COMP527 - Assignment 2 - Data Clustering

Rob Lamprell - 201459448

Due Date: 7th May 2021

Contents

List of Figures

Assignment Task Descriptions

Below are the tasks listed in the assignment.

1. (25 marks) Implement the k-means clustering algorithm to cluster the instances into k clusters.

2. (25 marks) Implement the k-medians clustering algorithm to cluster the instances into k clusters.

3. (10 marks) Run the k-means clustering algorithm you implemented in part (1) to cluster the given instances. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

4. (10 marks) Now re-run the k-means clustering algorithm you implemented in part (1) but normalise each object (vector) to unit ‘2 length before clustering. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

5. (10 marks) Run the k-medians clustering algorithm you implemented in part (2) over the unnormalised objects. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

6. (10 marks) Now re-run the k-medians clustering algorithm you implemented in part (2) but normalise each object (vector) to unit ‘2 length before clustering. Vary the value of k from 1 to 9 and compute the B-CUBED precision, recall, and F-score for each set of clusters. Plot k in the horizontal axis and the B-CUBED precision, recall and F-score in the vertical axis in the same plot.

7. (10 marks) Comparing the different clusterings you obtained in (3)-(6), discuss in which setting you obtained best clustering for this dataset.

B-CUBED Metric Definitions

Throughout the assignment we use the B-CUBED metrics to assess the performance of our clustering models. These take the form of three components, Precision, Recall and F-score.

Precision works by taking each data-point within a given cluster, taking the count of all items in the cluster which share the same class and dividing it by the total number of data-points, within the cluster. Precision can help us identify when the addition of more clusters no longer increases accuracy.

Recall takes an alternative approach to Precision. Instead of comparing the count of each data-point's class to those found within its own cluster, it divides them by the total number of instances of that in-class across all clusters. This has a fairly obvious weakness, if you only have a single (or low count) or clusters, Recall would report extremely high performance.

F-Score bridges the gap between Precision and Recall, taking both into account to compensate for their individual bias(es) [1].

Tasks

This section contains solutions and discussions around the assignment tasks.

Task-1 & Task-2

Please refer to the file **Assignment_2.py** for the implementations of the K-means and K-medians algorithms. Running the file will execute the *main* function its base - providing samples of the graphs and results found below.

Task-3

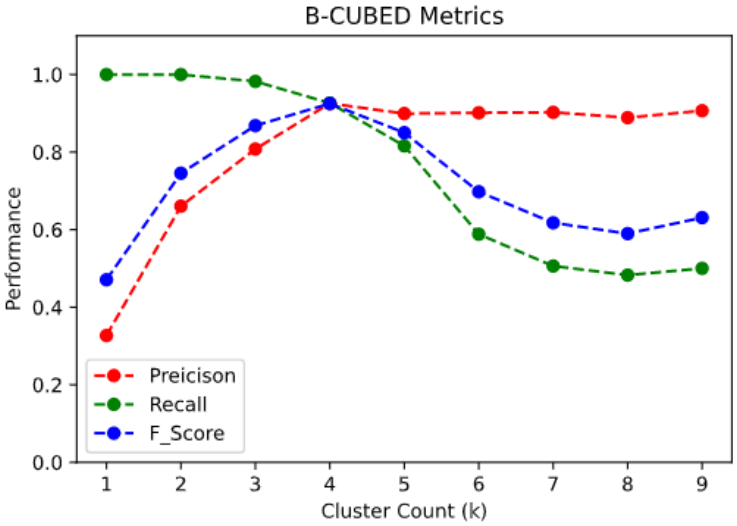

Figure 1: K-Means B-CUBED precision, recall and F-score for k values of 1 to 9 - Numpy Seed 54.

Cluster(k)	Precision	Recall	F-Score
1	.33	1.0	.47
2	.66	1.0	.75
3	.81	.98	.87
4	.93	.93	.93
5	.90	.82	.85
6	.90	.59	.70
7	.90	.51	.62
8	.89	.48	.59
9	.91	.50	.83

Table 1: Precision, Recall and F-Score Results - Numpy Seed 54.

