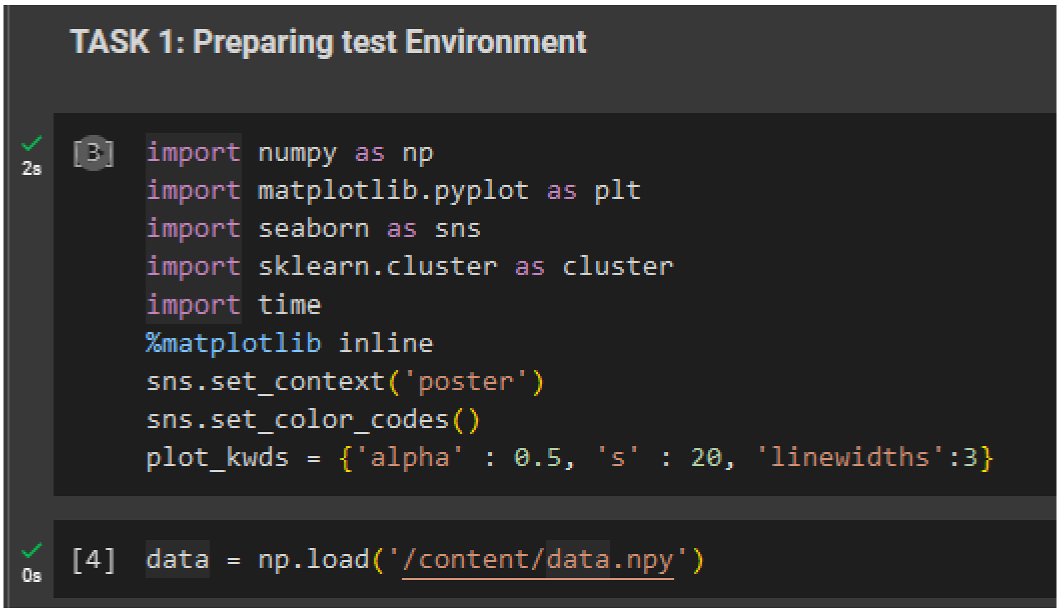
TABET Akram Naoufel
w1979246

Comparing SKLEARN Clustering Algorithms

1) TASK 1: Preparing test Environment:

A. evidence of Loading the 'data.npy' data file



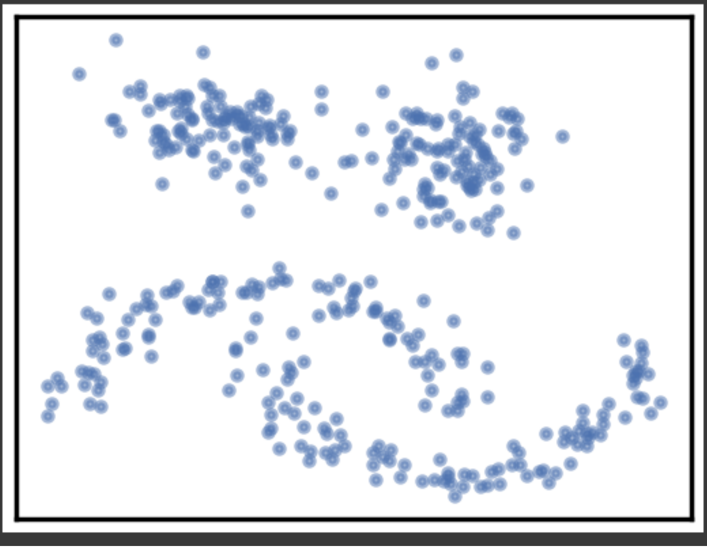
TASK 1: Preparing test Environment

```
[3] import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    import sklearn.cluster as cluster
    import time
    %matplotlib inline
    sns.set_context('poster')
    sns.set_color_codes()
    plot_kwds = {'alpha' : 0.5, 's' : 20, 'linewidths':3}
```

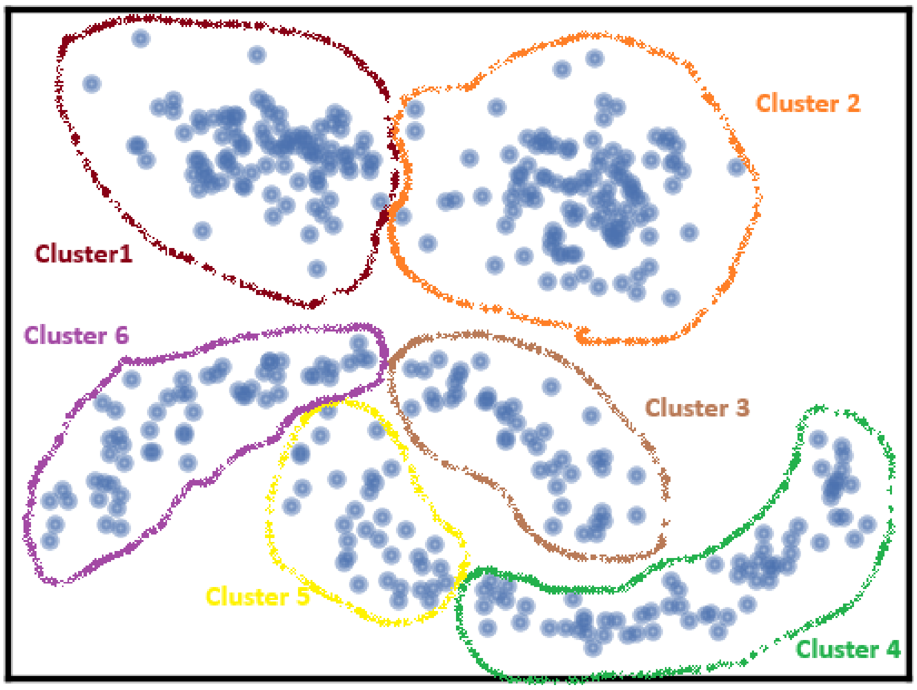
```
[4] data = np.load('/content/data.npy')
```

