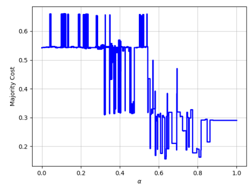

(c) Attraction

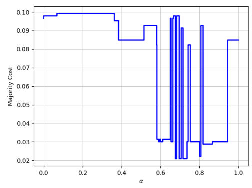

(d) Beverage

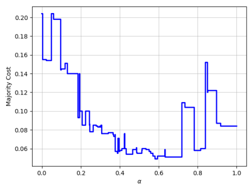

(e) Bodypart

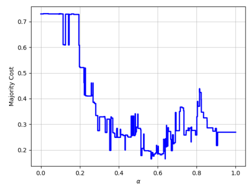

(f) Building

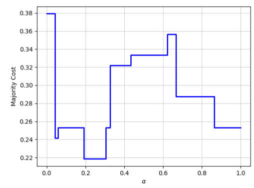

(g) Company

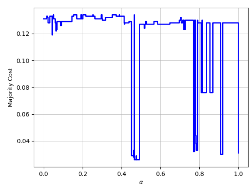

(h) Creative Work

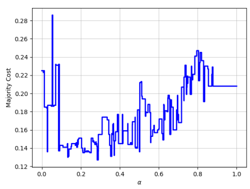

(i) Date

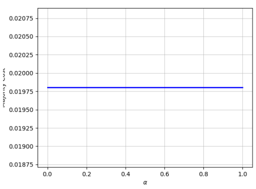

(j) Event

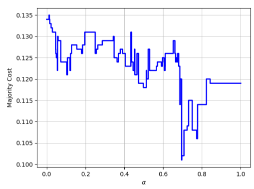

(k) Food

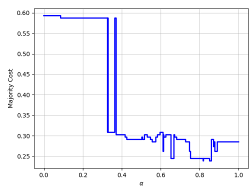

(l) Fruit

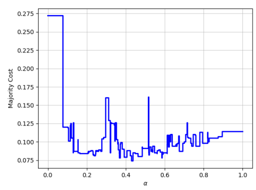

(m) Game

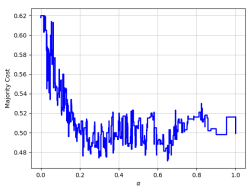

(n) Household Item

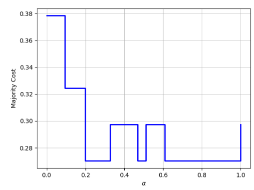

(o) Invertebrate

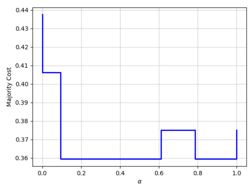

(p) Location


(q) Mediacompany

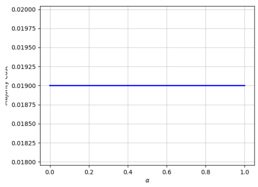

(r) Organization


(s) Person

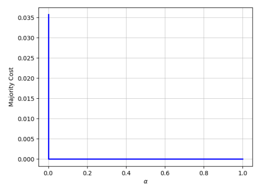

(t) Person by Location



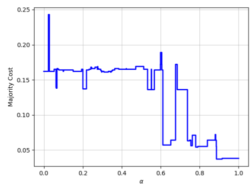
(u) Person NA



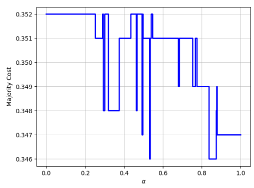
(v) Physiological Condition



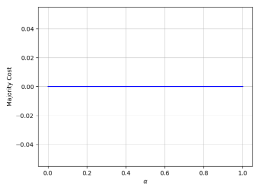
(w) Politics Group



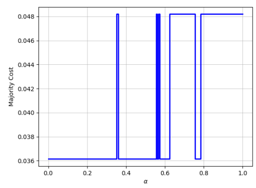
(x) Product



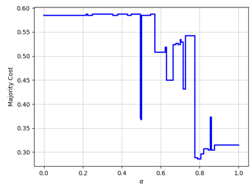
(y) Publication



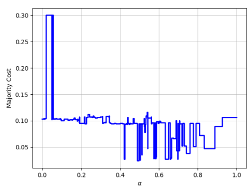
(z) School



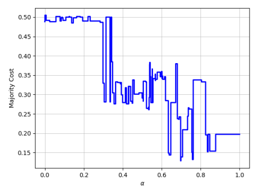
(aa) Software



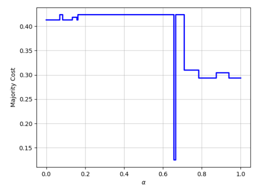
(ab) Sports Event



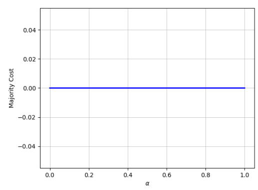
(ac) Traditional Game



(ad) Vertebrate

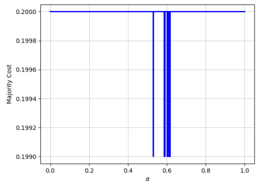


(ae) Visualizable Attribute

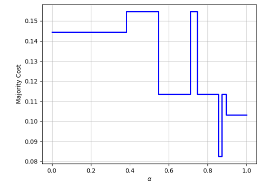


(af) Website

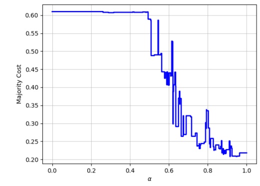
Figure C.1.: Interpolating between average and complete linkage for the NELL data.