The above graph and table have been produced using a numpy seed of 54 (default). Here we can see, Precision demonstrates poor performance with a single cluster but, rapidly improves when moving to two. The performance increase continues until the number of clusters (k) is equal to four, at which point Precision becomes flat with only slight variances. This makes sense, as our data set contains four (hopefully) distinct classes of animals, countries, fruits and veggies.

Recall demonstrates 100% accuracy when the number of clusters is equal to one or two. Falling only slightly when the cluster counts are equal to three and four. This then descends far more quickly until the number of clusters is equal to seven - when it flattens out.

F-Score here help us identify the best clustering (k) selection at four. Where can see the three-way-intersection of Precision, Recall and F-Score.

Figure-1 is not representative of every possible outcome of the K-Means algorithm. The initial placement of the centroids is random and has therefore has the potentially to have a significant impact on final model - the centroids may fall into local optimals rather than the global optimals. The array of graphs displayed in Figure-2 demonstrate this randomness - the final sub-figure finds strong performance with five clusters.

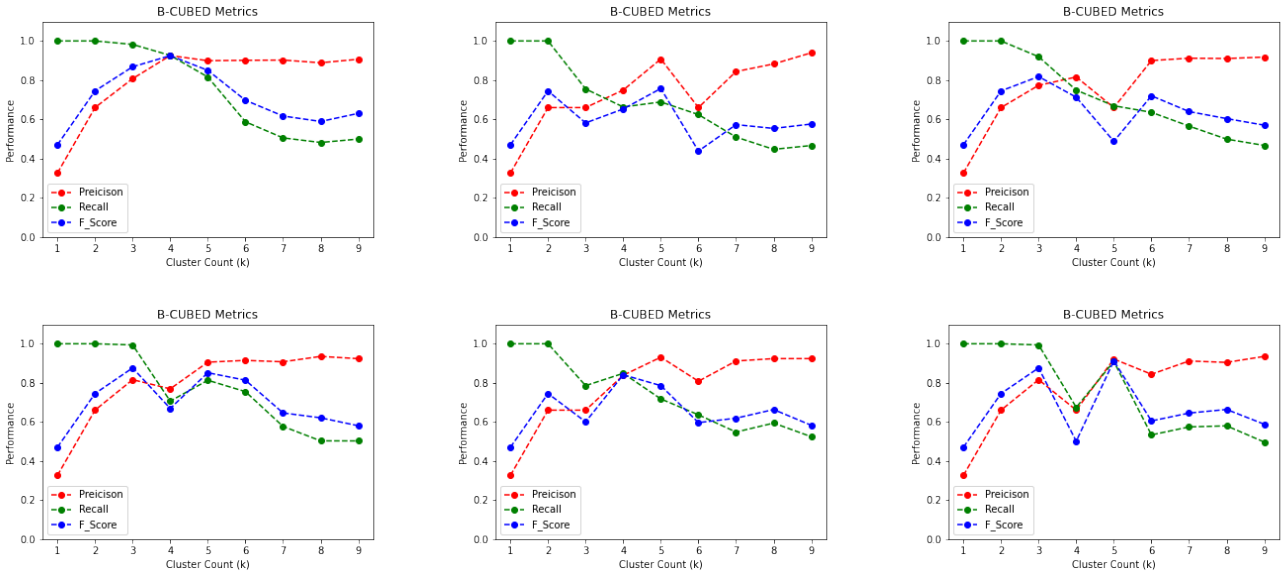

Figure 2: K-Means B-CUBED precision, recall and F-score for k values of 1 to 9. Results shown have been produced using numpy seed values [54, 59] (Left to Right).

