CENG499 Assignment-2 Report

Batuhan Karaca 2310191

November 2022

Contents

List of Tables

List of Figures

1 Introduction

This report aims to inform about the methodology and the results of an experiment, for each part. For each part in first subsection, the preparations, methodology, and the details of the experiment is introduced. In the subsection after that, the results are discussed. Note that some parts have additional subsections (Worst Case Time Complexity Analysis for part 2). Further note that all confidence scores have level 95% for all results.

2 Part 1

2.1 Experiment

The experiment tests the K-Nearest-Neighbors (KNN) classification method on a dataset. KNN algorithm is applied on 10 folds of data 5 times. Each time, the data is shuffled. Scikit-learn's `RepeatedStratifiedKFold` is used for this purpose [2]. As written in its documentation page, `RepeatedStratifiedKFold` *Repeats Stratified K-Fold n times with different randomization in each repetition*, as the parameters K and number of repetitions/times are the parameters `n_splits` and `n_repeats` in this function.

The given values for `n_splits` and `n_repeats` are 10 and 5 for our task respectively. In total `RepeatedStratifiedKFold` is an iterator that yields `get_n_splits` times, which is equal to `n_splits*n_repeats`, 50. After taking the average accuracy with a confidence interval, the results are obtained for each configuration.

The experiment consists of 12 configurations. The tested parameters are the distance function, the auxillary parameter of the distance function if there is any (None otherwise), and the K value. Note that the Minkowski Distance with auxillary parameter p is actually the Lp-norm of the distance vector. In this regard, for p 1 and 2, the Minkowski distance becomes the Manhattan and Euclidean distances respectively. If you see the definition of the Cosine distance function in `Distance.py`, it is actually the inverse of the Cosine similarity function. The expression

$$\frac{\overrightarrow{x} \cdot \overrightarrow{y}}{||\overrightarrow{x}|| * ||\overrightarrow{y}||}$$

Is always in the closed range $[-1, 1]$. Regarding that the dot product gets larger when the vectors are more similar, this is indeed a similarity metric. The model may perform poorly using the standard KNN. To alleviate this, negated version of this expression is preferred. The values are still in the same range, but inverted. In this regard, the points that are further away will give larger values. Mahalanobis distance is typically Euclidean distance with strings attached. It is multivariate standardization of a distribution (i.e. $(x-\mu)/\sigma$ where μ is the mean and σ is the standard deviation). Similar to the standard normal distribution, it gives distances in terms of the standard deviation, providing invariance to different units and scales.

2.2 Results

The results are shown in Table 1. The configurations show somewhat similar results, at about 90% except the Mahalanobis distance. Mahalanobis distance gives better results for correlated data. The data may not be correlated enough that it performed more poorly. Using Cosine similarity instead of Cosine distance performs much worse, at 0%. I did not include that. Interestingly for $K = 5$, the results are slightly better, except for Cosine distance in which the results climb with increasing K. We may hypothesize that for variations of Euclidean distance (i.e. Euclidean, Manhattan, Mahalanobis and similar) we have a local maximum at $K = 5$ and then test on this hypothesis. According to the results, the algorithm attains its best value with Euclidean distance (Minkowski distance with $p = 2$) when $K = 5$. Nevertheless, there is not much difference when one uses the Manhattan distance.

Distance function	Auxillary parameters	K	Test accuracy
$calculateCosineDistance$	$None$	1	$91.06666666666666 \pm 2.236194990504083$
$calculateCosineDistance$	$None$	5	$93.86666666666665 \pm 1.5166342150374363$
$calculateCosineDistance$	$None$	9	$94.4 \pm 1.6676261344931136$
$calculateMinkowskiDistance$	1	1	$92.4 \pm 1.694036767277761$
$calculateMinkowskiDistance$	1	5	$94.4 \pm 1.708088740864088$
$calculateMinkowskiDistance$	1	9	$94.0 \pm 1.819976068218725$
$calculateMinkowskiDistance$	2	1	$93.06666666666666 \pm 1.429471834085738$
$calculateMinkowskiDistance$	2	5	$94.4 \pm 1.8611630867700861$
$calculateMinkowskiDistance$	2	9	$94.26666666666668 \pm 1.478786481469781$
$calculateMahalanobisDistance$	$None$	1	$86.93333333333332 \pm 2.0893595914112586$
$calculateMahalanobisDistance$	$None$	5	$89.19999999999999 \pm 2.2749513225561553$
$calculateMahalanobisDistance$	$None$	9	$87.46666666666668 \pm 1.9778634847846408$