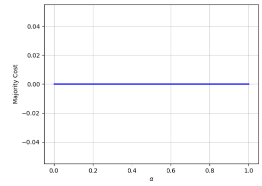
(a) Animal



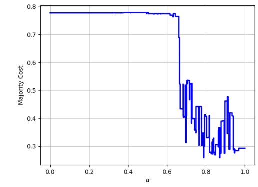
(b) Arthropod



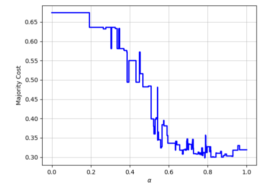
(c) Attraction



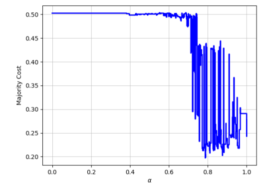
(d) Beverage



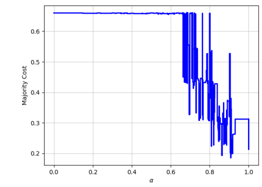
(e) Bodypart



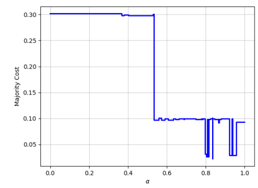
(f) Building



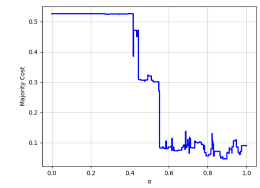
(g) Company



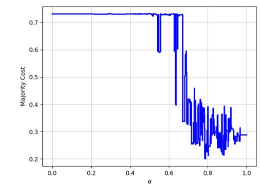
(h) Creative Work



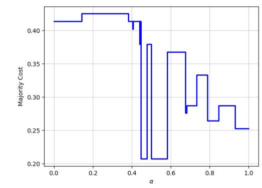
(i) Date



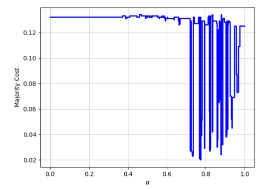
(j) Event



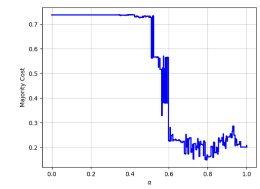
(k) Food



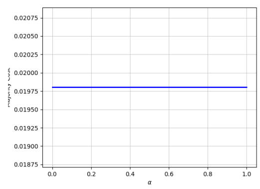
(l) Fruit



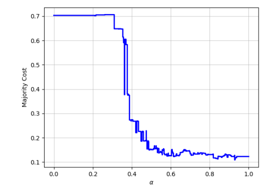
(m) Game



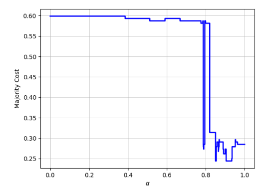
(n) Household Item



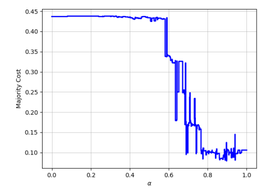
(o) Invertebrate



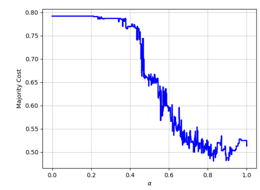
(p) Location



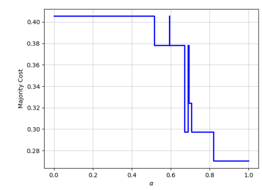
(q) Mediacompany



