

# HW3 Report

0516034 楊翔鈞

## #1 題目

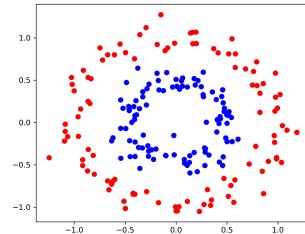
這次作業要做 clustering，我選擇的是 track 1，要實作 hierarchical clustering 與 DBSCAN，dataset 的部分是利用 sklearn.dataset 裡面的 make\_circle 與 make\_moon 來產生隨機的二維點。

## #2 實驗

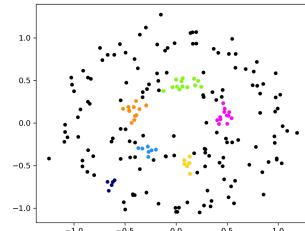
make\_circle 跟 make\_moon 有 noise 參數可以調整，我使用了三種不同的 noise : 0、0.05 與 0.1，一共產生六組 dataset，每組 200 個點。接著會說明 DBSCAN 與 hierarchical clustering 各自所做的實驗。

### #2.1 DBSCAN - circle (noise=0.1)

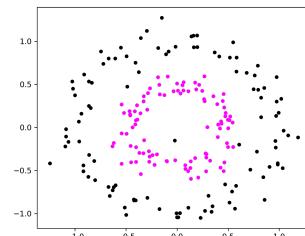
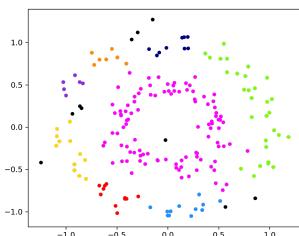
右圖為 noise=0.1 的情況下用 make\_circle 產生的點，為 2-class 的 dataset，不同顏色代表不同的 class。雖然所有點大致都以內圈、外圈的方式排列，但內、外圈在某些地方很難區分（距離相近）。



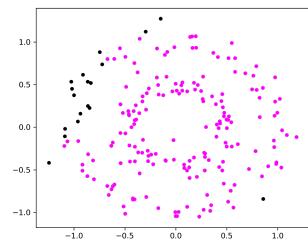
radius=0.1、minPts=5 這個組合找出了六個 cluster，並有 145 個點被歸類為 outlier（黑點），ARI 為 0.1256（右圖）。當 minPts 提高到 10 跟 15 時，200 個點全部被判定成 outlier。



當 radius=0.2、minPts=5 時，DBSCAN 找出了八個 cluster，10 個點被歸類為 outlier，此時的 ARI 為 0.5249（左下），可以發現內圈的點都被歸類成同一個 cluster 了。如果把 minPts 提高到 10 的話，只會剩下內圈一個 cluster，外圈的點因為無法達到 minPts 的門檻而全部被視為 outlier（右下）。而當 minPts 提高到 15 時，僅內圈一小部分的點組成一個 cluster，其他點都變成 outlier。

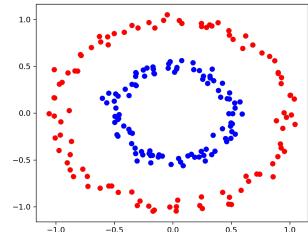


在 radius 等於 0.3 到 0.5 的這個區間，幾乎所有點都可以達到 minPts 的門檻，加上 core point 彼此間的距離都很近（小於 radius），使得 core point 彼此間都是 density-connected，所以 200 個點中只有零星幾個點被當成 outlier，其他點都被分到同一個 cluster 中，如右圖，右圖是 radius=0.3、minPts=10 的結果。

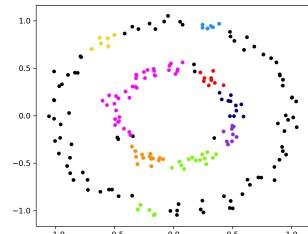


## #2.2 DBSCAN - circle (noise=0.05)

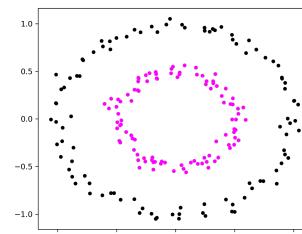
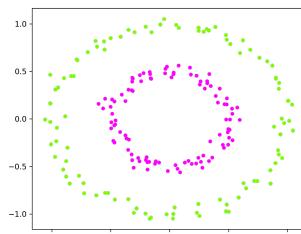
右圖為 noise=0.05 的情況下用 make\_circle 產生的點，為 2-class 的 dataset，不同顏色代表不同的 class。相較於 noise=0.1 的情況，這組的內、外圈又更接近理想的圓。