Table 1: Results of the experiment for part 1

3 Part 2

3.1 Experiment

KMeans clustering algorithm on two datasets, namely `dataset1` and `dataset2` are tested. The parameters are the K value and the initialization method. Please note that instead of a separate K-Means++ class in `KMeansPlusPlus.py`, an initializer parameter in `KMeans` class is used. In the experiment code file for KMeans++, the default `KMeans` class is used with KMeans++ initializer. Nevertheless, `KMeansPlusPlus` class is still implemented by inheriting from `KMeans` for completeness. The default argument for `initializer` is `KMeans.KMeansMinusMinus` static function, which is actually a purely random initialization method. K cluster centers are chosen uniformly from the dataset. On the other hand `KMeans.KMeansPlusPlus`, the initialization method used in the KMeans++ algorithm tries to choose equidistant points, with randomness attached to handle outliers. The probability of a dataset point to be chosen as a new cluster center is proportional to the distance to the closest already chosen cluster center (Actually quadratically proportional , $\frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2}$ as given in the assignment text). The first data point is chosen uniformly similar to `KMeans.KMeansMinusMinus`.

The experiment is as follows. For a K, the algorithm is applied to the dataset a number of times (for this experiment 10), and minimum of the values is taken. This is repeated for the same K a number of times (again 10 is preferred). A confidence score from these 10 values is obtained. This score is shown in the results for each K. The K values lie in the closed range of integers from 2 to 10.

3.2 Results

For each time that the algorithm performs, a loss value is calculated. The loss (objective) function is as the following

$$\sum_{k \in \{1,..,K\}} \sum_{x \in C_k} D(x, m_k)$$

Each cluster C_k has its own loss value as the sum of the distance of all its points to its center m_k. These loss values for all K clusters are summed to give the loss value.

The results are given in tables 2, 4, 3, 5, and figures 1, 2, 3, 4, 5, 6. For each dataset and initializer combination in 2 datasets and 2 initializer methods a table and a plot of K value versus the loss function is shown (4 in total). The plot and the table show the same results for the same combination. For convenience, There are 2 additional plots that merge both results of different initializers for each dataset.

In order to find the optimal K value, one can use the elbow-method. After some K, there is not much decline in the loss value. This is interpreted such that after this K, the model is overfitting to the data, hence no need to further increase number of clusters. This K is considered as the best/optimal value. Using elbow-method on the plots is much simpler as the kink in the objective may be visually observable. However, this is not always the case. As seen in the plots of the default KMeans, the loss have a smooth curve. Deducing from the fact that the optimal K should be same for the same dataset despite of the different parameters, one may use KMeans++ with elbow-method, as KMeans++ gives much sharper/pointier curves in the plots. According to the plots with the initializer KMeans++, the optimal K values should be 5 and 4 for `dataset1` and `dataset2` respectively. Alternatively, one could use the silhouette score (as will be used in part 3) to assess the optimal value.

K	Loss
2	$752.6571502685547 \pm 3.8729569283483283$
3	$494.8007335662842 \pm 17.42579226379845$
4	$343.48138904571533 \pm 17.264825024381583$
5	$254.01008615493774 \pm 40.72607189150742$
6	$207.48568210601806 \pm 26.838410160202855$
7	$175.3849290370941 \pm 5.218621541994606$
8	$167.97487647533416 \pm 0.8357853625239936$
9	$161.40006533265114 \pm 1.2852171105101144$
10	$156.02911278009415 \pm 0.9185488484820983$

Table 2: Results of the experiment for part 2 with `dataset1`

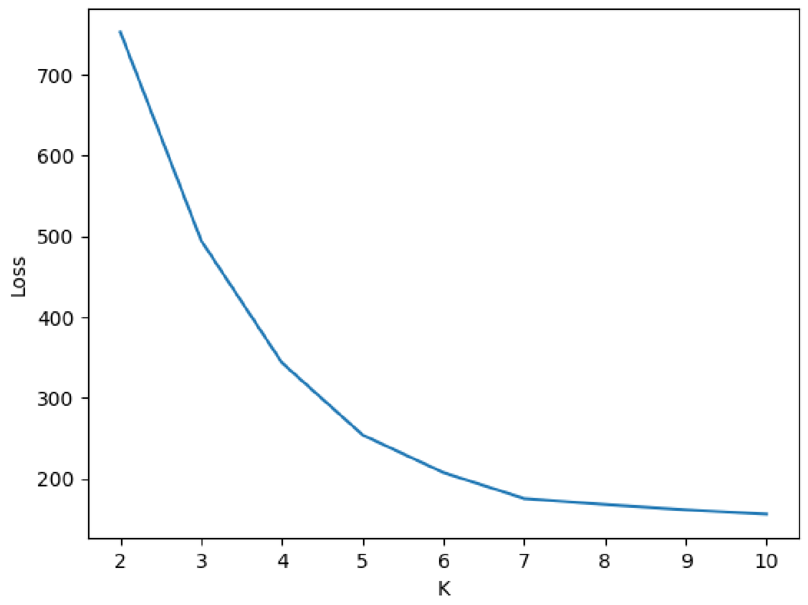

Figure 1: Plot of the results of the experiment for part 2 with `dataset1`

K	Loss
2	1553.9598999023438 ± 0.0
3	938.0059967041016 ± 0.0
4	761.1647804260253 ± 64.98907492811628
5	643.1765909194946 ± 1.3539299777485025
6	632.5855583190918 ± 1.7635761017625056
7	620.7680004119873 ± 1.6291289581585418
8	613.3124961853027 ± 1.4070534090266786
9	604.7733342647552 ± 0.906044576993102
10	594.6105914115906 ± 2.147358571203209