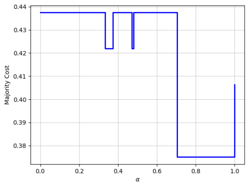
(r) Organization



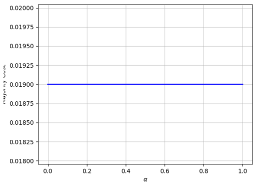
(s) Person



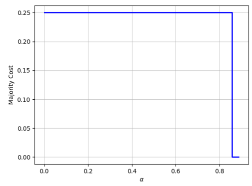
(t) Person by Location



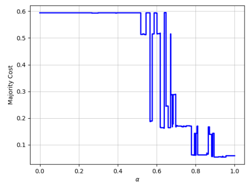
(u) Person NA



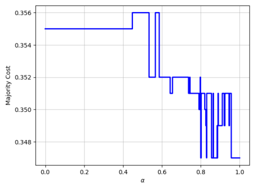
(v) Physiological Condition



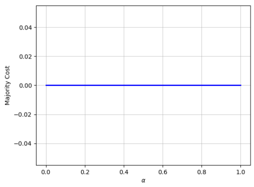
(w) Politics Group



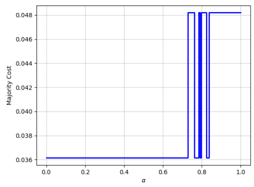
(x) Product



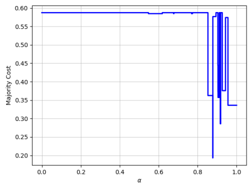
(y) Publication



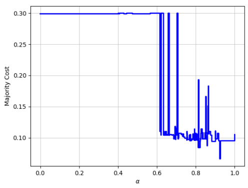
(z) School



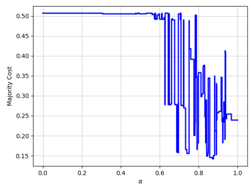
(aa) Software



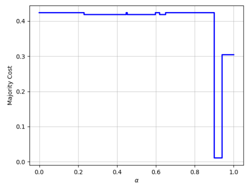
(ab) Sports Event



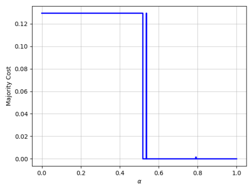
(ac) Traditional Game



(ad) Vertebrate



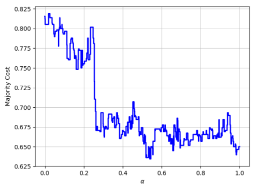
(ae) Visualizable Attribute



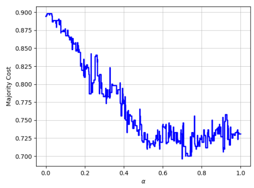
(af) Website

Figure C.2.: Interpolating between single and complete linkage for the NELL data.