B. Scatter plot the 'data.npy' data

```
[5] plt.scatter(data.T[0], data.T[1], c='b', **plot_kwds)
    frame = plt.gca()
    frame.axes.get_xaxis().set_visible(False)
    frame.axes.get_yaxis().set_visible(False)

    # plt.savefig('scatterplot.png')
```

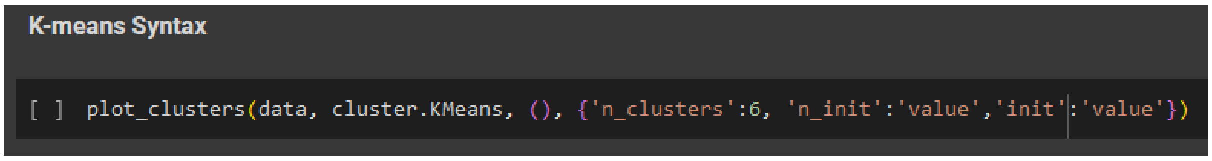


C. Annotation of the various groups of data as I observe them



D. The syntax of each algorithm with the little utility function.

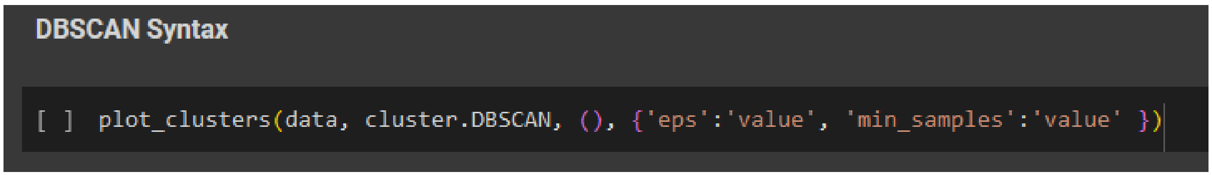
a. K-means



K-means Syntax

```
[ ] plot_clusters(data, cluster.KMeans, (), {'n_clusters':6, 'n_init':'value','init':'value'})
```

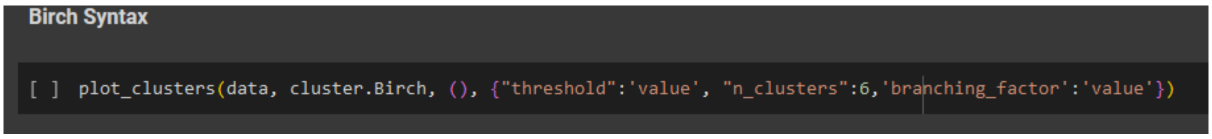
b. DBSCAN



DBSCAN Syntax

```
[ ] plot_clusters(data, cluster.DBSCAN, (), {'eps':'value', 'min_samples':'value' })
```

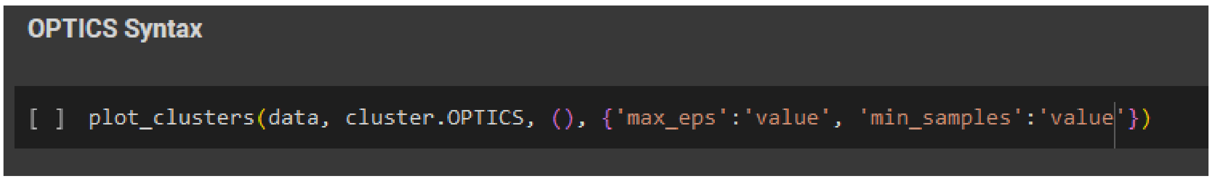
c. Birch



Birch Syntax

```
[ ] plot_clusters(data, cluster.Birch, (), {"threshold":'value', "n_clusters":6,'branching_factor':'value'})
```

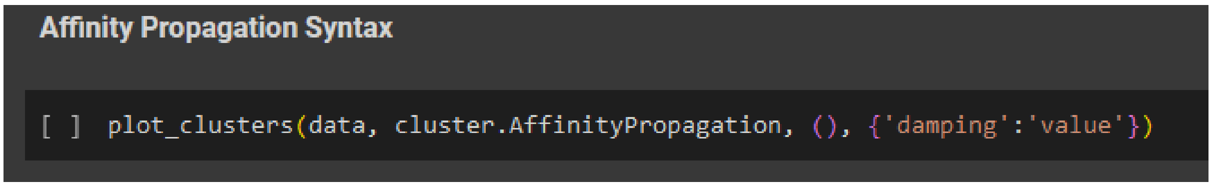
d. Optics



OPTICS Syntax

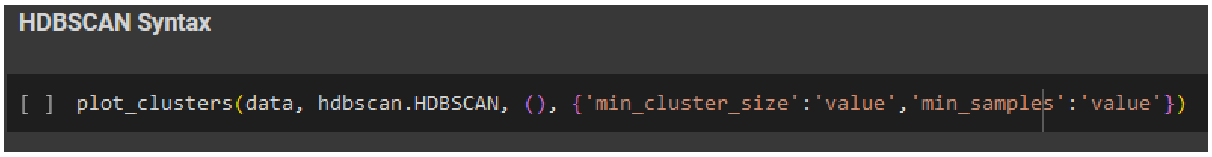
```
[ ] plot_clusters(data, cluster.OPTICS, (), {'max_eps':'value', 'min_samples':'value'})
```

e. Affinity Propagation


Affinity Propagation Syntax

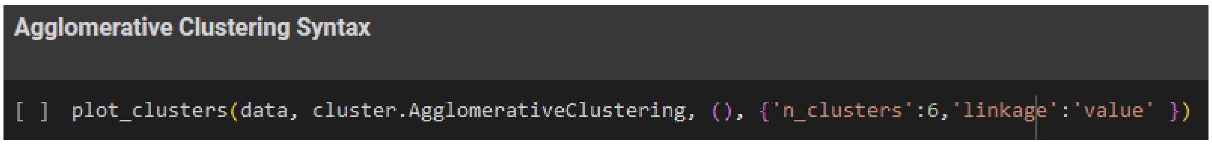
```
[ ]  plot_clusters(data, cluster.AffinityPropagation, (), {'damping':'value'})
```