Table 3: Results of the experiment for part 2 with `dataset2`

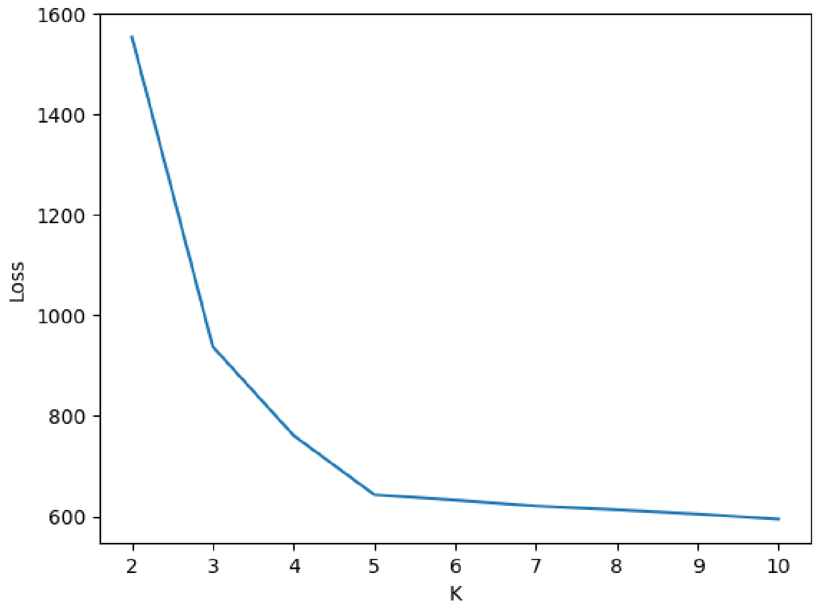

Figure 2: Plot of the results of the experiment for part 2 with `dataset2`

K	Loss
2	753.2413299560546 ± 3.4313868928238263
3	472.96296043395995 ± 8.031656145395967
4	314.4487903594971 ± 0.8290783030066796
5	185.3420608520508 ± 0.01433746096867929
6	177.86972923278807 ± 0.1913570347834377
7	171.40753622055053 ± 0.5932453866373316
8	164.99596018791198 ± 0.5752033283059629
9	158.8628695011139 ± 0.5119522944056305
10	153.77027497291564 ± 0.6793880063043388

Table 4: Results of the experiment for part 2 with `dataset1`, KMeans++

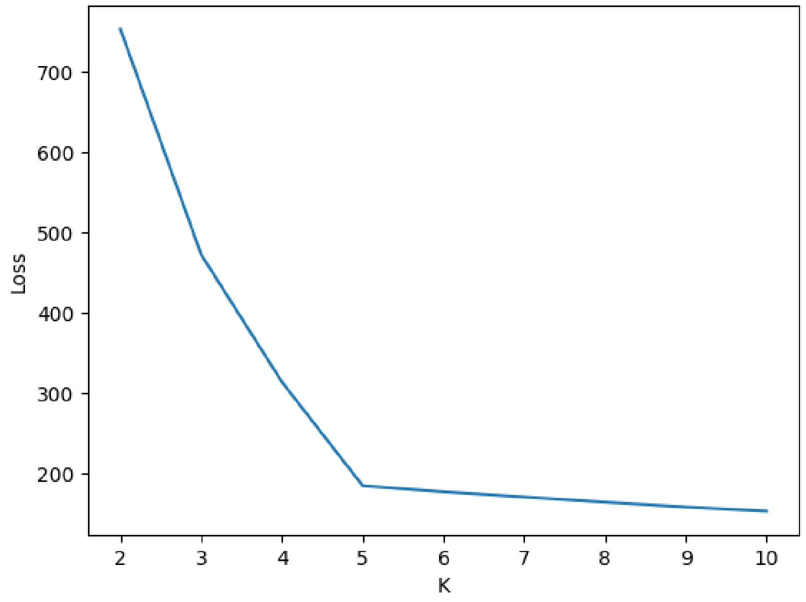

Figure 3: Plot of the results of the experiment for part 2 with `dataset1`, KMeans++