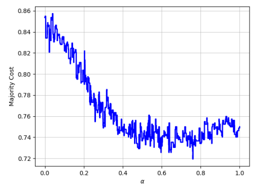
D. Omniglot Intra-Alphabet Results



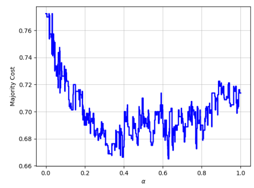
(a) Magi



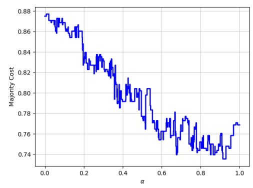
(b) Anglo-Saxon



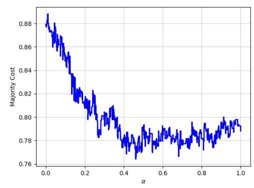
(c) Arcadian



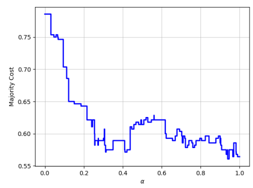
(d) Armenian



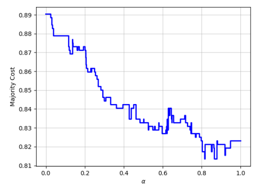
(e) Asomtavruli



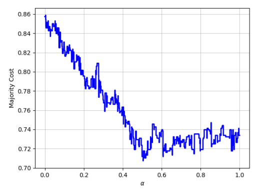
(f) Balinese



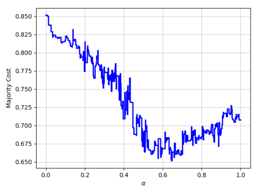
(g) Bengali



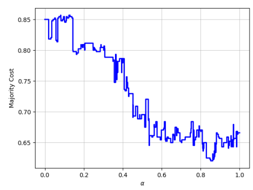
(h) Blackfoot



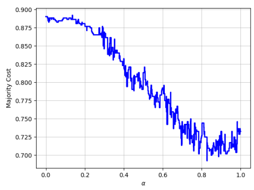
(i) Braille



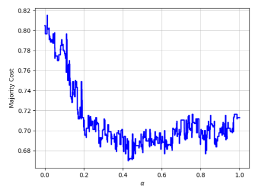
(j) Burmese



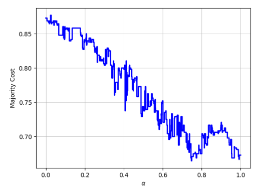
(k) Cyrillic



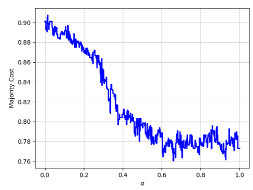
(l) Early_Aramaic



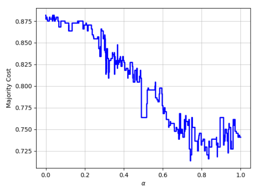
(m) Futurama



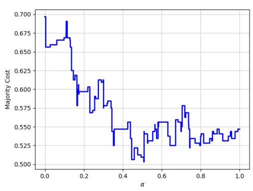
(n) Grantha



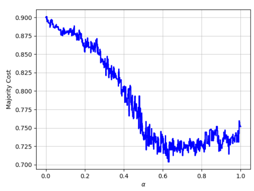
(o) Greek



