COMP40370 – Data Mining

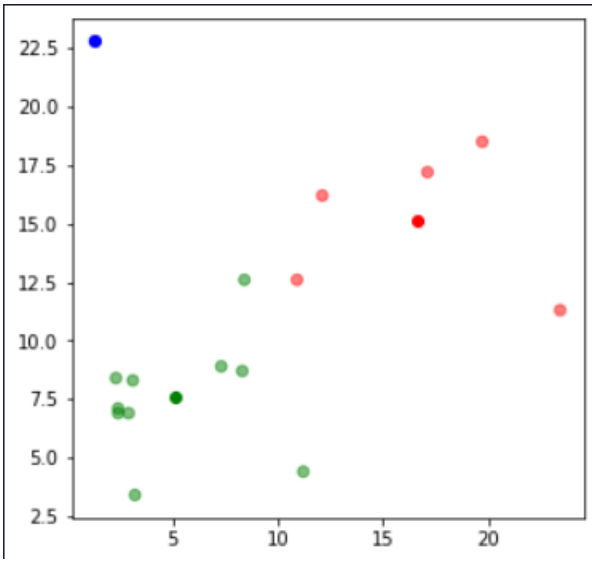
Practical 6

Weronika Wolska – 17301623

Question 1:

The code for this question is in run.py:
Part 1: lines 18-32
Part 2: lines 34-37
Part 3: lines 40-51

The generated graph (also saved in ./output/question_1.pdf) is:



With the three clusters being represented by red, blue and green above. The centroids for each cluster are highlighted using bolder corresponding cluster colours.

As we can see, the blue cluster contains one point, red contains five points and green contains ten points. This means that green has the largest variation, red has the second largest variation and blue has the smallest variation, which has the value of 0, since it only contains one point.

The variation can be illustrated as follows:

Ideally, the variations of all clusters should be equal, or at least close, in size. This means that the solution, although it is the best the algorithm found, is not a good one.

The biggest issue is that the blue cluster has a variation of 0, which is the worst-case scenario. This is because the blue coordinate (1.3, 22.8) is a clear outlier, and there is no other point on the graph that has a Euclidean distance to this point that is shorter than the Euclidean distance to other points.

Question 2:

The code for question 2 can be found in run.py file and the output of the question can be found in ./output/question_2.csv.

2.4 *Are the results for 2.2 and 2.3 different?*

(NOTE: The clusters are labelled 0-9 in the csv file, so cluster number 1 is the 2nd cluster)

Answer: The results obtained are clearly different, as can be seen in the csv file mentioned above. For example, in the first algorithm, the 10th cluster has only 2 brands of cereal, whereas in the second it has 15. Likewise, the 7th cluster has 11 elements using 5 maximum runs (config1) but 3 elements using 100 maximum runs (config2). This insinuates that some cereals in the first algorithm are in different clusters than in the second algorithm, for example, the first row of the file is in the 10th cluster the first time kmeans is applied and in the 7th cluster the second time kmeans is applied. It is also worth noting that some cereals are in the same cluster in both cases, for example the 35th entry, (row 36 of the csv file), is in the 5th cluster both times kmeans is applied.

This means that the clusters have different sizes (and hence variations) depending on the number of maximum runs specified.

Config 1:

Config 2:

2.6 *Which clustering solution is better?*

Answer:

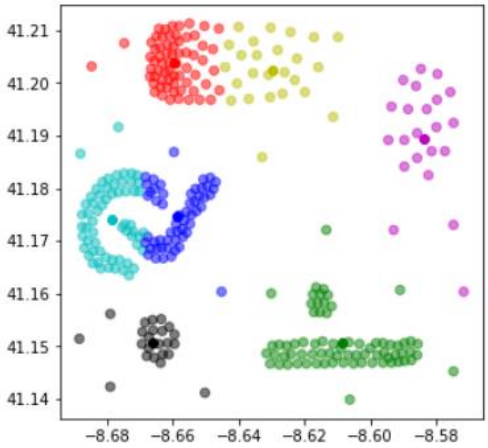
Just by looking at the 'config3' column we can see that a lot of cereals are assigned to the first cluster ('0'), 47 to be exact. While the second and third clusters contain 14 and 16 cereals, respectively. If we represent this visually:

We can see that the difference in variation is significant. Ideally, we want the clusters to be almost equal in size.