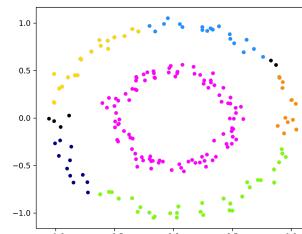
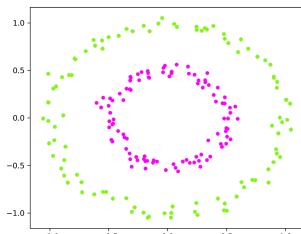
radius=0.1、minPts=5 這個組合找出了 10 個 cluster，並有 89 個點被歸類為 outlier，ARI 為 0.3778（右圖）。因為這次減少了 make\_circle 的 noise，產生的點會比較貼近圓的形狀，不像 noise=0.1 那樣鬆散，所以當 minPts 提高到 10 的時候，內圈仍有少量的點被 DBSCAN 歸類成同一個 cluster，不過當 minPts 等於 15 時，200 個點還是全被判定成 outlier。



當 radius=0.2、minPts=5 時，200 個點被精準分成兩個 cluster，此時的 ARI 為 1（左下）。但是如果把 minPts 提高到 10 的話，外圈的點因為無法達到 minPts 的門檻而被視為 outlier（右下），只剩下內圈一個 cluster。而當 minPts 提高到 15 時，僅內圈形成了兩個小 cluster。

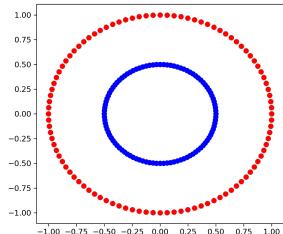


當 radius=0.3、minPts=5 的時候，200 個點又被準確分成兩個 cluster（左下）。minPts 提高到 10 的時候，外圈開始分裂成多個 cluster（右下）。當 minPts 提高到 15 時，外圈的點又被全部是 outlier 了。

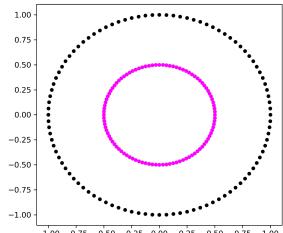


## #2.3 DBSCAN - circle (noise=0)

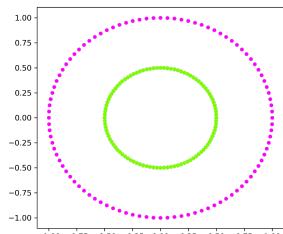
右圖為 noise=0 的情況下用 make\_circle 產生的點，為 2-class 的 dataset，不同顏色代表不同的 class。內、外圈各自是完美的圓。



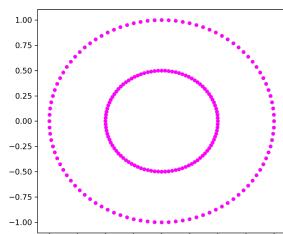
當 radius=0.1、minPts=5 時，內圈形成一個 cluster，外圈的點全被視為 outlier，ARI 為 1 (右圖)。把 minPts 提升到 10 跟 15 後，內圈的點也無法超過 minPts 的門檻了，200 個點全被視為 outlier。



當 radius=0.2、0.3、0.4 與 minPts=5 這三種情況下，200 個點皆被精準分成兩個 cluster (右圖)，而跟之前的實驗一樣，隨著 minPts 上升到 10 與 15，外圈首先被視為 outlier，接著是內圈。

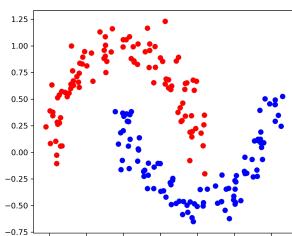


當 radius=0.5 時，三種 minPts 跑出來的結果都一樣 (右圖)，因為 radius 太大了所以所有點都被歸類在同一個 cluster 內。

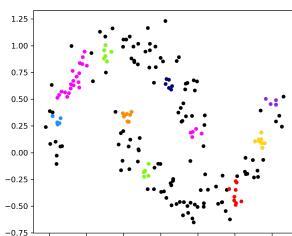


## #2.4 DBSCAN - moon (noise=0.1)

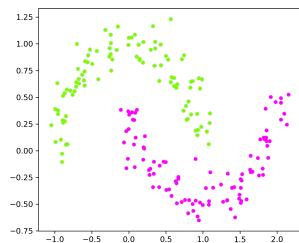
右圖為 noise=0.1 的情況下用 make\_moon 產生的點，為 2-class 的 dataset，不同顏色代表不同的 class。