f. HDBSCAN


HDBSCAN Syntax

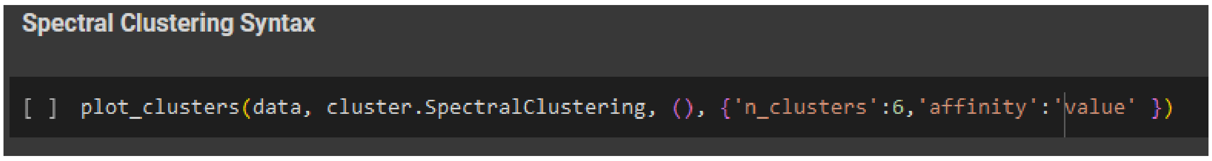
```
[ ]  plot_clusters(data, hdbscan.HDBSCAN, (), {'min_cluster_size':'value','min_samples':'value'})
```

g. Agglomerative Clustering


Agglomerative Clustering Syntax

```
[ ]  plot_clusters(data, cluster.AgglomerativeClustering, (), {'n_clusters':6,'linkage':'value' })
```

h. Spectral Clustering


Spectral Clustering Syntax

```
[ ]  plot_clusters(data, cluster.SpectralClustering, (), {'n_clusters':6,'affinity':'value' })
```

2) TASK 2: Choosing each algorithm's parameters values:

Note: Our approach involves the careful selection of optimal parameters for each algorithm, along with their corresponding values, using an artificial dataset designed for research. It is important to recognize that the ideal parameter values may differ, and I advise exploring a range of values while evaluating clustering results to determine the most effective one.

Algorithm Name	args or kwds used	Args or kwds values	Justification
K-Means	number of clusters	6	Looking at the generated scatter plot, we can conclude that setting k to 6 clusters sounds reasonable enough(based on the expected number of clusters). We can use elbow and silhouettes methods as well**[2]**.
	n_init	auto	Several runs are recommended for sparse high-dimensional problems. When `n_init='auto'`, the number of runs depends on the value of init**[2]**.
	init	k-mean++	For efficient initialization, this technique speeds up convergence**[1]**.
	max_iter	default:300	Maximum number of iterations of the k-means algorithm for a single run, it depends on convergence behavior**[2]**.

Affinity Propagation	Damping	[0.5,1]	Use a `damping` value between 0.5 and 1 to control the impact of messages on cluster centers. A higher damping factor stabilizes the updates**[3]**.
DBSCAN	epsilon or “eps”	0.15	To choose the value of ε, a k-distance graph is plotted by ordering the distance to the *k=MinPts-1* nearest neighbor from the largest to the smallest value. Good values of ε are where the plot shows an “elbow”. If ε is too small, a large portion of the data will not be clustered. If ε is too high, clusters will merge, and most objects will be in the same cluster. Small values of ε are generally preferred, and only a small fraction of points should be within this distance of each other. According to our simulation we can conclude that the optimal value of epsilon is between 0.15 and 0.20**[4].**
	`min_samples`	4	To derive the minimum value for *MinPts*, a rule of thumb is to set it to be greater than or equal to the number of dimensions *D* in the data set, i.e., *MinPts* ≥ *D* + *1*. A low value of *MinPts*, such as 1, is not meaningful because it results in each point forming its own cluster. If *MinPts*≤ 2, the result will be similar to hierarchical clustering with the single link metric, where the dendrogram is cut at height ε. Therefore, *MinPts* must be at least 3. However, larger values are usually better for data sets with noise as they lead to more significant clusters. For 2-dimensional data, use DBSCAN’s default value of MinPts = 4 (Ester et al., 1996). If the data has more than 2 dimensions, choose MinPts = 2*dim, where dim= the dimensions of your data set (Sander et al., 1998)**[4]**.
Birch	`threshold`	Typically set between 0.2 and 0.5(0.5 default according to scikit learn)	This parameter determines the radius of the subcluster. Setting it too low may lead to overfitting, while setting it too high may merge clusters. A moderate value, such as 0.2 to 0.5, often works well in practice**[5]**.

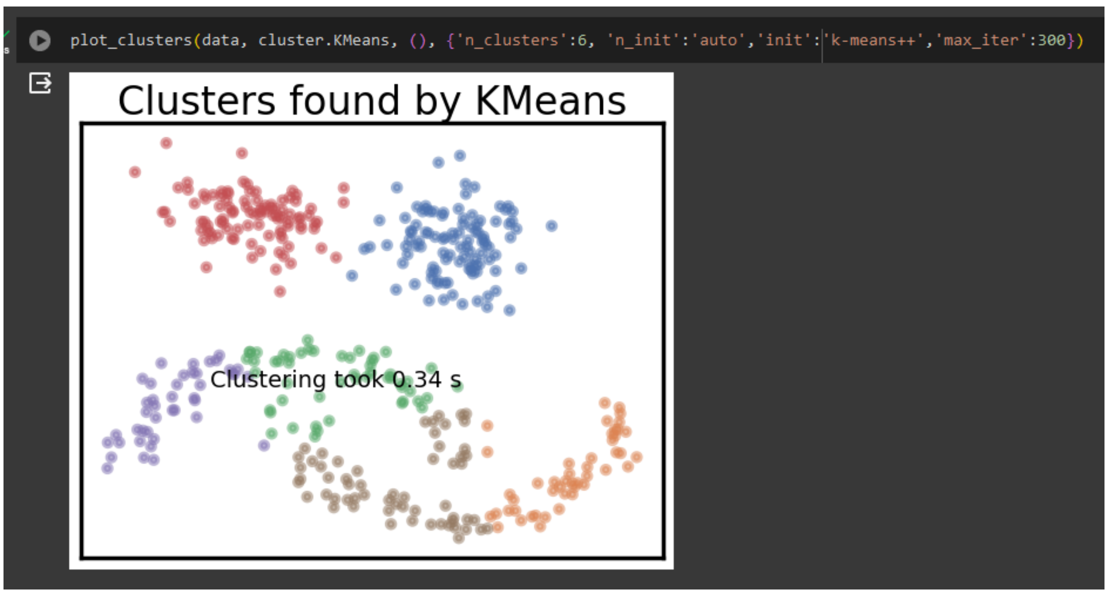
	`n_clusters`	6	Set based on the expected number of clusters**[2]** and by visual inspection.
	`branching_factor`	A commonly used value is 50	This parameter controls the number of subclusters in each node. A higher value may lead to a smaller tree size but higher memory usage. A value around 50 is often a good balance**[5]**.
OPTICS	`min_samples`	slightly higher value > 5	artificial data includes a significant amount of noise or outliers, you might consider a slightly higher value for `min_samples` to filter out small, isolated regions that might be considered noise.According to our visual inspection,we are not anticipating a clusters with a small number of points,based on that we are not going to select a lower value of min_samples**[6]**.
	`max_eps`	default=np.inf	Default value of `np.inf` will identify clusters across all scales; reducing `max_eps` will result in shorter run times**[7]**.
HDBSCAN	`min_samples`	Commonly set between 1 and 10.	`min_samples` clearly has a dramatic effect on clustering, the question becomes: how do we select this parameter? The simplest intuition for what `min_samples` does is provide a measure of how conservative we want clustering to be. The larger the value of `min_samples` we provide, the more conservative the clustering – more points will be declared as noise, and clusters will be restricted to progressively more dense areas. In our case and according to visual inspection,we will adjust it between 1 and 5**[8]**.
	`min_cluster_size`	Slightly high value>10	According to the official website of HDBSCAN and their experiment,The primary parameter to effect the resulting clustering is `min_cluster_size`. Ideally this is a relatively intuitive parameter to select – set it to the smallest size grouping that we wish to consider a cluster. Increasing it reduces the number of clusters and merges them. This is a result of HDBSCAN* reoptimizing which flat clustering provides greater stability under a