Task-4

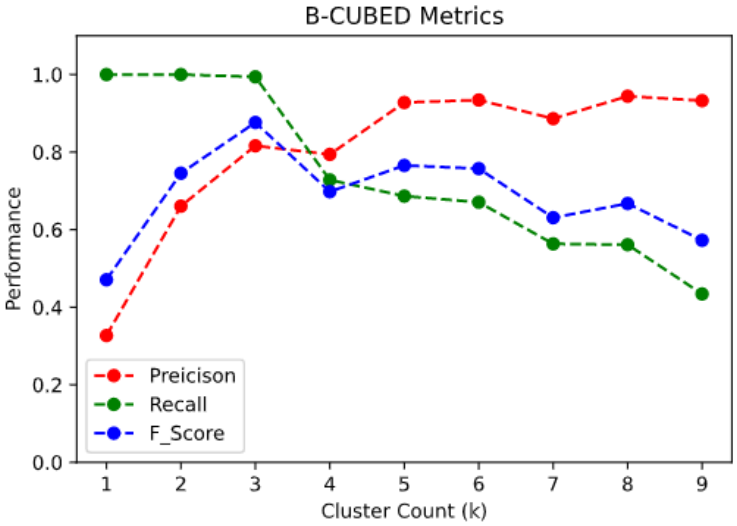

Figure 3: K-Means B-CUBED precision, recall and F-score for k values of 1 to 9, with normalised features - Numpy Seed 54.

Cluster(k)	Precision	Recall	F-Score
1	.33	1.0	.47
2	.66	1.0	.75
3	.82	.99	.88
4	.79	.73	.70
5	.93	.69	.77
6	.93	.67	.76
7	.89	.56	.63
8	.94	.56	.67
9	.93	.43	.57

Table 2: Precision, Recall and F-Score Results - Numpy Seed 54.

Figure-3 and Table-2 describe K-Means with normalisation of feature data, again the results provided use the numpy seed 54. We can see that normalisation in this instance has a negative on the clustering results. Although the Precision performance climbs between k=1 and k=3, it unexpectedly plateaus at k=4. Before climbing higher at k=5.

Additionally, Recall drops substantially between k=3 and k=4, falling below Precision's value in the latter. Thus, unlike Figure-1, we do not have a k value where all three metrics intersect.

Finally, when comparing Table-1 and Table-2, we can see that although the normalised data objects provide higher precision scores, the F-Score is much lower - 0.93 vs 0.88.

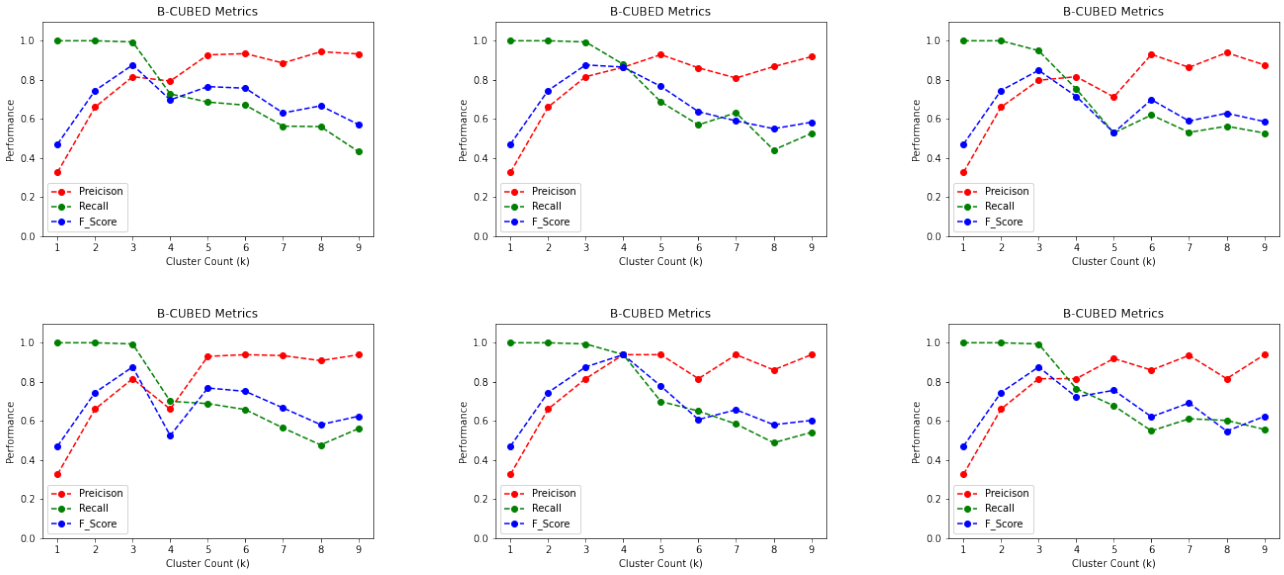

Figure 4: K-Means B-CUBED precision, recall and F-score for k values of 1 to 9, with normalised features. Results shown have been produced using numpy seed values [54, 59] (Left to Right).