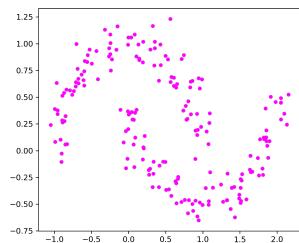
當 radius=0.1、minPts=5 時，DBSCAN 找出了 10 個 cluster，並有 122 個點被歸類為 outlier，ARI 為 0.0358 (右圖)。當 minPts 提高到 10 跟 15 時，200 個點全部被判定成 outlier。



當  $\text{radius}=0.3$  與  $\text{minPts}=10, 15$  這兩種情況下，200 個點被分成兩個 cluster（右圖），僅有少數幾個點的 class 與 ground truth 不一樣，此時的 ARI 為 0.9602。

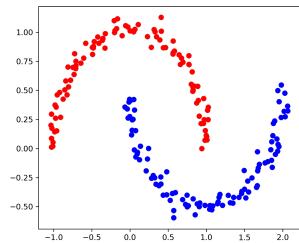


當  $\text{radius}$  超過 0.5 時，所有點都被分在同一個 cluster 裡（右圖）。

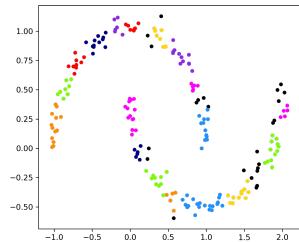


## #2.5 DBSCAN - moon (noise=0.05)

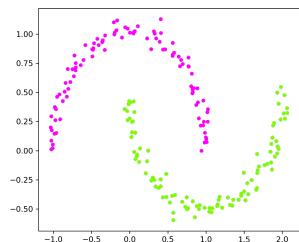
右圖為  $\text{noise}=0.05$  的情況下用 `make_circle` 產生的點，為 2-class 的 dataset，不同顏色代表不同的 class。兩個 class 的排列越來越接近弧線。



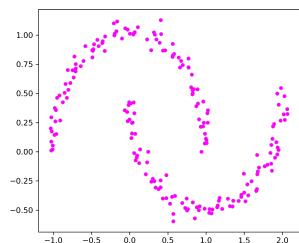
當  $\text{radius}=0.1$ 、 $\text{minPts}=5$  時，有 18 個 cluster 與 24 個 outlier，ARI 為 0.0954（右圖）。把  $\text{minPts}$  提升到 10 跟 15 後，因為沒有點能滿足  $\text{minPts}$  的門檻，200 個點全都被視為 outlier。



當  $\text{radius}=0.3$  與  $\text{minPts}=5, 10, 15$  這三種情況下，所有點都被精準地分成兩堆，而且得到的 class 跟 ground truth 一樣，此時 ARI 為 1。

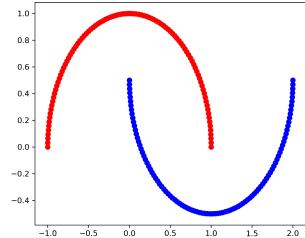


就像上面  $\text{noise}=0.1$  的實驗結果一樣，當  $\text{radius}$  超過 0.5 時，所有點都被分在同一個 cluster 裡（右圖）。

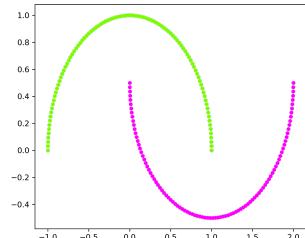


## #2.6 DBSCAN - moon (noise=0)

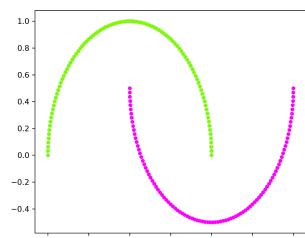
右圖為 noise=0 的情況下用 make\_circle 產生的點，為 2-class 的 dataset，不同顏色代表不同的 class。兩個 class 各自以弧線的方式排列。