			slightly different notion of what constitutes a cluster**[9]**.
Agglomerative Clustering	`n_clusters`	6	Set based on the expected number of clusters**[2]** and according to the desired hierarchical structure**[10]**.
	`linkage`	Average	The choice of this parameter varies according to specific use cases. For instance, the 'ward' linkage is suitable when anticipating clusters with similar sizes and shapes, while 'average' is generally versatile. However, it's essential to evaluate each parameter individually to understand its performance characteristics in a given context**[10]**.
Spectral Clustering	`n_clusters`	6	Set based on the expected number of clusters by the visual inspection of our artificial dataset**[2]**.
	`affinity`	Common choices include 'nearest_neighb ors', 'rbf' (Radial basis function), or 'poly' (Polynomial).	The choice of affinity depends on the nature of the data. 'Nearest_neighbors' is suitable for sparse datasets, while 'rbf' and 'poly' are effective for capturing non-linear relationships. Consider experimenting with different affinities to find the most appropriate**[11]**.
	`gamma`	float,default=1,(i f using 'rbf' or 'poly'): Controls the width of the kernel and might need adjustment**[12]**.	The optimal value may depend on the scale and distribution of our artificial dataset**[11]**.

3) TASK 3: Testing and documenting the output of the algorithms

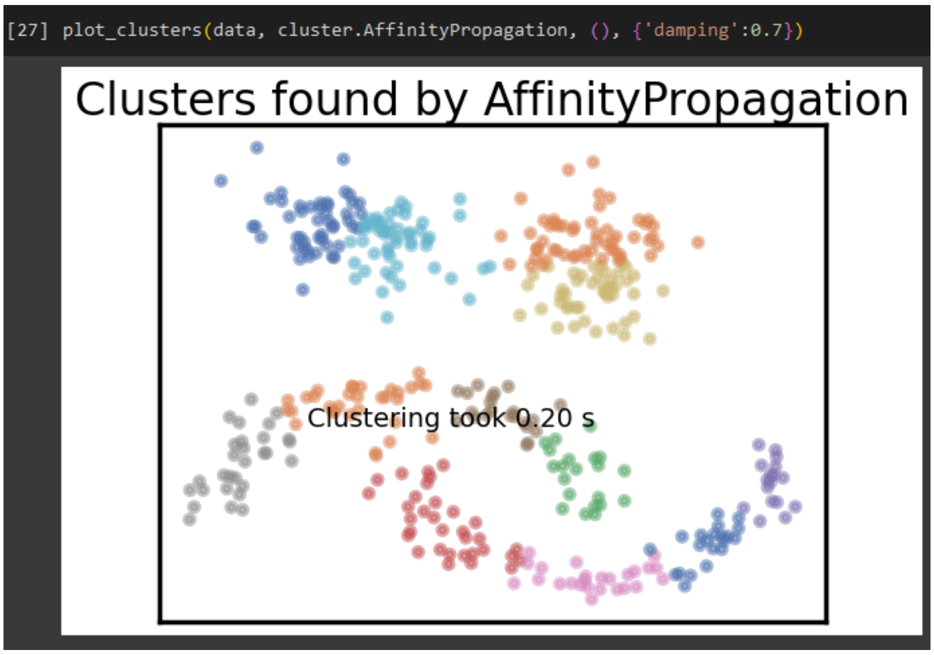
In this section,we are going to execute each algorithm and plot the output. In task 4we will document each algorithm and present our own findings and thoughts.