K	Loss
2	1553.9598999023438 ± 0.0
3	938.0059967041016 ± 0.0
4	653.5376892089844 ± 0.0
5	641.0321422576905 ± 0.37422548868254263
6	629.7291969299316 ± 0.5376512861967266
7	619.1261020660401 ± 1.245511746796119
8	610.5372814178467 ± 1.0695075111472228
9	601.0770818710328 ± 1.8076922999818628
10	593.2355173826218 ± 1.2487501379822823

Table 5: Results of the experiment for part 2 with `dataset2`, KMeans++

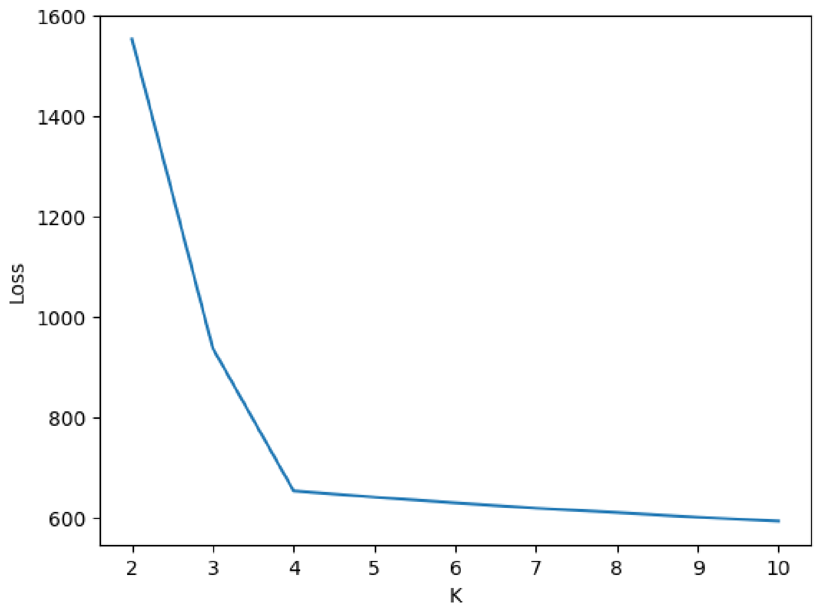

Figure 4: Plot of the results of the experiment for part 2 with `dataset2`, KMeans++

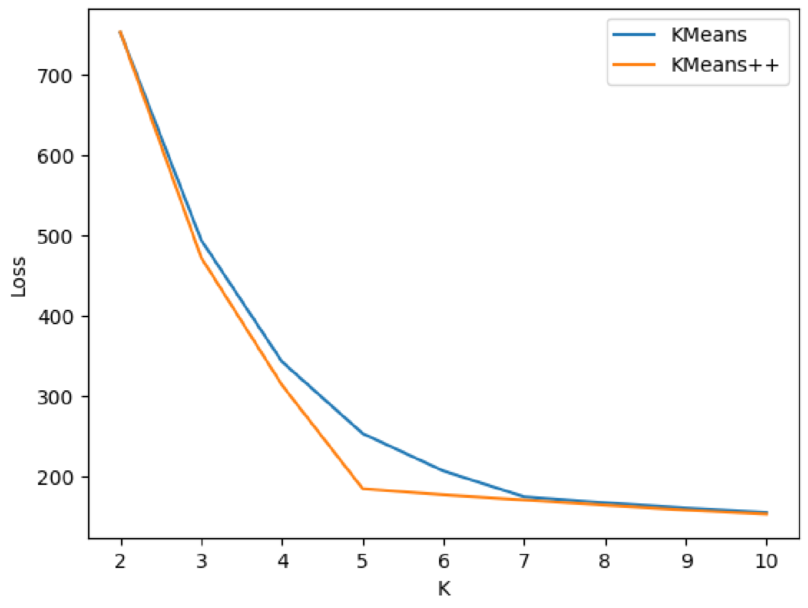

Figure 5: Merged plot of the results of the experiment for part 2 with `dataset1`