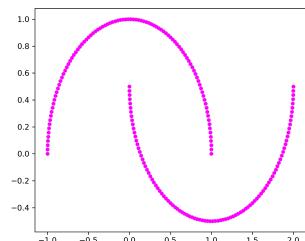
當 radius=0.1、minPts=5 時，200 個點皆被正確分類，ARI 為 1 (右圖)，不過當 minPts 上升至 10 與 15 時，因為無法滿足 minPts，所以所有點都被歸類成 outlier。



當 radius=0.3 與 0.5 時，三種 minPts (5, 10, 15) 得到的結果都一樣，所有點都被正確分類 (右圖)。



當 radius 超過 0.7 時，三種 minPts 跑出來的結果都一樣 (右圖)，因為 radius 太大了所以所有點都被歸類在同一個 cluster 內。



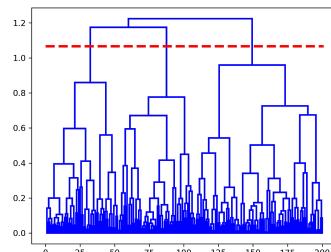
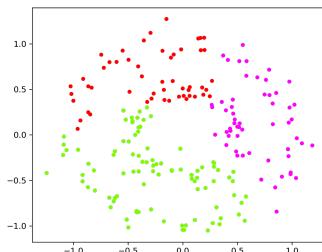
## #2.7 Hierarchical clustering - circle (noise=0.1)

這邊使用的 dataset 跟 DBSCAN 的相同，所以就不放 ground truth 的圖片了，另外採用的距離計算方式是 Euclidean distance。

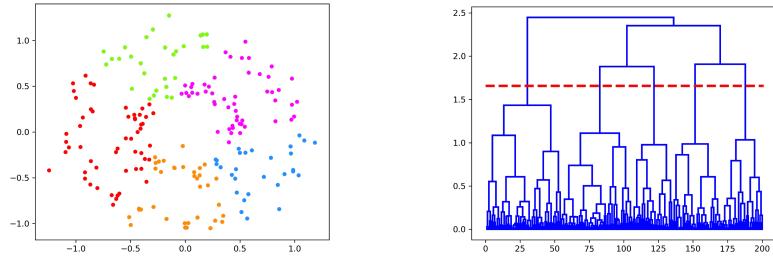
dendrogram 中紅線的決定方式有以下兩個條件，紅線就是在符合這兩個條件下，將兩條水平線取平均得到的結果：

- 任意兩條水平線所圍成的區域內只能有垂直的藍線
- 兩條水平線的距離越大越好

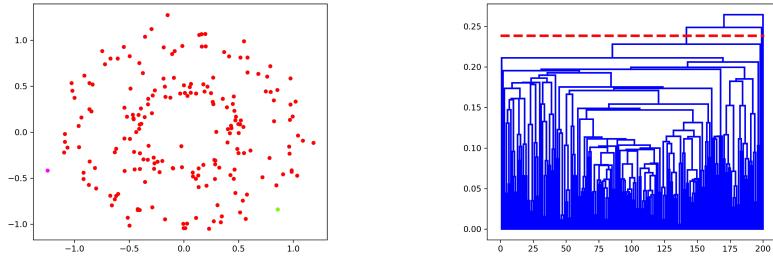
下面為採用 average linkage 的 cluster 結果，最佳的 cluster 數量是三個，200 個點被分成三個扇形區域，ARI 為 0.0084。



下面為採用 complete linkage 的 cluster 結果，dataset 同樣以扇形的方式被分群，不過這次的最佳 cluster 數量是五個，ARI 為 0.0087。

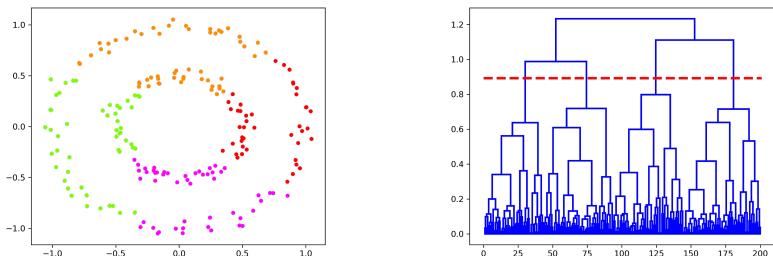


下面為採用 single linkage 的 cluster 結果，最佳 cluster 數量是三個，不過幾乎所有點都落在其中一個 cluster 裡面，ARI 為 0.0001。

