Tutorial 7

for

Data Mining - Tutorial 7 Version 1

Adam Ryan (14395076)

COMP40370

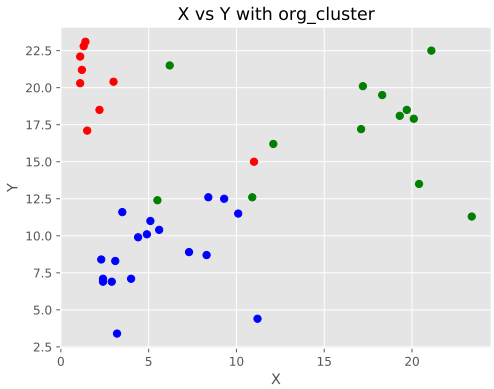
November 9, 2021

Contents

1 Question 1

1.1 Part 1

I visualise the dataframe using matplotlib. The result is below. We see three clusters present broadly in the top left, bottom left, and top right of the plot.



1.2 Part 2

I write a clustering and scoring function, and plot the intertia using the resulting dictionary. We see the elbow point is at three clusters from the graph, as the resultant gains slow considerably after this point. This suggests that we should choose three clusters for the dataset.

1.3 Part 3

I calculate the rand index and silhouette score using sklearn's metrics. For three clusters, I get the values below:

rand $= 0.8666...$

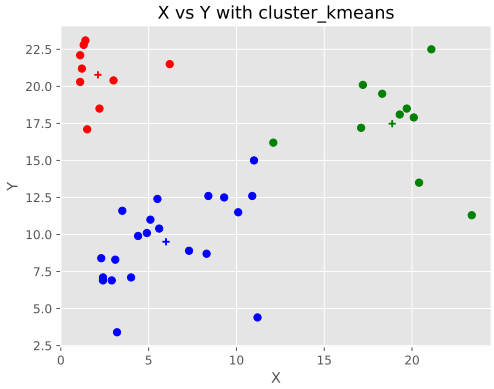
silhouette $= 0.6093075207861811$

1.4 Part 4

I add on this column by taking the result from my clustering dictionary.

1.5 Part 5

I plot the centroids and new clusters using matplotlib and force the label colourings to match the original clusters for plotting purposes (I do not relabel these in the dataframe however the mapping is present in the colouring function if required). The centers are the markers in the graph with the '+' symbol in the same colour as the rest of that cluster.



2 Question 2

2.1 Part 1

I discard the columns by taking the set difference of the original columns and columns to discard, and taking the dataframe with the remaining columns.

2.2 Part 2

I modify the clustering dictionary created for question 1 and modify it for the new parameters and dataset in question 2. This part corresponds the first set of parameters in run_config of '[5,5,100]'. I label the column appropriately in the dataframe.

2.3 Part 3

I modify the clustering dictionary created for question 1 and modify it for the new parameters and dataset in question 2. This part corresponds with the second set of parameters in run_config of '[5,100,100]'. I label the column appropriately in the dataframe.

2.4 Part 4

We see that only nine labels match exactly if we compare the assigned labels between config1 and config2. Changing n_init alters how many different initial centroid starting positions are used by the kmeans++ algorithm, and the best run by inertia is then returned.

2.5 Part 5

I modify the clustering dictionary created for question 1 and modify it for the new parameters and dataset in question 2. This part corresponds with the third set of parameters in run_config of '[3,10,100]'. I label the column appropriately in the dataframe. One point of note is it doesn't say what to use for the second parameter in the question, so I've chosen the sklearn default value of 10.

2.6 Part 6

With the following configuration values, the highest silhouette score is obtained with three clusters, while the lowest inertia is obtained in the second configuration. The precise scores identified are described below.