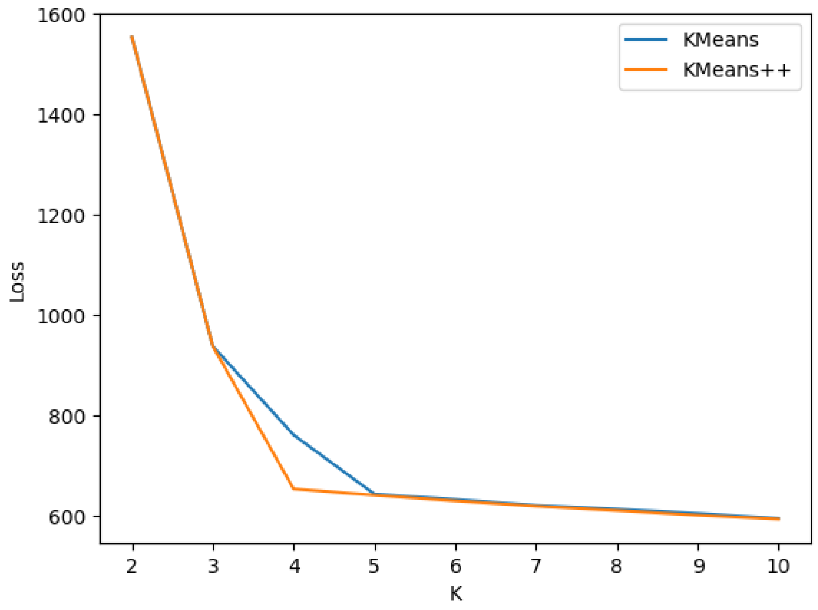

Figure 6: Merged plot of the results of the experiment for part 2 with `dataset2`

3.3 Worst Case Time Complexity Analysis

The KMeans algorithm is as follows

```
Initialize K number of cluster centers using an initializer
  Repeat
  (1) For each data point $x$ in the dataset
        Find the closest cluster center $m_z$ and label the data point as the cluster of it ($z$)
  (2) For each cluster $k$
        Assign cluster center $m_k$ to the mean of the points labeled as the cluster $k$
  until the labels of the data points do not change
```

Assuming a naive implementation, given number of data points (N), number of features (d), the K value and the number of iterations (I) a crude estimation on the complexity could be achieved.

For the expression (1), N data points are iterated. Finding the closest cluster center gives $O(K)$ at the worst case. A naive implementation for a distance function has to iterate all features (d for each cluster center), the complexity for the expression (1) becomes $O(NKd)$.

Similarly for the expression (2), K cluster centers are iterated. For each cluster center, a naive mean algorithm should iterate all the data points in the cluster k. Knowing that this is done for all features also, and N_k is the number of elements in cluster k, the complexity is $O(N_k d)$. Summing up all N_ks under the paranthesis of d, the complexity of (2) is $O(Nd)$.

Summing up the complexities for (1) and (2) gives $O(NKd)$ for an iteration. Knowing the number of iterations I gives $O(NKId)$.

The results with execution time is given in the tables 6, 7, 8, 9. We can see that for the default initializer, the results confirm our deduction for complexity. `dataset1` has 1000 observations whereas `dataset2` has 800. Because K values are close to each other, we can neglect them. The ratio of the values 4.2 and 3.4 is approximately 1.23, almost equal to 1000/800 = 1.25. Execution time for KMeans++ significantly increases with K, as KMeans++ initializer adds overhead. For each new cluster center, we need to iterate for the number of existing clusters N_e. For each cluster center we calculate the distance with complexity $O(d)$. Summing for N_e values from 1 to K-1 gives $\frac{K(K-1)}{2}$, times d giving $O(K^2 d)$, which is the complexity overhead coming from the initializer. Note that these deductions ignore the noise in the hardware.

K	Loss	Time
2	752.6571502685547 ± 3.8729569283483283	4.2771453857421875
3	494.8007335662842 ± 17.42579226379845	4.223248720169067
4	343.48138904571533 ± 17.264825024381583	4.210155487060547
5	254.01008615493774 ± 40.72607189150742	4.238519191741943
6	207.48568210601806 ± 26.838410160202855	4.248480796813965
7	175.3849290370941 ± 5.218621541994606	4.240788698196411
8	167.97487647533416 ± 0.8357853625239936	4.290220499038696
9	161.40006533265114 ± 1.2852171105101144	4.408391237258911
10	156.02911278009415 ± 0.9185488484820983	4.283907175064087

Table 6: Results of the experiment for part 2 with `dataset1`, with execution time

K	Loss	Time
2	1553.9598999023438 ± 0.0	3.418414354324341
3	938.0059967041016 ± 0.0	3.4045450687408447
4	761.1647804260253 ± 64.98907492811628	3.4091646671295166
5	643.1765909194946 ± 1.3539299777485025	3.438220977783203
6	632.5855583190918 ± 1.7635761017625056	3.4473764896392822
7	620.7680004119873 ± 1.6291289581585418	3.4617652893066406
8	613.3124961853027 ± 1.4070534090266786	3.4764537811279297
9	604.7733342647552 ± 0.906044576993102	3.479931592941284
10	594.6105914115906 ± 2.147358571203209	3.4791650772094727

Table 7: Results of the experiment for part 2 with `dataset2`, with execution time