1) K-means



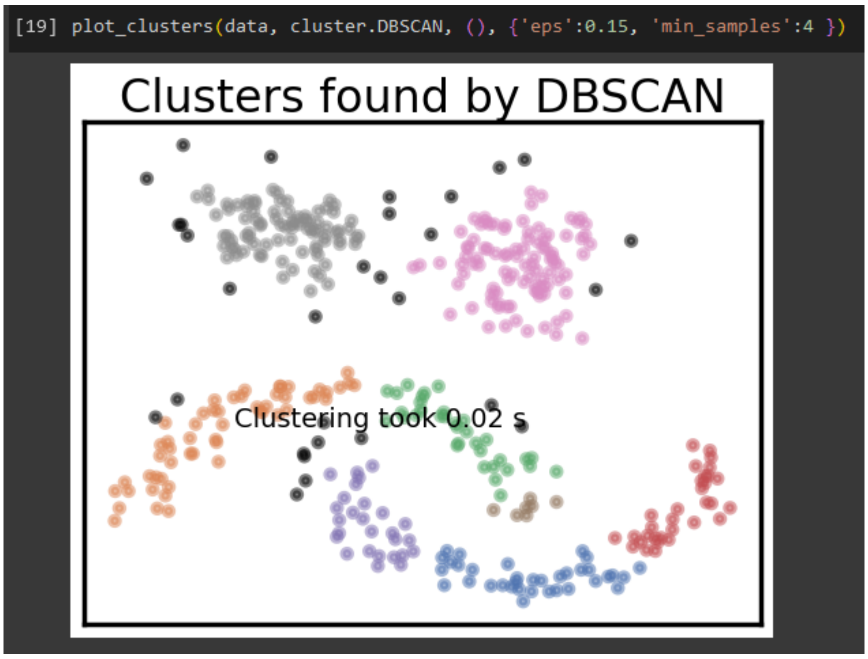
```
plot_clusters(data, cluster.KMeans, (), {'n_clusters':6, 'n_init':'auto','init':'k-means++','max_iter':300})
```

2) Affinity Propagation



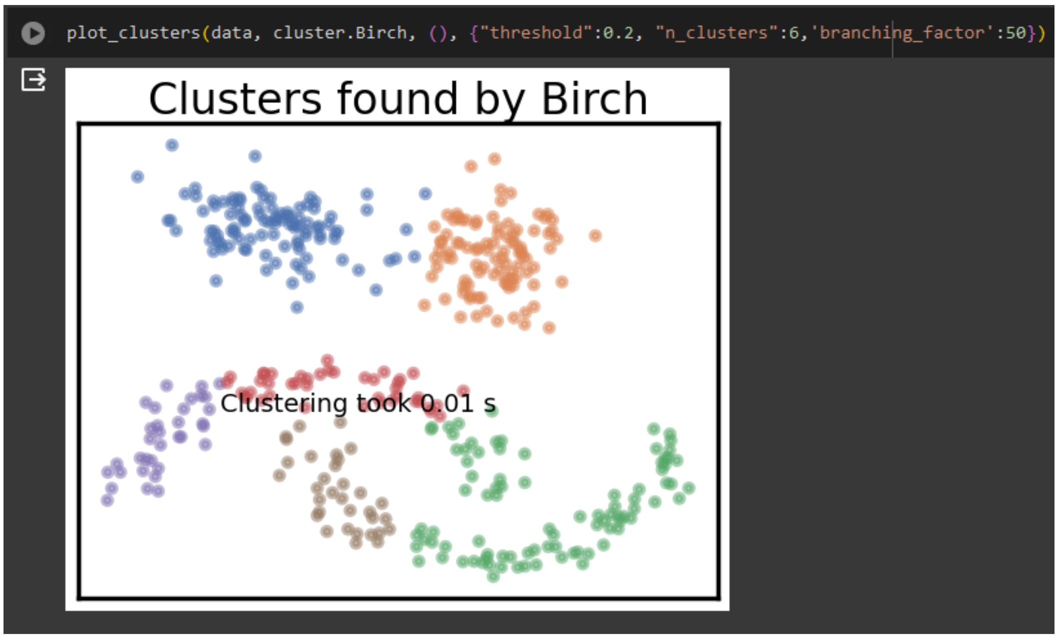
```
[27] plot_clusters(data, cluster.AffinityPropagation, (), {'damping':0.7})
```

3) DBSCAN



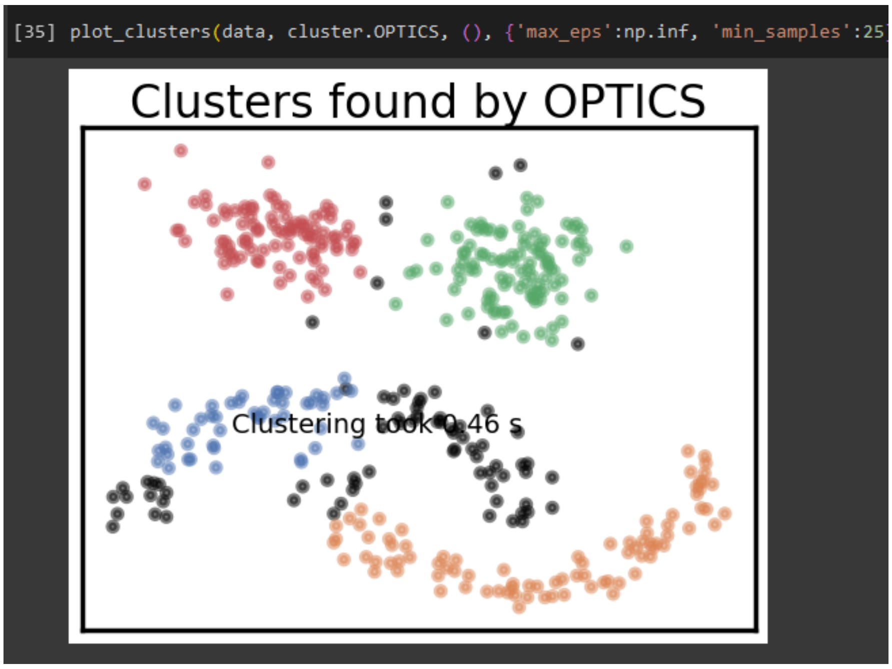
```
[19] plot_clusters(data, cluster.DBSCAN, (), {'eps':0.15, 'min_samples':4 })
```