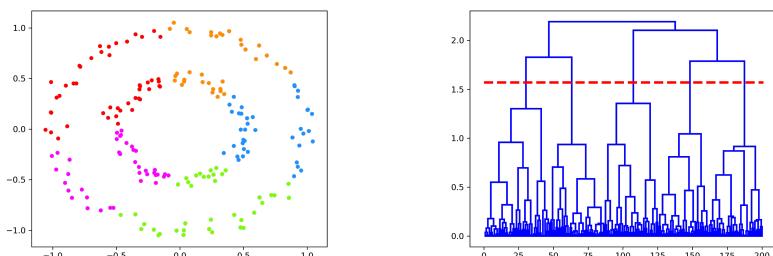


## #2.8 Hierarchical clustering - circle (noise=0.05)

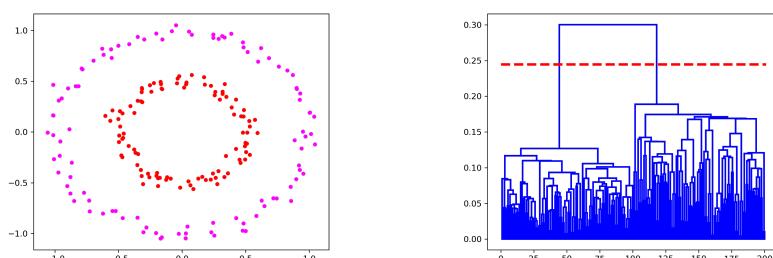
下面為採用 average linkage 的 cluster 結果，最佳的 cluster 數量是四個，200 個點扇形的方式被分區，ARI 為 -0.004。



下面為採用 complete linkage 的 cluster 結果，最佳 cluster 數量是五個，200 個點也是以類似扇形的方式被分群，ARI 為 0.006。

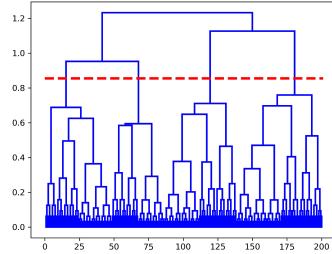
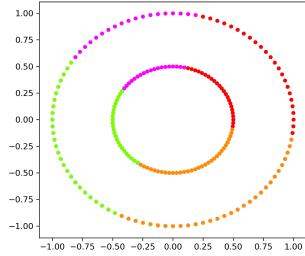


下面為採用 single linkage 的 cluster 結果，分類結果跟 ground truth 一樣，ARI 為 1。

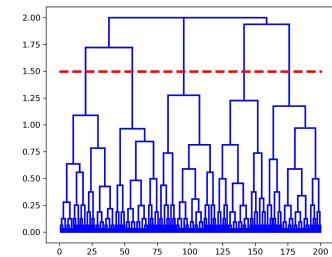
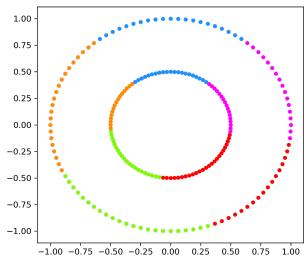


## #2.9 Hierarchical clustering - circle (noise=0)

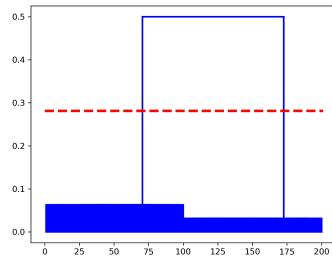
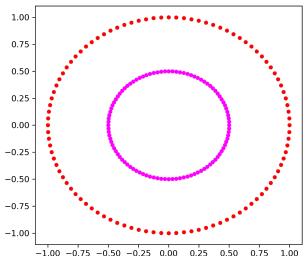
下面為採用 average linkage 的 cluster 結果，最佳的 cluster 數量是四個，200 個點被分成四個扇形區域，ARI 為 -0.007。



下面為採用 complete linkage 的 cluster 結果，200 個點以扇形的方式被分群，最佳 cluster 數量為五個，ARI 為 -0.0028。