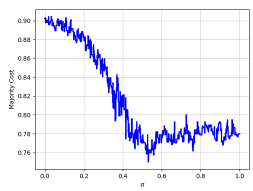
(p) Gujarati



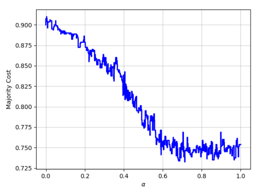
(q) Hebrew



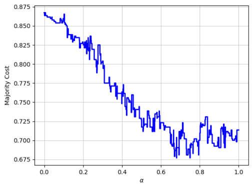
(r) Inuktitut



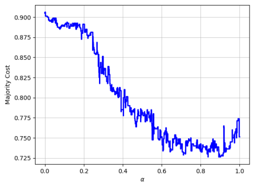
(s) Hiragana



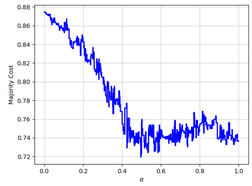
(t) Katakana



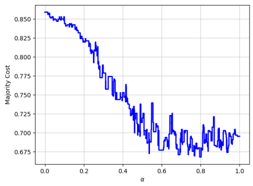
(u) Korean



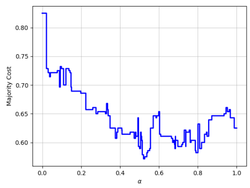
(v) Latin



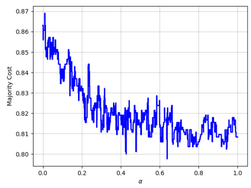
(w) Malay



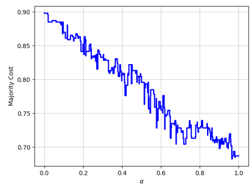
(x) Mkhedruli



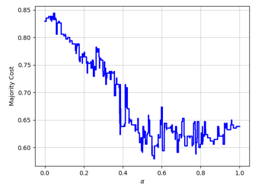
(y) N_Ko



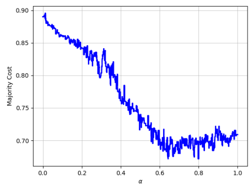
(z) Ojibwe



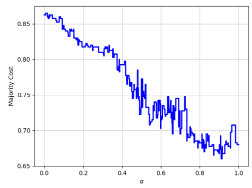
(aa) Sanskrit



(ab) Syriac

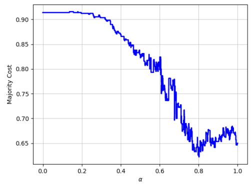


(ac) Tagalog

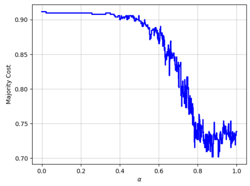


(ad) Tifinagh

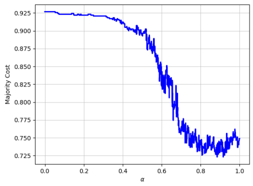
Figure D.1.: Interpolating between average and complete linkage for the omniglot data.