4) Birch



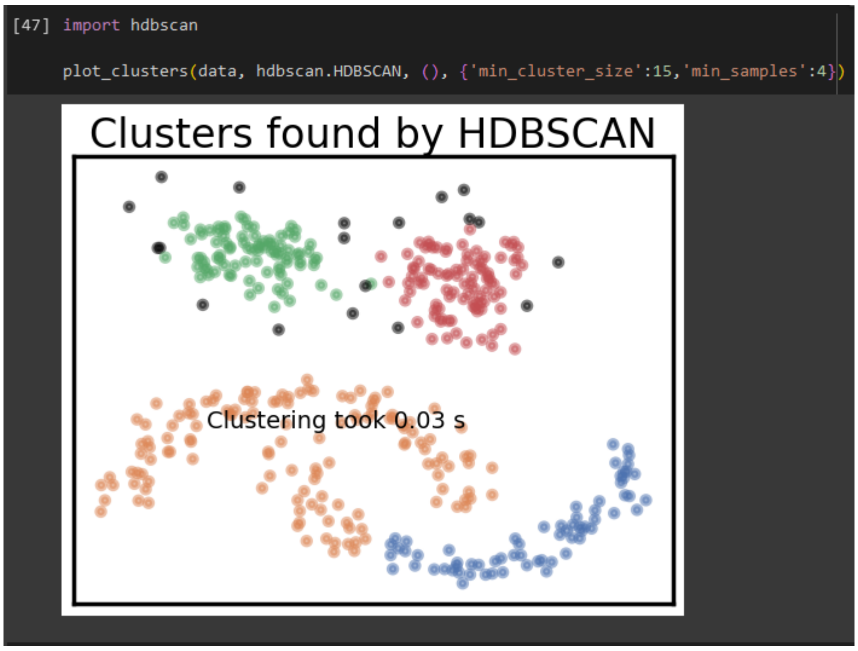
```
plot_clusters(data, cluster.Birch, (), {"threshold":0.2, "n_clusters":6,'branching_factor':50})
```

5) Optics



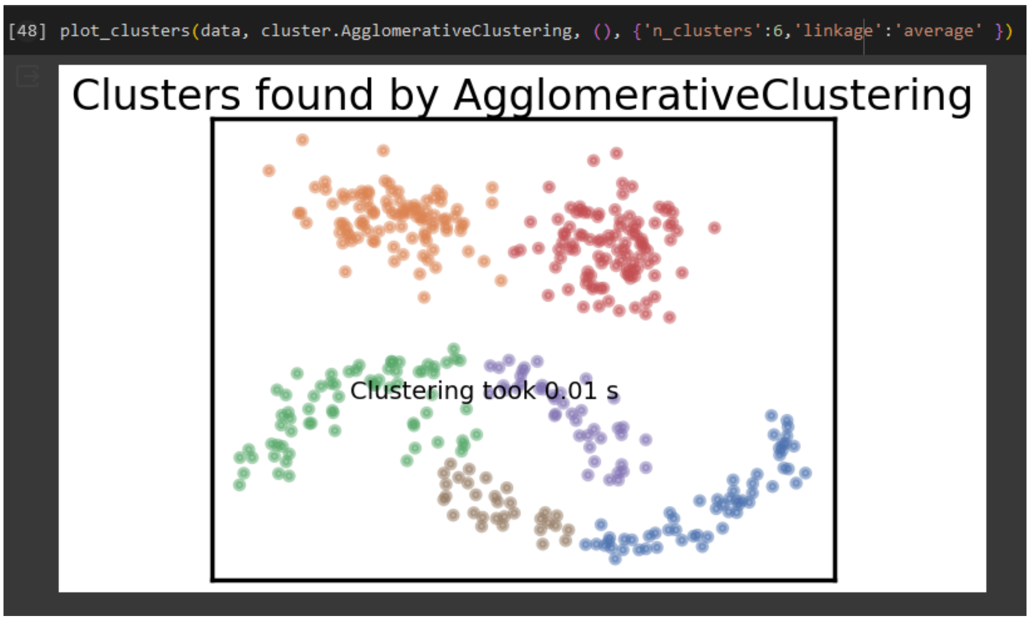
```
[35] plot_clusters(data, cluster.OPTICS, (), {'max_eps':np.inf, 'min_samples':25
```

6) HDBSCAN



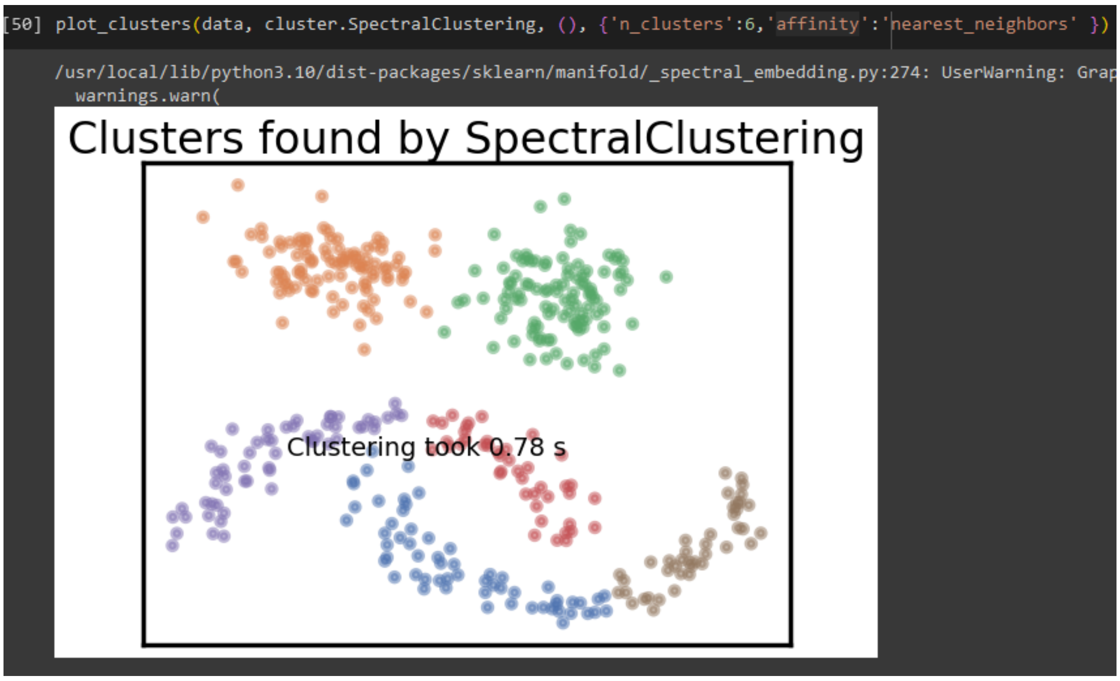
```
[47] import hdbscan

     plot_clusters(data, hdbscan.HDBSCAN, (), {'min_cluster_size':15,'min_samples':4})
```