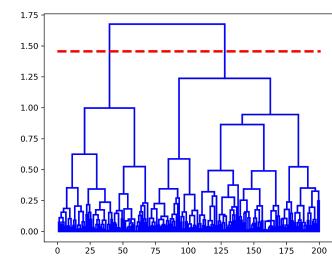
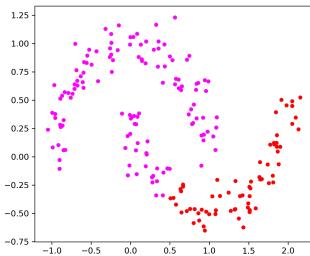


下面為採用 single linkage 的 cluster 結果，最佳 cluster 數量是兩個，分類結果跟 ground truth 一樣，ARI 為 1。

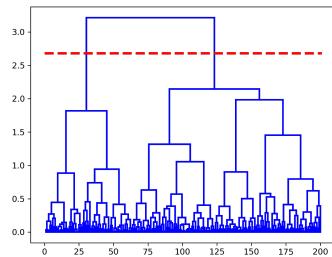
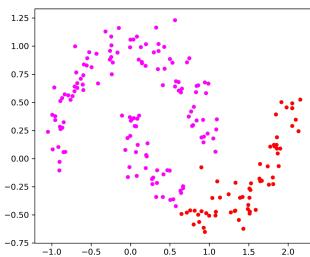


## #2.10 Hierarchical clustering - moon (noise=0.1)

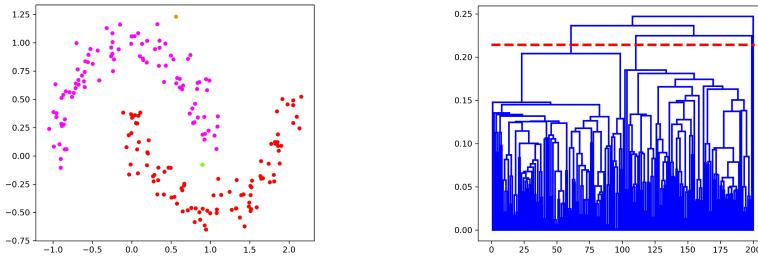
下面為採用 average linkage 的 cluster 結果，最佳的 cluster 數量是兩個，凹向下的弧形被正確分類，凹向上的弧形有大約一半的點被錯誤分類，ARI 為 0.4464。



下面為採用 complete linkage 的 cluster 結果，最佳的 cluster 數量是兩個，跟average linkage 一樣，凹向上的弧形有一半的點被錯誤分類，ARI 為 0.3453。

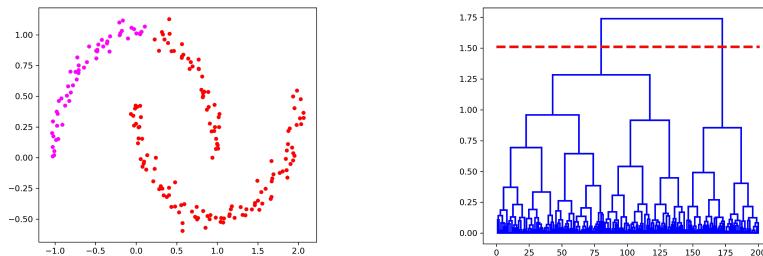


下面為採用 single linkage 的 cluster 結果，最佳 cluster 數量是四個，不過幾乎大部分的點都被分在粉紅色與紅色兩個 cluster 內，其餘兩個 cluster 只有少數幾個點，ARI 為 0.9604。