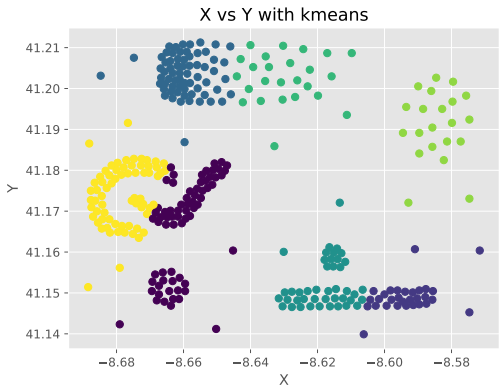
- config1: 5_5_100 : Silhouette: 0.33509164287562027
- config2: 5_100_100 : Silhouette: 0.3601346535452705
- config3: 3_10_100 : Silhouette: 0.46430924739274937
- config1: 5_5_100 : Inertia: 221721.30216033317
- config2: 5_100_100 : Inertia: 221177.56684887334
- config3: 3_10_100 : Inertia: 349679.2299669071

Based on these results, my recommendation would be that config2 appears to produce the most accurate clusters as the inertia is the lowest indicating the points are most tightly clustered, however the improvement is only minor compared to config1.

3 Question 3

3.1 Part 1

I discard ID, create clusters using SKLearn, and generate the plot below. We can clearly see there are five (or six depending on your view of the middle left shape) shapes in the dataset with some random points outside of these shapes falling into a final additional cluster. KMeans fails to appropriately capture this behaviour in the classification as it instead broadly creates clusters depending where in the plot the point falls, missing the graphical shape behaviour of the data.

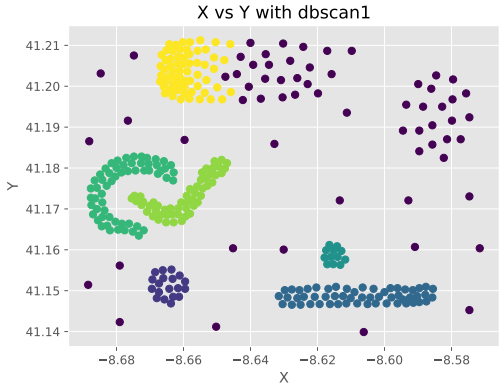


3.2 Part 2

Plot as above in 3.1

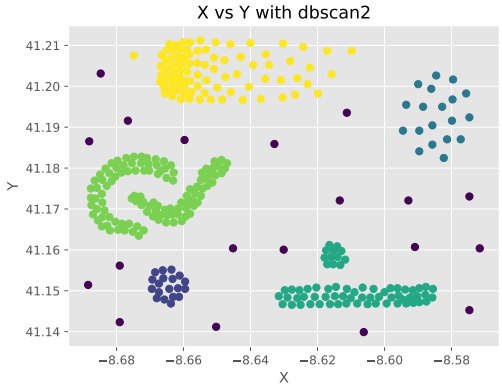
3.3 Part 3

I do this using epsilon=0.04 instead of 0.4 Using 0.4 produced only a single label, while the test case seems to require a value of 0.04 to match the assertion in the test script. I use the minmax scaler to scale things in a range of 0 to 1 as part of my clustering function, and add the dbscan1 results to my dataframe. The following plot is generated:



3.4 Part 4

I do this using epsilon=0.08. I use the minmax scaler to scale things in a range of 0 to 1 as part of my clustering function, and add the dbscan2 results to my dataframe. The following plot is generated:



3.5 Part 5

I save the file using pandas.

3.6 Part 6

Comparing the three graphs obtained in the preceeding question, we see that dbscan2 produces the most accurate clustering result. kmeans fails to account for the 'shape' of the clusters and instead clusters based on their position in the graph (which makes sense given how kmeans is based on the generation of balls with a given metric). DBScan1 is an improvement however it fails to appropriately identify that the 'circle tick' shape on the mid-left of the plot is part of the same cluster, and has difficulties with the shape on the top and the random points. dbscan2 is clearly the best result as it captures the different shapes present in the data and appropriately groups the 'random' data together. This is sensible as dbscan is specifically designed to identify clusters with a distinctive shape and noise present. The difference between kmeans and dbscan is that while kmeans produces convex balls of a given metric around a centroid, dbscan works by assigning points as core points, (which contain at least as many points as the minimum number of points in a ball of a given metric of radius epsilon centered on a given core point), border points (which are themselves contained in a ball surrounding a core point but are not themselves a core point), and outliers which don't fall into the above. The process of creating neighbourhoods around each point works to identify areas of high and low density and cluster them together. By expanding the epsilon in dbscan2, the trailing points in the top shape fall within core and border points clustering them together (whereas epsilon is too tight in dbscan1 to capture them together) and likewise for the shape on the mid-left. This the key reason why increasing epsilon results in better behaviour between dbscan1 and dbscan2.