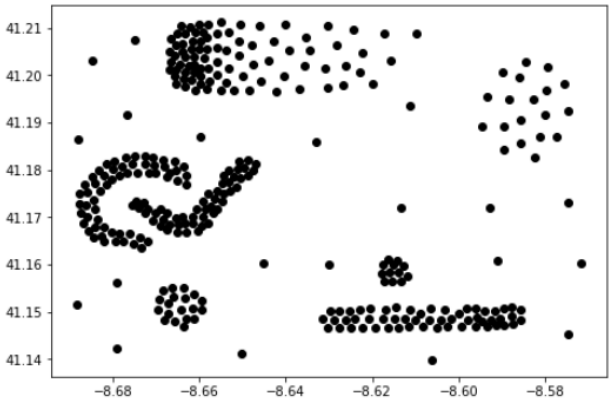
This is why using 10 clusters instead of 3 is a better solution to this particular set of data, as the results are more consistent. Both of the results obtained using 10 clusters are equal in variation and both have a large difference between the smallest variation and the largest variation, therefore it is difficult to say that one of these is better than the other, though they are both clearly better than the result obtained using three clusters.

Question 3:

K-Means graph generated for Part 2 (* ‘./output/question_3_1.pdf’), using 7 clusters, 5 maximum runs and 100 maximum optimization steps:

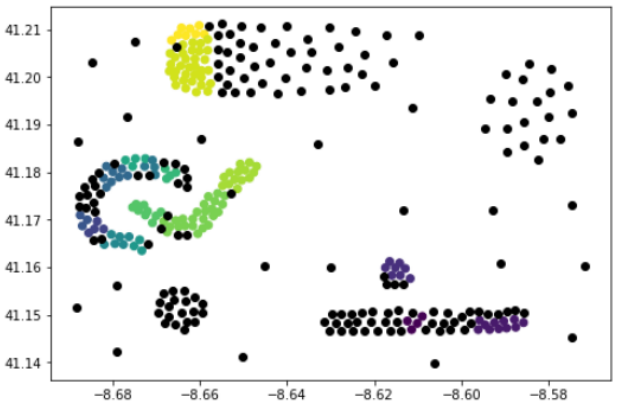


Graph generated for Part 3 (* ‘./output/question_3_2.pdf’), using DBScan clustering, with epsilon=0.04:



NOTE: Black is the colour assigned to outliers, i.e. points that are assigned “cluster” -1.

Graph generated for Part 4 (* ‘./output/question-3_3.pdf’), using DBScan clustering, with epsilon=0.08:



3.6 *Discuss the different clustering solutions in your report. Which solution is the best? What is the reason behind the differences in the results?*

From the graphs above, it is clear that using the kmeans algorithm gives the best results, as all of the data points are assigned to a cluster and the clusters look reasonably distributed, though they have large differences in variation. Meanwhile, using DBScan with epsilon=0.04 identifies all points as outliers, meaning the data is not split into clusters, which is an extremely poor solution. Using DBScan with epsilon=0.08 gives a slightly better solution, however, 175 points are still counted as outliers and the remaining points are split into 146 different clusters, meaning that the data is still difficult to operate on, which defeats the purpose of splitting it into clusters.

The difference in results is due to the data being normalised for performing the DBScan. Although normalising has many benefits, such as increasing computation speed, it can also be bad for the data as it leads to information loss on min and max values of the data. As well as that, the dataset for the question is reasonably large in size, and DBScan is known to perform less efficiently on large datasets.

The two clustering solutions have a number of differences: For k-means, user needs to specify the number of target clusters, which is not necessary with DBScan, as can be seen from the implementation in run.py. DBScan is also more efficient at handling outliers and noisy data, unlike k-means. As outlined above, k-means clustering also performs better with large datasets, while DBScan struggles to handle large data efficiently.