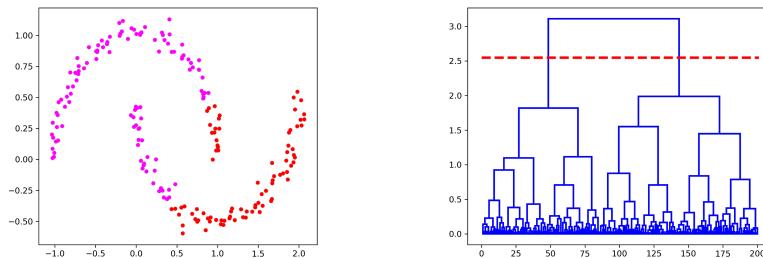


## #2.11 Hierarchical clustering - moon (noise=0.05)

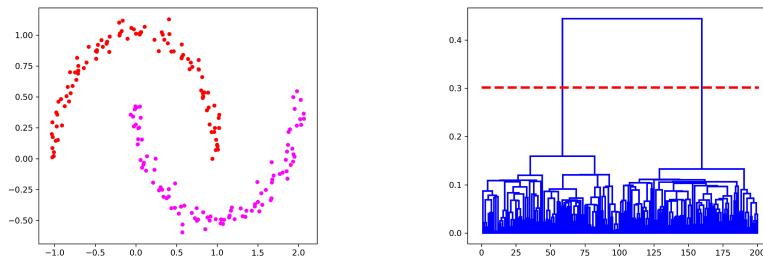
下面為採用 average linkage 的 cluster 結果，最佳的 cluster 數量是兩個，跟noise=0.1 的情況類似，其中一個弧形有一半的點被錯誤分類，ARI 為 0.3108。



下面為採用 complete linkage 的 cluster 結果，最佳 cluster 數量是兩個，兩個弧形中都有某些點被錯誤分類，ARI 為 0.2774。

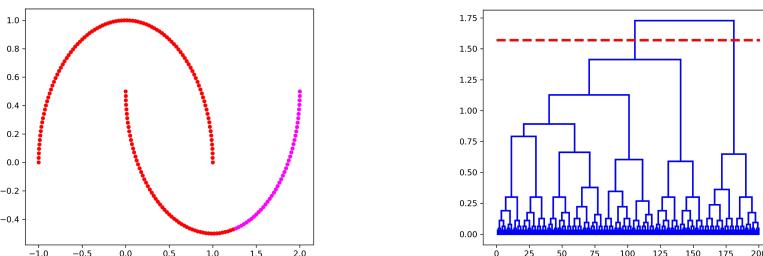


下面為採用 single linkage 的 cluster 結果，跟 ground truth 一樣，ARI 為 1。



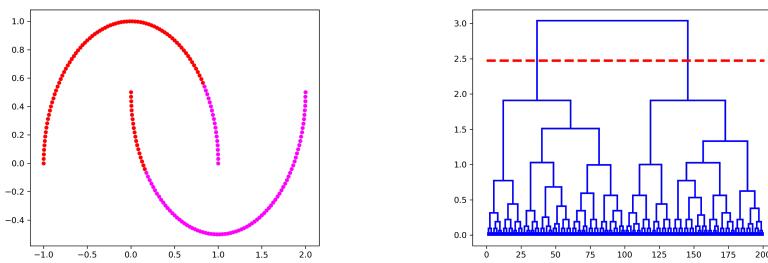
## #2.12 Hierarchical clustering - moon (noise=0)

下面為採用 average linkage 的 cluster 結果，最佳的 cluster 數量是兩個，同樣也出現其中一個弧形中的部分點被錯誤分類的情況，ARI 為 0.1736。

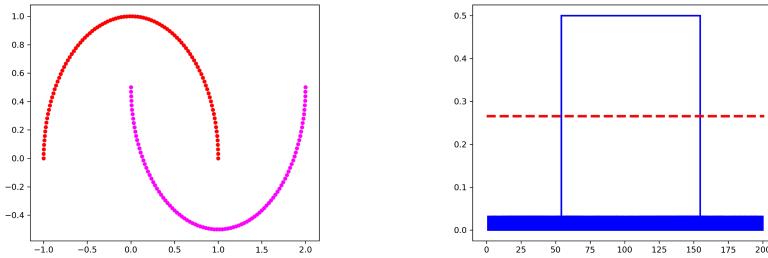


下面為採用 complete linkage 的 cluster 結果，最佳 cluster 數量是兩個，分類的結果跟 noise=0.05 的 complete linkage 類似，兩個弧形都有部分點被錯誤分類，ARI 為 0.3813

◦

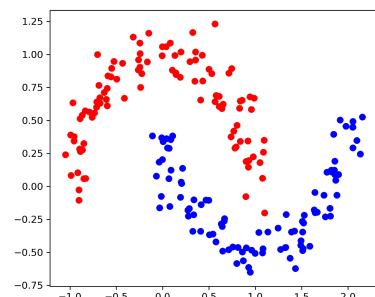


下面為採用 single linkage 的 cluster 結果，最佳 cluster 數量是兩個，分類結果跟 ground truth 一樣，ARI 為 1。