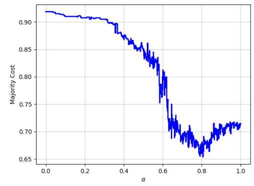
(a) Magi



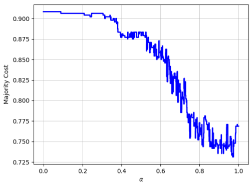
(b) Anglo-Saxon



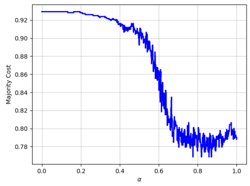
(c) Arcadian



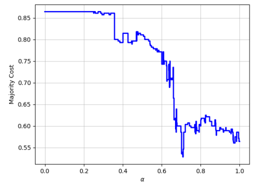
(d) Armenian



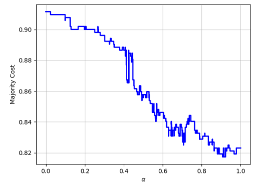
(e) Asomtavruli



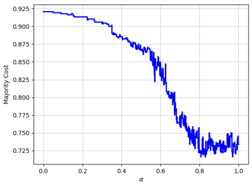
(f) Balinese



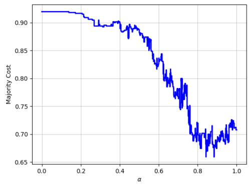
(g) Bengali



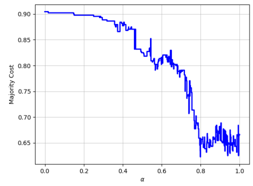
(h) Blackfoot



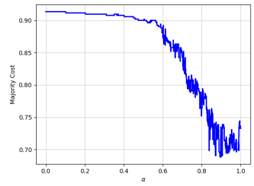
(i) Braille



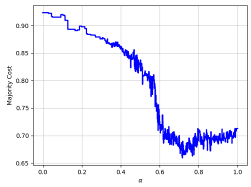
(j) Burmese



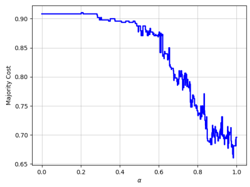
(k) Cyrillic



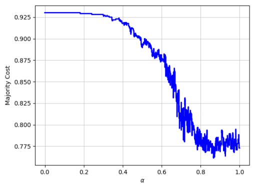
(l) Early_Aramaic



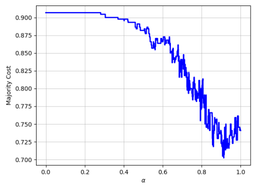
(m) Futurama



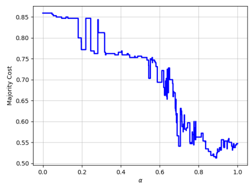
(n) Grantha



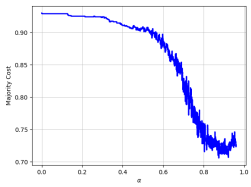
(o) Greek



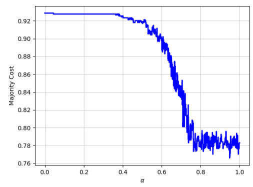
(p) Gujarati



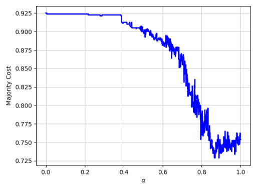
(q) Hebrew



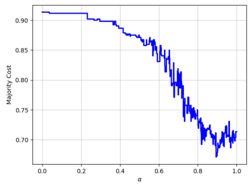
(r) Inuktitut



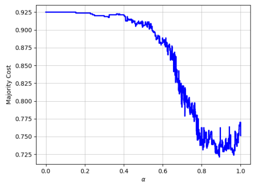
(s) Hiragana



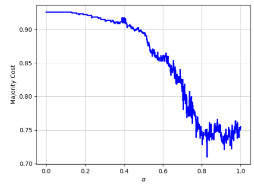
(t) Katakana



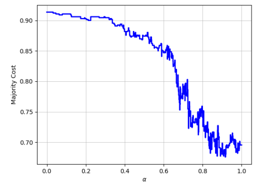
(u) Korean



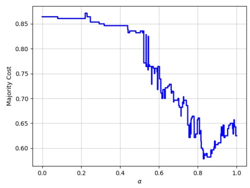
(v) Latin



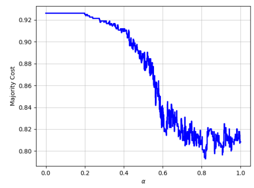
(w) Malay



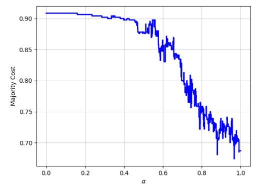
(x) Mkhedruli



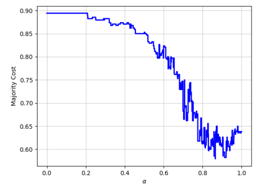
(y) N_Ko



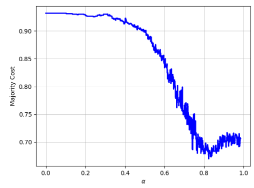
(z) Ojibwe



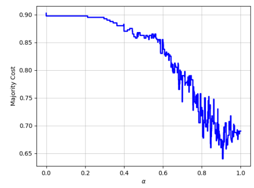
(aa) Sanskrit



(ab) Syriac

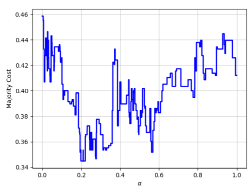


(ac) Tagalog

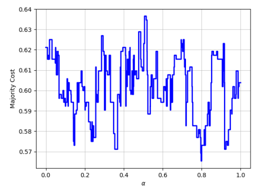


(ad) Tifinagh

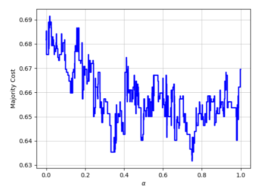
Figure D.2.: Interpolating between single and complete linkage for the omniglot data.