K	Loss	Time
2	753.2413299560546 ± 3.4313868928238263	5.508848667144775
3	472.96296043395995 ± 8.031656145395967	6.8978564739227295
4	314.4487903594971 ± 0.8290783030066796	8.323169946670532
5	185.3420608520508 ± 0.01433746096867929	9.773839473724365
6	177.86972923278807 ± 0.1913570347834377	11.192806959152222
7	171.40753622055053 ± 0.5932453866373316	12.63964319229126
8	164.99596018791198 ± 0.5752033283059629	14.018837451934814
9	158.8628695011139 ± 0.5119522944056305	15.469636917114258
10	153.77027497291564 ± 0.6793880063043388	16.86057186126709

Table 8: Results of the experiment for part 2 with `dataset1`, with KMeans++, with execution time

K	Loss	Time
2	1553.9598999023438 ± 0.0	4.590485095977783
3	938.0059967041016 ± 0.0	5.661029577255249
4	653.5376892089844 ± 0.0	6.782186985015869
5	641.0321422576905 ± 0.37422548868254263	7.892095327377319
6	629.7291969299316 ± 0.5376512861967266	9.09592080116272
7	619.1261020660401 ± 1.245511746796119	10.149774074554443
8	610.5372814178467 ± 1.0695075111472228	11.352937936782837
9	601.0770818710328 ± 1.8076922999818628	12.416603088378906
10	593.2355173826218 ± 1.2487501379822823	14.113362550735474

Table 9: Results of the experiment for part 2 with `dataset2`, with KMeans++, with execution time

4 Part 3

4.1 Experiment

Hierarchical Agglomerative Clustering (HAC) method is tested on a dataset. With the distance metrics Euclidean and Cosine, linkage types single and complete, and the K values 2, 3, 4 and 5, there were 16 parameter configurations in total.

Because HAC algorithm is deterministic compared to KMeans, and no cross validation method is used, confidence values are not considered. Instead silhouette score metric is considered for each configuration. Silhouette scores $s(x)$ close to 1 imply $a(x) \ll b(x)$ for all x, and vice-versa [6]. Smaller $a(x)$ mean that the groups are more tightly formed, and larger $b(x)$ mean that the data points are more distant from the points in other clusters. Therefore, $s(x)$ is telling how well the clustering is made, values closer to 1 being appropriate and -1 being inappropriate. If $a(x)$ equals $b(x)$, $s(x)$ is not telling much. An appropriate clustering may be impossible. Maybe the points are equidistant to each other, as in higher-dimensions (see the Curse of Dimensionality).

When the K value becomes 2, the program plots the dendrogram. The program does not plot dendrogram for each K because it already plots the full tree, details for which is in the next (Results) section.

For the HAC method, `AgglomerativeClustering` function of sci-kit learn is used [3]. `compute_distances` parameter is set to `True` to obtain the distances for the upcoming calculations. Sci-kit learn does not have a method for plotting dendrograms, but `scipy` has `scipy.cluster.hierarchy.dendrogram` function. The code that converts the `AgglomerativeClustering`'s output to the input format necessary for this function is the `plot_dendrogram` function, taken from scikit-learn's website [4].

For the silhouette scores, there is `sklearn.metrics.silhouette_score` [5]. Peer-wise distance matrix is used as argument to this function. However, `AgglomerativeClustering` was not outputting any distance matrix, nor had it as a property of the instance of the class. The distances were stored in the property `AgglomerativeClustering.distances_` (please refer to the documentation) thanks to `compute_distances = True`. I wrote a class `Silhouette` that takes in this property and the tree as `AgglomerativeClustering.children_` to obtain the distance matrix. This class would then give the required score value in its `score` function.

4.2 Results

The results giving the silhouette scores are given in tabular form, in table 10. There are also visual representations in the form of dendrograms (figures 9, 10, 11, 12), for each parameter except the K value. Because the full trees are computed as shown in the dendrograms, only other parameters were considered, hence 4 dendrograms in total (less than 16, the number of configurations). When the reader traverses the tree in top-down fashion (from the most distant to least), they can see the results of increasing K values, from 1 to the number of data points. Although the data is 3-dimensional it is still more difficult to visualize on a page. Dendrograms are useful for high-dimensional spaces.

As may be confirmed by the dendrograms, the single linkage forms more long-stringy clusters, which are more inefficient to traverse.

The results for this experiment imply that the parameters are well enough for the clustering, close to 1. The data is most appropriately grouped for the configuration (*cosine*, *single*, 2), almost 1. The best results are observed for 2 clusters, except the combination (*euclidean*, *complete*). The data could be suitable for 2 long-stringy clusters, such as the picture of two Koi fish in figure 7. Nevertheless, there could be small noise that the exception combination gave close results for 2 and 3 clusters, 0.79 and 0.82 respectively.

