Amin Fadaee

Machine Learning HW5

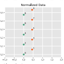
Python 3.5, Libraries: Numpy, Matplotlib, Pandas, Math, random, sklearn Documentation: HTML, Bootstrap 4, jqMath

Problem 1

- **Average:** This is the most intuitive way to represent a cluster as it has the least distace from all the points in the cluster.
- **Mode:** Using mode is a simple way to represent a cluster as it is not influenced by the structure and the space of the points directly and also it can be used in non-euclidean spaces as well.
- **Median:** Due to its robustness, using median is most useful when there are outliers present in the data.

Problem 2

K-Means with and without normalization will have **different** results. This is due to the fact that after normalization, the previous distances wouldn't be preserved and there is no garantee that we achieve the same clustering results. Here is an example with k=2 demonstrating this fact:



As can be seen the obtained clusters are different for each set.

Problem 3

All 3 of the criterion have the same complexity. If we were to find the linkage between two clusters we would have to find the distance between each point in the first cluster (having m points) from the points in the second cluster (n points) which results in $O(mn)$ complexity. Using the obtained distances ($d_{i,j}$), the formula for deriving the linkages are as following:

Single: $\text{Min}(d_{i,j})$
Complete: $\text{Max}(d_{i,j})$
Average: $\frac{\text{Sum}(d_{i,j})}{m \times n}$

Single and Complete linkage are prone to outliers, if a point deviates from the majority of the points in cluster it can be mistaken for a point too close or too far from the other clusters, however, average linkage is robust to outliers.

Problem 4

The hyperparameters of DBSCAN has a lot of effect and the final clusters. If we want to derive to large clusters a high radius is needed and in the other hand by choosing a small radius the number of clusters would increase. From other perspective if we know in advance the density of the clusters, we can choose the minpoints properly. A low minpoints would result in low density clusters and a set of dense cluster would require a higher minpoints.

In absence of strong prior knowledge of the clusters structure, we can utilize the grid search and validation set error as a method for finding the hyperparameters.

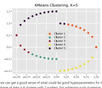
Problem 5

A. The codes solving this problem can be found in `Problem 5.py`. Also the implementation of Kmeans and DBSCAN is respectively in `KMeans.py` and `DBSCAN.py`.

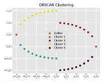
Here are the results for each k:

- 2: 0.82501
- 3: 0.74936
- 4: 0.60137
- 5: 0.57372

As can be seen $k = 5$ would result in the best DB-Index. Here is the plot of the data clustered by K-Means with k=5:



B. As the data is 2 dimensional we can get a good sense of what could be good hyperparameters for the dbscan algorithm. A good result judging based on the shape of data is 4 clusters with 2 outliers. For achieving such clustering, $epsilon = 0.3$ and $minpoints = 3$ seems like a good choice. Here is a plot depicting dbscan using these parameters:



C. Using exploratory analysis and getting help from the data we can deduce that there are 4 clusters along with 2 outliers which dbscan captured perfectly!

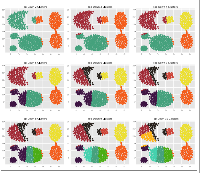
Problem 6

For the top down clustering, K-Means with K=2 is used. Also, for finding the cluster to divide, we pick a cluster with the highest average dissimilarity given by the formula below:

$$\underset{c}{argmax} \sum_{i} \| X_{ci} - \overline{X}_c \|_2^2$$

Where X_c is the data in cluster c.

Here are the results of the Top Down Clustering:



For measuring the quality of the number of clusters C, the CH-Index is used:

$$CH = \frac{B \ / \ (C-1)}{W \ / \ (N-C)}$$

Where W is the within cluster variation and B is the between cluster variation:

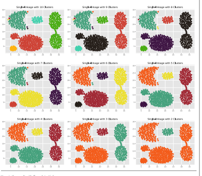
$$W = \sum_{c=1}^{C} \sum_{i} \| X_i - \overline{X}_c \|_2^2$$

$$B = \sum_{c=1}^{C} n_c \| \overline{X} - \overline{X}_c \|_2^2$$

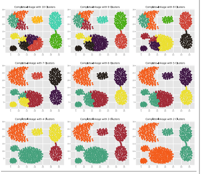
n_c is the number of point belonging to cluster c and $\overline{X}$ is the mean of all data.

A complete discussion of the quality of the algorithms based on CH is discussed at the end of this problem.

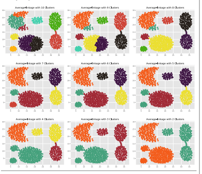
Using Single-Linkage bottom up clustering, we derived the following results:



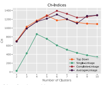
Here is the result with Complete-Linkage:



And Average-Linkage:



Using CH-Index we can compare the results of the above clustering using the following graph:

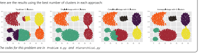


As can be seen, the best number of clusters for each approach is:

- 5 for Top Down
- 4 for Single-Linkage
- 6 for Complete-Linkage
- 6 for Average-linkage

With Complete-Linkage having the highest CH.

Here are the results using the best number of clusters in each approach:



The codes for this problem are in `Problem 6.py` *and* `Hierarchical.py`