Normalisation does not always negatively affect results. When comparing Figure-2's and Figure-4's seed-58 result (the bottom-center subgraph in both Figures), normalisation demonstrates a very positive affect. Smoothing out the graph and providing providing an improved F-score of 0.94 (vs 0.84) - even surpassing the best F-Score in Table-1 (0.93).

Task-5

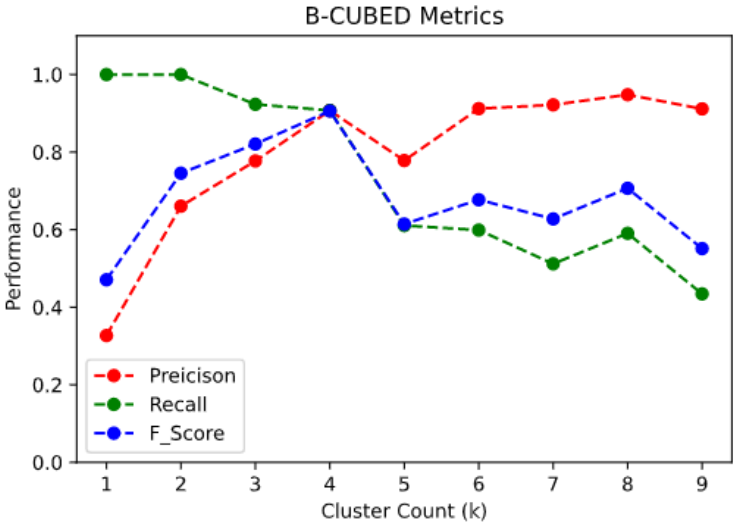

Figure 5: K-Medians B-CUBED precision, recall and F-score for k values of 1 to 9 - Numpy Seed 54.

Cluster(k)	Precision	Recall	F-Score
1	.33	1.0	.47
2	.66	1.0	.75
3	.78	.92	.82
4	.91	.91	.91
5	.78	.61	.61
6	.91	.60	.68
7	.92	.51	.63
8	.95	.59	.71
9	.91	.43	.55

Table 3: Precision, Recall and F-Score Results - Numpy Seed 54.

The K-Medians results, within Figure-5, present themselves in a very similar fashion to K-Means (Figure-1). We can observe a gradual increase in Precision up until k=4. Interesting it appears as silhouette, a peak at k=4 followed by a dip when k=5. This differs from the elbow presentation in K-Means [2]. But, it flattens out when k =6.

Again, Recall presents with 1.0 when k is one or two. Although, the reduction when k=3 is more pronounced than in K-Means and follows a much more rigid pattern overall. This is potentially due to the additional restrictions enforced on the K-Medians algorithm - only allowing centroids to be placed on-top of vectors from the data-set.

F-Score again intersects with both Precision and Recall at k=4. Though the score provided is lower than K-Means at 0.91.