7) Agglomerative Clustering



```
[48] plot_clusters(data, cluster.AgglomerativeClustering, (), {'n_clusters':6,'linkage':'average' })
```

8) Spectral Clustering



```
[50] plot_clusters(data, cluster.SpectralClustering, (), {'n_clusters':6,'affinity':'nearest_neighbors' })

    /usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:274: UserWarning: Grap
      warnings.warn(
```

4) Task 4 Writing a short technical report analysing the “Clustering Results

a) Introduction

In the world of data analysis, clustering algorithms play a crucial role in helping us find patterns in datasets. Clustering is like grouping similar things together, which can reveal interesting insights. However, the success of clustering depends on choosing the right method that fits the data we're working with. In this short technical report, we're going to thoroughly examine different clustering methods using a specially created dataset. Our goal is to understand how well each method works for our data and figure out which one is the best fit for the job.

b) Clustering Algorithm Analyses:

Now, let's take a closer look at each clustering method one by one. I'll examine how effective they were in sorting our artificial data into groups(Given domain). I'll discuss the specific settings they used (keys and values), how well they managed distances within and between clusters(Inter/Intra clusters distance), and I'll showcase the results through scatter plots for each algorithm. After evaluating all the algorithms, I'll share my thoughts on which one I believe performed the best for our specific task. This conclusion will be based on how well each algorithm understood our artificial data and the patterns it uncovered.
K-means: Upon observing the scatterplot, K-Means effectively grouped our artificial data into 6 clusters, aligning with our initial expectations from visual inspection. While it doesn't precisely match our manual clustering, there's notable similarity, especially in the upper clusters. Four key parameters were utilized:

a. Number of Clusters (k): Set to 6, aligning with the anticipated clusters.
b. n_init: Employed 'auto' for multiple runs, enhancing convergence as suggested in the article.

c. init: Applied 'k-means++' for efficient initialization, expediting convergence.
d. Max Iterations: Defaulted to 300, considering convergence dynamics.

Analyzing inter-class distance reveals the red and blue clusters are distinctly separated, particularly in the topside groups. The green and brown clusters exhibit satisfactory but comparatively lower inter-class distances. Regarding intra-class distance, it appears normal, with shorter distances observed in the red and blue clusters, indicating some points overlap. While overall the results are sound, the algorithm's performance, while commendable, slightly falls short of our visual inspection. Notably, there are no outliers,all the points are clustered and the algorithm's effectiveness is affirmed.

Affinity Propagation: Upon inspecting the scatterplot, the Affinity Propagation algorithm produced 12 clusters, twice the number derived from K-Means and our initial visual estimate. The algorithm subdivided each of the two main clusters into two, resulting in a total of 4 clusters in the topside and lower regions. While deviating from our anticipated clustering, it demonstrated relatively fewer errors compared to subsequent algorithms.

As for parameters, Affinity Propagation has a single compulsory parameter: damping. A damping factor of 0.7 was chosen to control the impact of messages on cluster centers, aiming for stability in updates. Analyzing inter-class distance, the topside and downside clusters remain distinct due to the inherent structure of our data. However, within the upper cluster, inter-distance is close, discernible mainly through color. The lower clusters similarly exhibit close inter-class distances, indicating less-than-optimal separation. In terms of intra-class distance, the overall closeness is promising, yet some overlap, especially in the orange and yellow clusters, is noted.

Documenting the scatterplot, Affinity Propagation performs reasonably well, although the deviation in the number of clusters is notable. No outliers were observed, and all points successfully clustered.

DBSCAN: DBSCAN generated 7 clusters, flagging some points in black as outliers. It successfully identified the upper clusters as two primary clusters, but overall, its performance is slightly less impressive than the two preceding algorithms. Concerning parameters, DBSCAN relies on two main ones: epsilon (ε) and min_samples.

For epsilon, small values are generally favored, and our simulation indicates that the optimal range is between 0.15 and 0.20, as illustrated by the k-distance graph. The second parameter, min_samples, typically benefits from larger values, especially for datasets with noise, as they contribute to more robust clustering.

Analyzing inter-class distance reveals similarities to K-Means clustering, with intra-class distances showing comparable patterns. Despite the presence of outliers that couldn't be clustered, an inconvenience the algorithm outperforms Affinity Propagation, particularly in terms of the expected number of clusters.

Birch: Judging by the scatter plot, Birch performed admirably, aligning closely with K-Means in terms of expected cluster numbers—parameters that were accurately set. It successfully identified all the points, and while it might not perfectly match visual inspection, it comes remarkably close, recognizing clusters like blue, orange, brown, and almost green.

Birch involves three crucial parameters. The number of clusters aligns with the K-Means discussion. The threshold determines the subcluster radius, and a moderate value, typically between 0.2 and 0.5, proves effective. The branching factor controls subclusters in each node, with a commonly used value around 50.

Assessing distances, especially intra-class in the large green cluster, reveals a notable error. However, for the remaining clusters, Birch performs well and is quite comparable to K-Means. Overall, the scatter plot indicates no outliers, with every point correctly clustered.

Optics: Upon documenting the outcome of this algorithm, it becomes apparent that it doesn't perform exceptionally well. It disregards an entire cluster as an outlier, and some points in the lower cluster face a similar fate. Optics involves two key parameters: max_eps and min_samples. Based on research and citations, considering a slightly higher value for min_samples is advisable to filter out small, isolated regions often deemed noise.

For max_eps, the default value of np.inf identifies clusters across all scales. Examining distances, particularly intra-class distance in the lone orange cluster, reveals a considerable drawback: the distance between points is large.

In essence, Optics struggles to discern patterns and cluster all the points. It even neglects an entire cluster, making it less valuable for this dataset type.