Figure 7: Picture of two Koi fish, famous in Taoism [1]

Our expectations were not fully wrong. In order to confirm the expectations, I plotted the data in 3D space (figure 8), lucky that the data distributed easy enough to be visualized on a 2D page. Actually there was two Koi fish like structure with a small noise (notice the outlying points on the left).

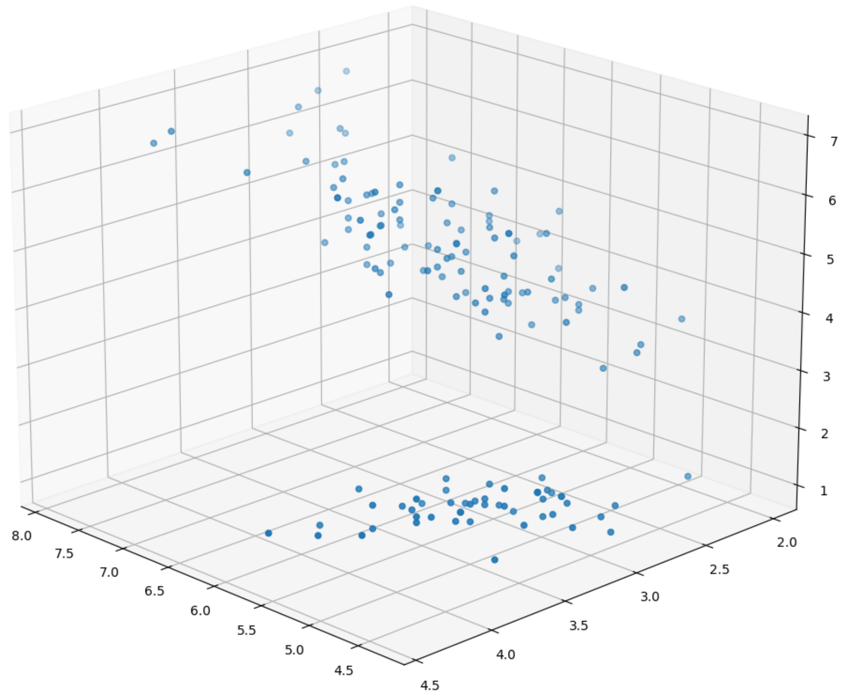

Figure 8: 3D plot of the data

Distance metric	Linkage	Number of clusters	Silhouette Score
euclidean	*single*	2	0.9237563251375881
euclidean	*single*	3	0.7979959816790024
euclidean	*single*	4	0.7825913199798943
euclidean	*single*	5	0.7194784732712853
euclidean	*complete*	2	0.7969111110481422
euclidean	*complete*	3	0.821793480142029
euclidean	*complete*	4	0.7563141880362125
euclidean	*complete*	5	0.7319086719682669
cosine	*single*	2	0.996575458972535
cosine	*single*	3	0.9704087388473427
cosine	*single*	4	0.8145991736857484
cosine	*single*	5	0.7325858173418087
cosine	*complete*	2	0.9462089623068103
cosine	*complete*	3	0.8720896090310216
cosine	*complete*	4	0.8067909381207267
cosine	*complete*	5	0.824524384726412

Table 10: Results of the experiment for part 3

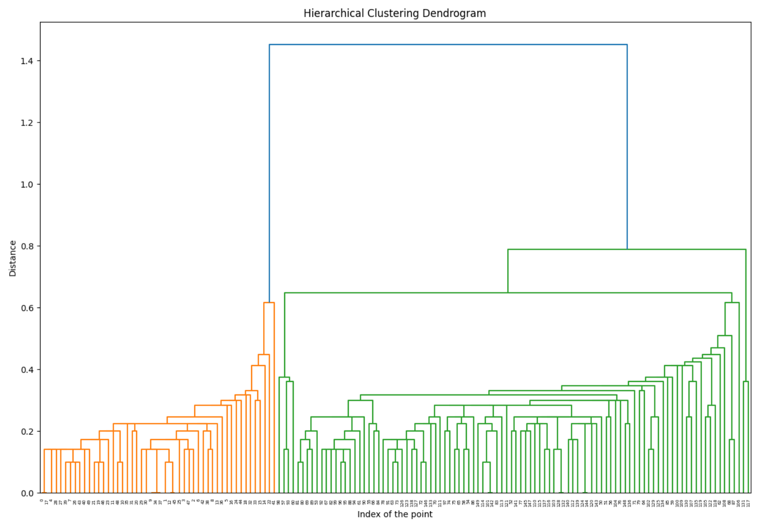

Figure 9: Dendrogram plot of the clusters with Euclidean Distance, Single linkage