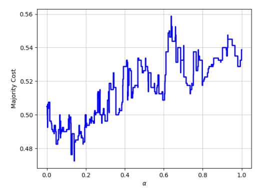
(a) Magi



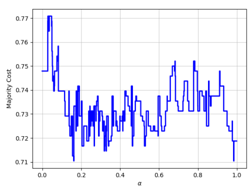
(b) Anglo-Saxon



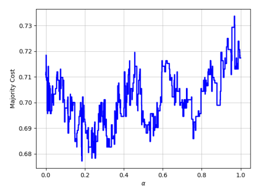
(c) Arcadian



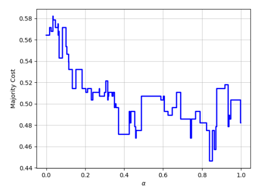
(d) Armenian



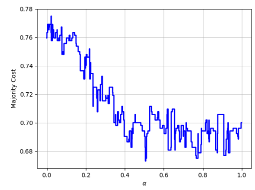
(e) Asomtavruli



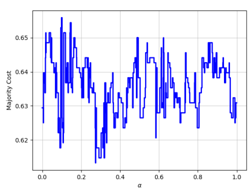
(f) Balinese



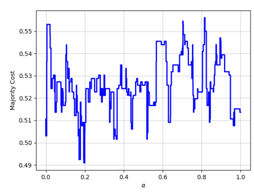
(g) Bengali



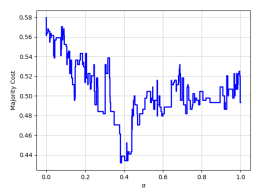
(h) Blackfoot



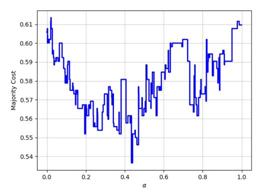
(i) Braille



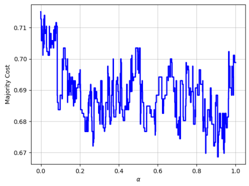
(j) Burmese



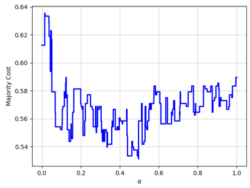
(k) Cyrillic



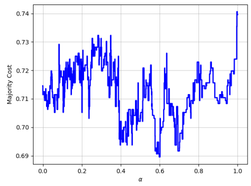
(l) Early_Aramaic



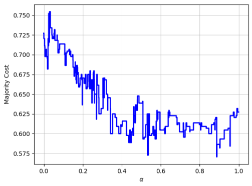
(m) Futurama



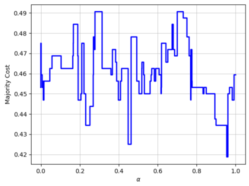
(n) Grantha



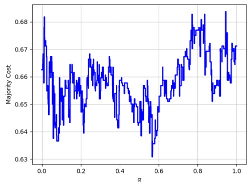
(o) Greek



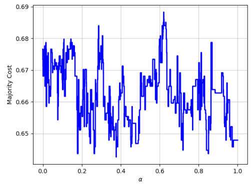
(p) Gujarati



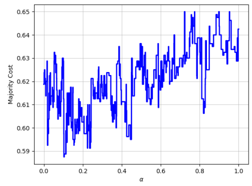
(q) Hebrew



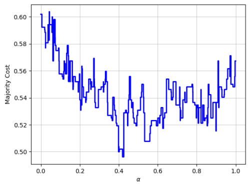
(r) Inuktitut



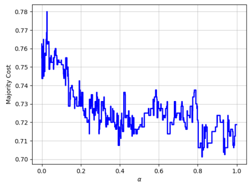
(s) Hiragana



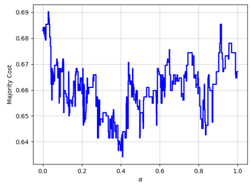
(t) Katakana



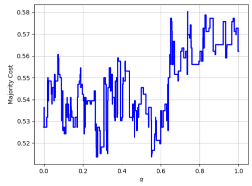
(u) Korean



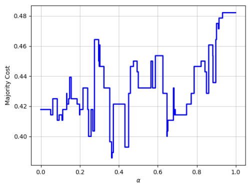
(v) Latin



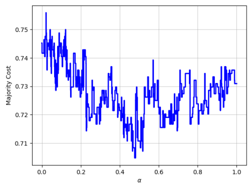
(w) Malay



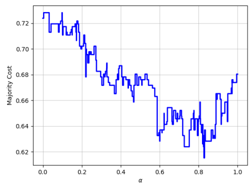
(x) Mkhedruli



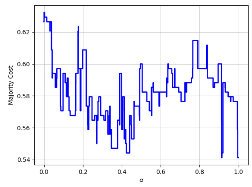
(y) N_Ko



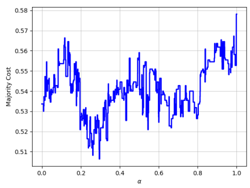
(z) Ojibwe



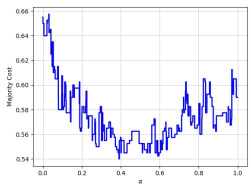
(aa) Sanskrit



(ab) Syriac

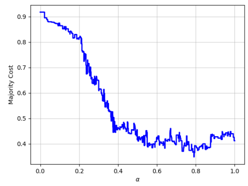


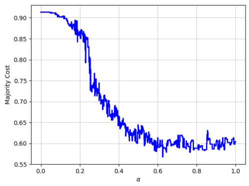
(ac) Tagalog

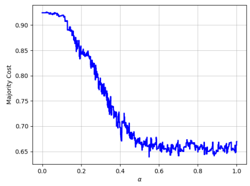


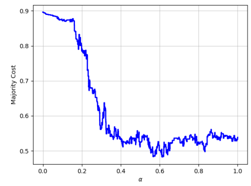
(ad) Tifinagh

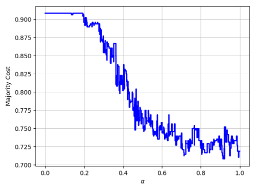
Figure D.3.: Interpolating between average and complete linkage for the omniglot data with CNN features.