HDBSCAN: In its entirety, HDBSCAN succeeded in forming only four distinct groups—two in the upper region and two in the lower region. Regrettably, this performance is suboptimal, marked by some black points flagged as outliers.

For the parameters, min_cluster_size is crucial, representing the smallest grouping size we consider a cluster. Following the guidance from citations, we opted for a value slightly exceeding 10. The second parameter, min_samples, functions as a measure of how conservatively we want clustering to be. Based on simulations and tests, we settled on 4.

Analyzing intra-class distance, the orange cluster presents a challenge with points distantly spread. On the positive side, inter-class distances allow for clear differentiation between classes.

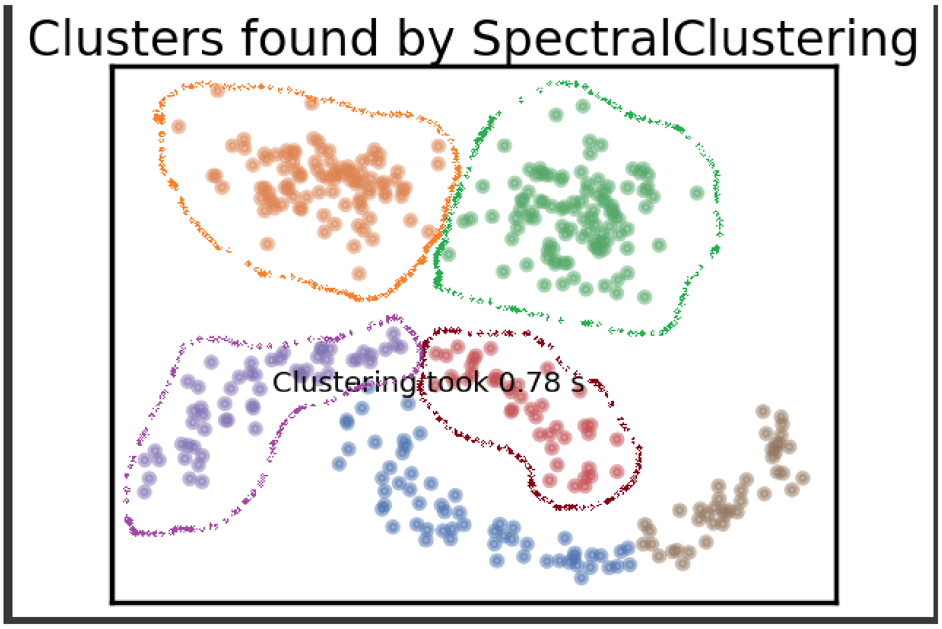
Overall, perusing the scatter plot, HDBSCAN outperforms Birch but still falls short in isolating specific points, gathering three groups into one.

Agglomerative Clustering: This algorithm did pretty well in grouping points. It aligned closely with our manual inspection, creating six clear groups without outliers.

As for the settings, we aimed for six clusters (matching our expectation) and chose 'average' for linkage, as it's generally versatile as it is shown on the notebook file, fitting our data. Distances show good separation between groups (interclass) and reasonably inside groups (intra class).

In summary, it performed admirably—no outliers, grouped all points, and mirrored our manual clustering with three similar groups, although there's room for improvement

Spectral Clustering: This algorithm stands out among the rest, showcasing remarkable performance. Upon inspecting the scatterplot, it precisely clustered four groups, closely resembling our manual clustering with only slight differences in the bottom clusters. Notably, it successfully avoided any outliers by effectively splitting all data points.



Regarding parameters, we aimed for six clusters, as expected. The 'nearest_neighbors' affinity was chosen due to its effectiveness with sparse datasets, outperforming 'rbf' and 'poly' in our tests. The gamma parameter, used with 'rbf' or 'poly' kernel.

Analyzing intra and interclass distances, classes are distinct, signifying a robust interclass separation. While there are some overlaps in intra class, they remain insignificant.

In summary, Spectral Clustering shines in this scenario, closely aligning with our manual clustering. Its ability to avoid outliers and precisely cluster groups makes it a standout performer in this type of dataset.

c. Conclusion

In this comprehensive clustering analysis, we explored various algorithms on an artificial dataset designed for research purposes. Our goal was to identify the most suitable algorithm for this unique dataset.

K-Means and *Birch* performed admirably, effectively clustering points in line with our visual inspection. *Affinity Propagation* showed potential but struggled with excessive cluster splitting. *DBSCAN* demonstrated resilience against outliers, yielding a performance comparable to K-Means. *Optics*, unfortunately, underperformed by neglecting an entire cluster.

HDBSCAN showed promise but fell short of expectations, struggling to identify clusters accurately. *Agglomerative Clustering* displayed impressive results, accurately clustering six groups, and proving versatile with different linkage methods. Notably, *Spectral Clustering* emerged as the top performer, precisely clustering groups with minimal outliers.

Considering the intra and interclass distances, Spectral Clustering demonstrated robust separation, outperforming others in distinguishing clusters effectively. The algorithm's efficiency in avoiding outliers and precisely grouping data points underscores its suitability for this unique dataset.

In conclusion, Spectral Clustering stands out as the most effective algorithm for this artificial dataset,closely mirroring our manual clustering which is at the beginning of this report and providing accurate clustering and minimizing outliers.

5) References:

[1] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[2] Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding

[3] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html

[4]http://www.sefidian.com/2022/12/18/how-to-determine-epsilon-and-minpts-parameters-of-dbscan-clustering/

[5] Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.htm

[7] Ankerst, M., Breunig, M. M., Kriegel, H. P., & Sander, J. (1999). OPTICS: Ordering Points To Identify the Clustering Structure

[8] McInnes, L., Healy, J., & Melville, J. (2017). HDBSCAN: Hierarchical density-based clustering

[9] https://hdbscan.readthedocs.io/en/latest/parameter_selection.html

[10] Athman Bouguettaya a,Andy Song,RMIT University, Australia, Efficient agglomerative hierarchical clustering,ELSEVIER.

[11] A tutorial on spectral clustering U Von Luxburg - Statistics and computing, 2007-Springer.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html