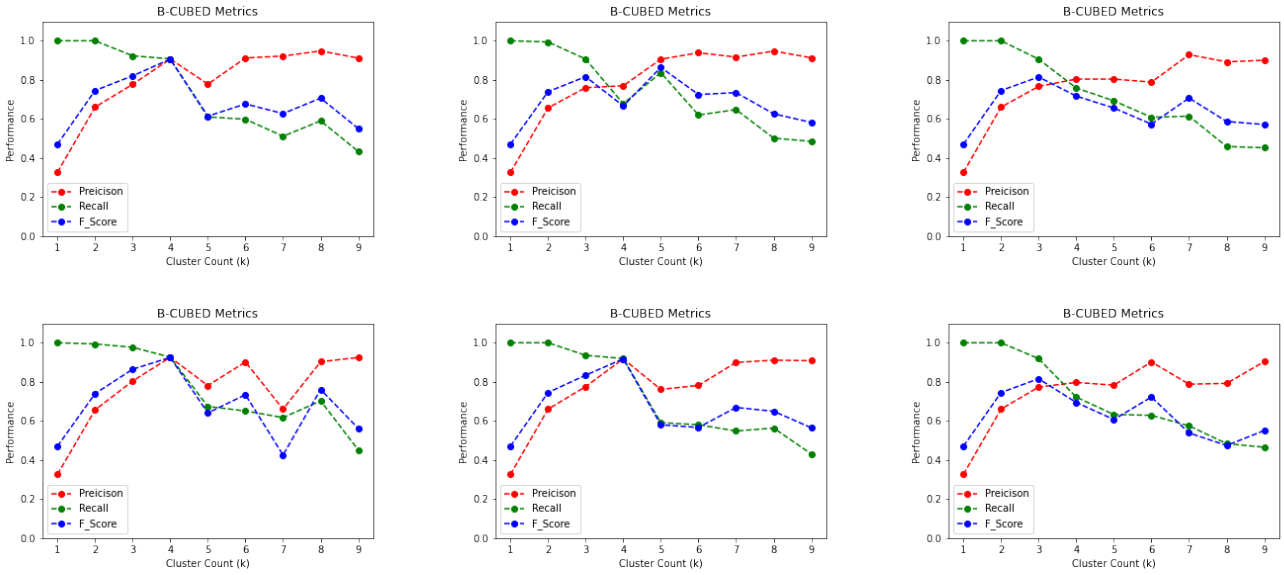

Figure 6: K-Medians B-CUBED precision, recall and F-score for k values of 1 to 9. Results shown have been produced using numpy seed values [54, 59] (Left to Right).

As in Figure-2 and Figure-4, Figure-6 demonstrates that the K-Medians algorithms is also subject to a large degree of randomness. Where the produced model may not be the global optimal.

Task-6

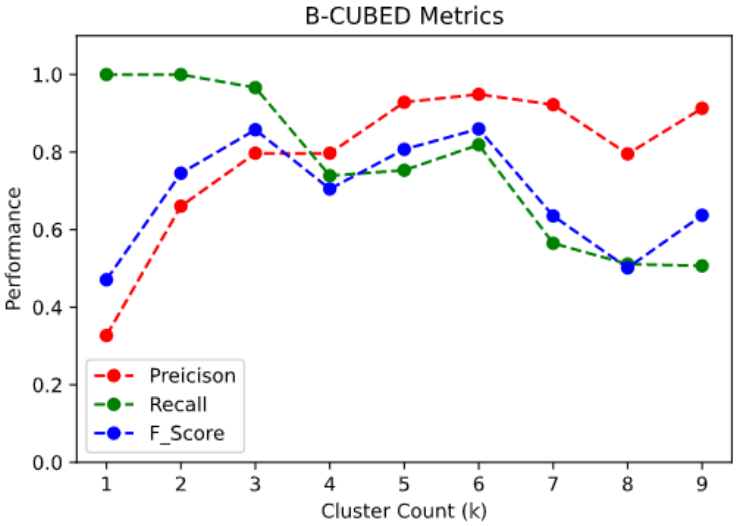

Figure 7: K-Medians B-CUBED precision, recall and F-score for k values of 1 to 9, with normalised features.

Cluster(k)	Precision	Recall	F-Score
1	.33	1.0	.47
2	.66	1.0	.75
3	.80	.97	.86
4	.80	.74	.70
5	.93	.75	.81
6	.95	.82	.86
7	.92	.56	.64
8	.80	.51	.50
9	.91	.51	.64

Table 4: Precision, Recall and F-Score Results

The normalised K-Medians result in Figure-7 displays the same general issue as the normalised K-Means results. Showing an unexpected dip when k=4 as well as no intersection of Precision, Recall and F-Score. Interestingly, we find our best clustering values here to be when k=3 and k=6 - both providing a result of 0.86