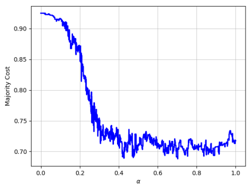

(a) Magi

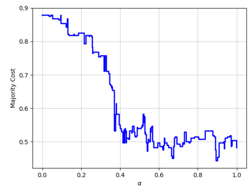

(b) Anglo-Saxon

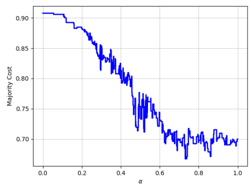

(c) Arcadian

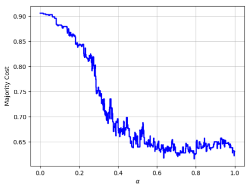

(d) Armenian

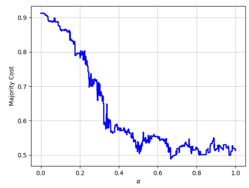

(e) Asomtavruli

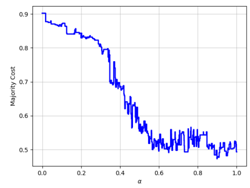

(f) Balinese

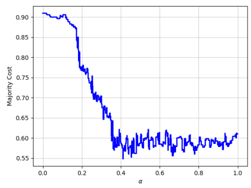

(g) Bengali

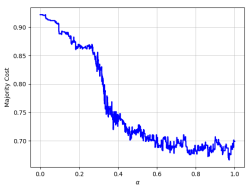

(h) Blackfoot

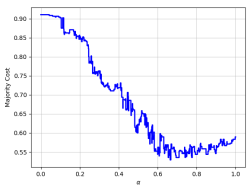

(i) Braille

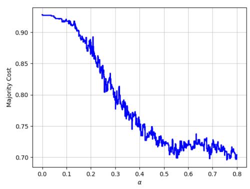

(j) Burmese

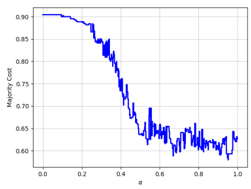

(k) Cyrillic

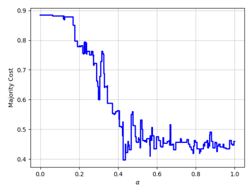

(l) Early_Aramaic

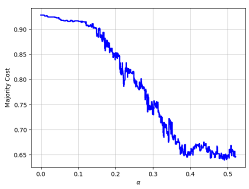

(m) Futurama

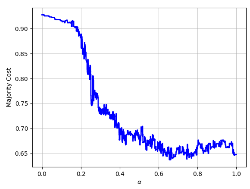

(n) Grantha

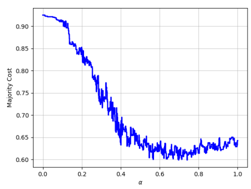

(o) Greek

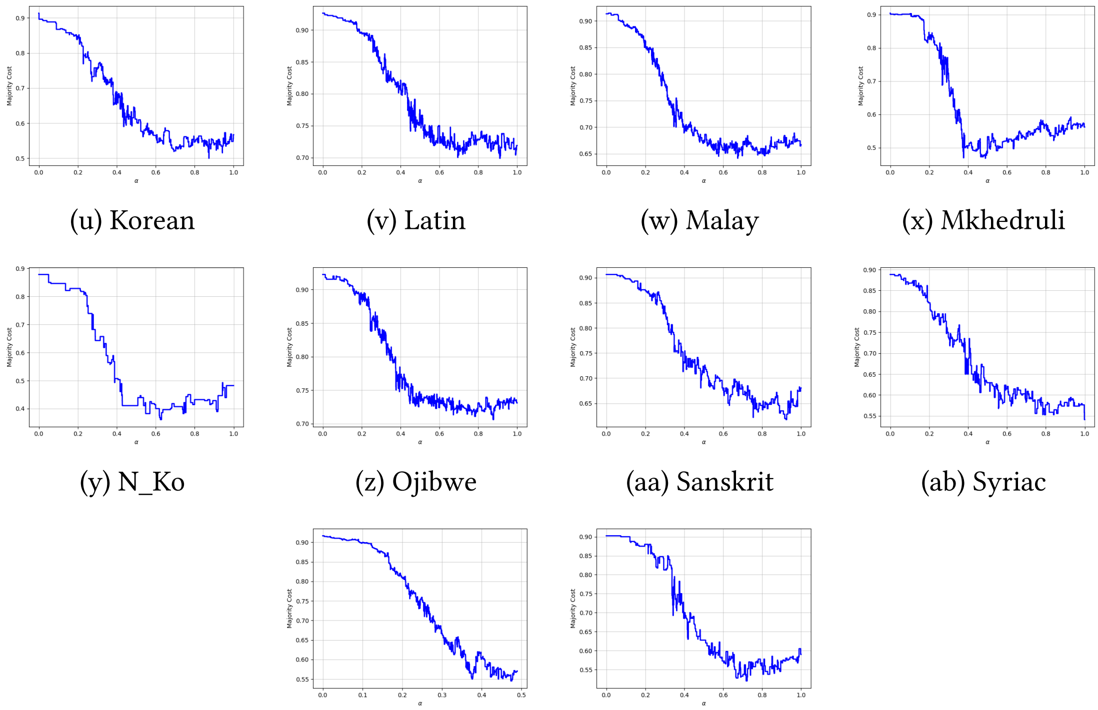

(p) Gujarati


(q) Hebrew


(r) Inuktitut


(s) Hiragana


(t) Katakana



(u) Korean (v) Latin (w) Malay (x) Mkhedruli

(y) N_Ko (z) Ojibwe (aa) Sanskrit (ab) Syriac

(ac) Tagalog (ad) Tifinagh

Figure D.4.: Interpolating between single and complete linkage for the omniglot data with CNN features.