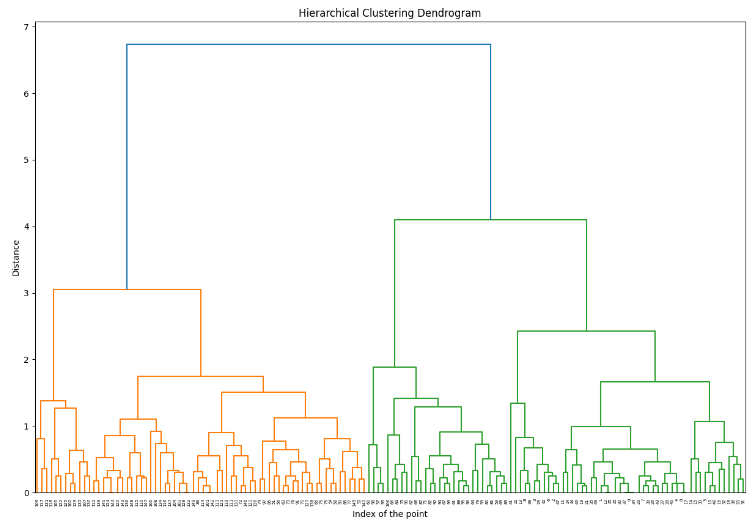

Figure 10: Dendrogram plot of the clusters with Euclidean Distance, Complete linkage

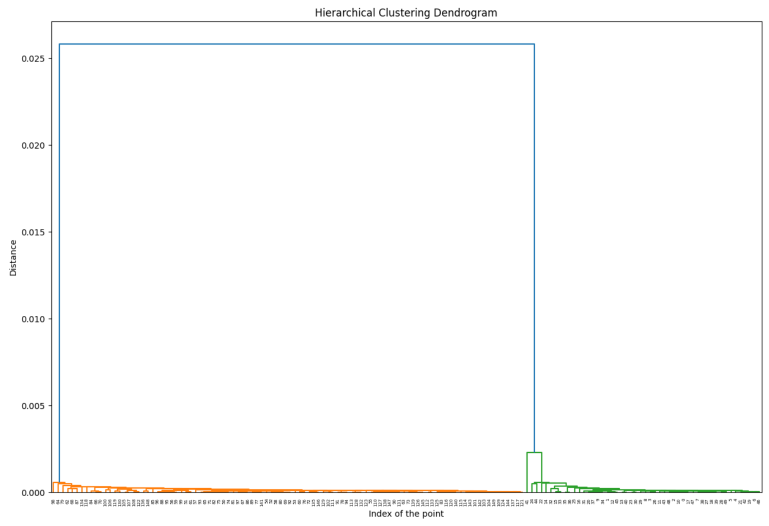

Figure 11: Dendrogram plot of the clusters with Cosine Distance, Single linkage

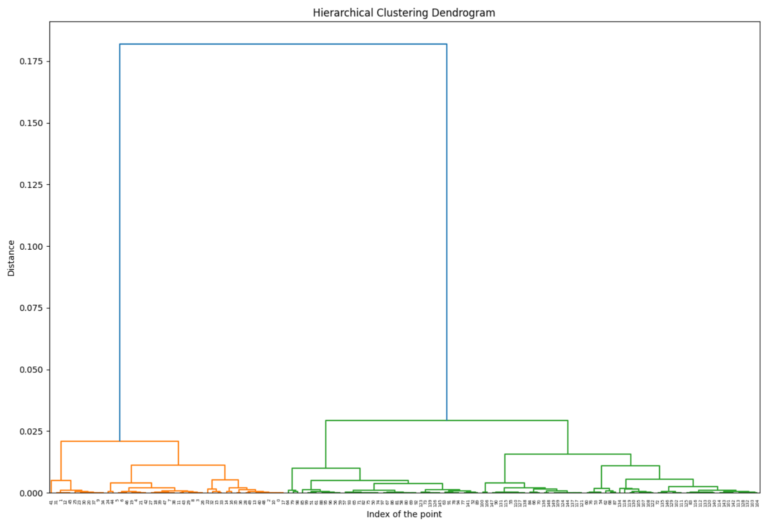

Figure 12: Dendrogram plot of the clusters with Cosine Distance, Complete linkage

5 References

References

[1] Marjansart Paintings. Card of two Koi Fish on black card [Painting]. Folksy. `https://folksy.com/items/7584457-Card-of-two-Koi-Fish-on-black-card-Original`

[2] sklearn.model_selection.RepeatedStratifiedKFold. (n.d.). Scikit-learn. `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html`

[3] sklearn.cluster.AgglomerativeClustering. (n.d.). Scikit-learn. `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html`

[4] plot_agglomerative_dendrogram. (n.d.). Scikit-learn. `https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html`

[5] sklearn.metrics.silhouette_score. (n.d.). Scikit-learn. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html`

[6] Wikipedia contributors. (2022, November 5). Silhouette (clustering). Wikipedia. https://en.wikipedia.org/wiki/Silhouette_(clustering)