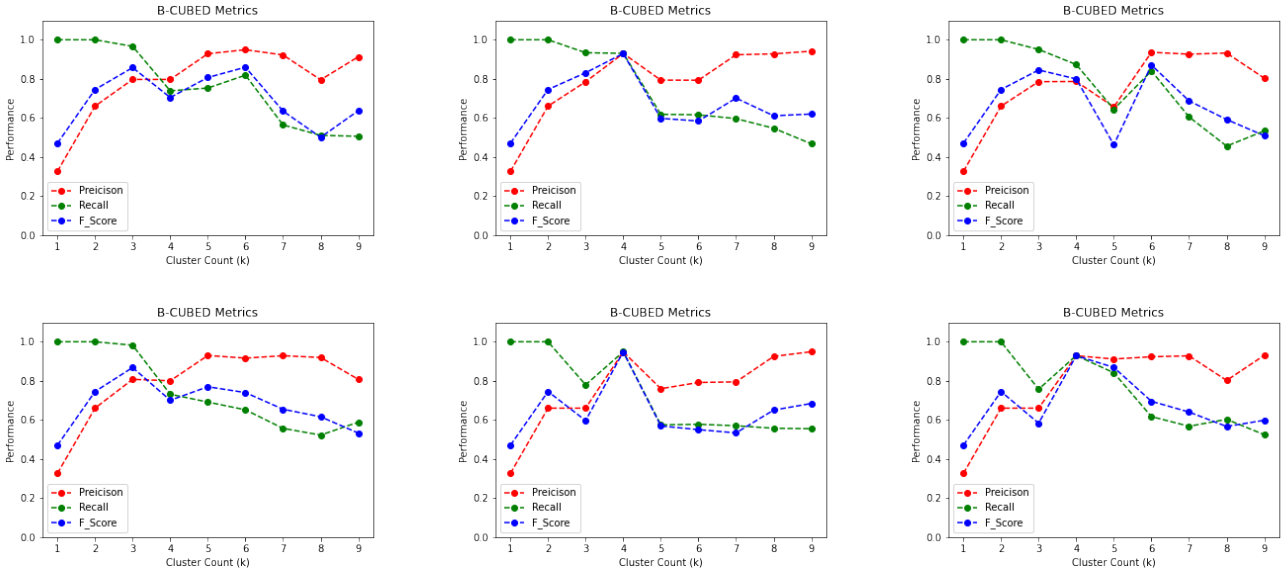

Figure 8: K-Medians B-CUBED precision, recall and F-score for k values of 1 to 9, with normalised features. Results shown have been produced using numpy seed values [54, 59] (Left to Right).

Once again if we compare the un-normalised and normalised graphs from Figure-6 and Figure-8 respectively, we can see instances where normalisation of the objects had both positive and negative affects on the clustering models produced. Where sub-figures 2, 5 and 6 provide the best results.

Task-7

From the limited testing above it is not clear as to whether K-Means or K-Medians provide better models, it is also unclear as to whether the normalisation of the data objects positively or negatively affects them overall. Below are two tables which have been constructed using the average B-CUBED metrics over one-hundred and one-thousand runs. These tables only use the clustering count of k=4, as we know to expect this based on our data.

Question	Precision	Recall	F-Score
K-Means	.80	.76	.73
K-Means (Norm)	.82	.79	.75
K-Medians	.79	.76	.71
K-Medians (Norm)	.81	.78	.74

Table 5: Precision, Recall and F-Score Results For k=4 - Averaged Over 100 Runs Using Numpy Seed [0, 100]

Question	Precision	Recall	F-Score
K-Means	.81	.78	.74
K-Means (Norm)	.83	.80	.77
K-Medians	.80	.77	.73
K-Medians (Norm)	.82	.79	.75

Table 6: Precision, Recall and F-Score Results For k=4 - Averaged Over 1000 Runs Using Numpy Seed [0, 1000]

These tables clearly display that on average K-Means performs better than K-Medians - achieving F-scores of 0.73 and 0.74 vs 0.71 and 0.73, for runs of length 100 and 1000, respectively. I suspect the reason K-Means provides better results is due to it not limiting the positioning of the centroids - freeing them to take up co-ordinates anywhere within the vector space. Whereas the centroids within K-Medians are only able to sit on-top of already existing data-points. Potentially there may be a data-point at an optimal position within a given data-set, though this will not always be the case.

Furthermore, normalisation of the data objects does in fact provide tangible benefits. Within Table-5 and Table-6, for each algorithm, the F-Score increases within the range of [0.02, 0.03]. Although not enormous, the improvement is still significant and has a larger affect than switching between K-Means and K-Medians.

However, as discussed, within the individual tests above we have observed instances where normalisation affected the results negatively. Therefore, it would be prudent when using these clustering algorithms to experiment with different combinations in order to find the best fit for any particular data-set.

References

[1] Viktor Zamaraev. Lecture notes. 2021.

[2] Khyati Mahendru. How to determine the optimal k for k-means?, 2019. URL `https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb`.