### #3 觀察

- Hierarchical clustering 中三種 linkage 的比較：從實驗結果可以發現 single linkage 會傾向產生鏈狀的 cluster，因為 single linkage 實際上跟 minimum spanning tree 的概念一樣，不斷尋找尚未合併且相距最短的兩個 set，而 complete linkage 則傾向把資料分成一落一落的，這部分也可以從兩者的 dendrogram 觀察出來，single linkage 會不斷地把已存在的 cluster 與還未屬於任何 cluster 的資料點合併，不斷地將 cluster 延長，而 complete linkage 則會盡量將資料點兩兩合併，再去將 cluster 兩兩合併，至於 average linkage 我看不出明顯的規律，不過感覺跟 complete linkage 比較相似。
- 產生 dataset 時選擇的 noise：我認為 DBSCAN 比較容易受到這個影響，因為做 DBSCAN 的時候需要給定 radius 與 minPts，如果資料分布太雜亂會導致無法準確判斷出 core point，而 hierarchical clustering 就比較不受影響，只要還看得出資料大致的走向，single linkage 就還是能夠把資料正確分群，以 noise 等於 0.1 的 moon 舉例（右圖），因為兩個弧形在某些區域的點非常靠近，如果 DBSCAN 沒有選擇適當的 radius 以及 minPts，很容易把兩個弧形歸類成一整個群，但是 single linkage 却依舊可以準確地把兩個弧形分開。
- 執行速度：這部分我沒有寫在上面的實驗數據裡，只是自己在跑實驗的觀察，完全沒有做 optimize 的 hierarchical clustering 的時間複雜度是  $O(n^3)$ ，當 dataset 的大小超過 250 個點的時候就會花費很久的時間，但是因為 single linkage 跟 minimum spanning tree 的運作方式其實是一樣的，經過修改後可以把時間複雜度降低到  $O(n^2)$ ，後來跑 1500 個點也只需要五分鐘，而 complete linkage 與 average linkage 同樣也可以利用 heap 去 optimize，average linkage 的我想了很久還是不知道怎麼把資料結構實作出來，所以只有修改 complete linkage，而修改後的複雜度是  $O(n^2 \log n)$ ，對 1500 個點做 clustering 只需要 14 分鐘（原本跑好幾個小時都還跑不出來）。



```

import numpy as np
from numpy.linalg import norm
from datetime import datetime
import matplotlib.pyplot as plt
from queue import Queue, PriorityQueue
from scipy.spatial.distance import cdist
from sklearn.metrics import adjusted_rand_score

CORE = 100
EDGE = 200
INF = 1e5
ENCODE = 1
MAPPING = {}
THRES = []
VERTICAL = []

def dbscan(points, radius, density_thres):
    point_num = points.shape[0]
    point_labels = np.zeros(points.shape[0])
    point_neighbors = []
    core = []
    non_core = []

    for i in range(points.shape[0]):
        neighbors = [j for j in range(points.shape[0]) if
norm(points[i]-points[j])<=radius]
        if len(neighbors) >= density_thres:
            point_labels[i] = CORE
            core.append(i)
        else:
            non_core.append(i)
        point_neighbors.append(neighbors)

    for i in non_core:
        for neighbor in point_neighbors[i]:
            if neighbor in core:
                point_labels[i] = EDGE
                break

    cluster_idx = 1
    for i, label in enumerate(point_labels):
        q = Queue()
        if label == CORE:
            q.put(i)
            while not q.empty():
                for neighbor in point_neighbors[q.get()]:
                    if point_labels[neighbor] in [CORE, EDGE]:
                        if point_labels[neighbor] == CORE:
                            q.put(neighbor)
                        point_labels[neighbor] = cluster_idx
            cluster_idx += 1

```

```

        cluster_idx += 1

    return point_labels

def plot_clusters(points, point_labels, cluster_idx, file=None):
    colors = [
        'red', 'magenta', 'lawngreen', 'darkorange', 'dodgerblue',
        'gold', 'navy', 'blueviolet',
    ]
    if cluster_idx is None:
        clusters = np.unique(point_labels).astype(np.int32)
        labels = np.asarray(point_labels)
        has_outlier = 0
        for idx in clusters:
            if idx == 0:
                color = 'black'
                has_outlier = 1
            else:
                color = colors[idx%8]
            c_i = points[labels==idx]
            plt.scatter(c_i[:, 0], c_i[:, 1], c=color, s=15)
        cluster_num = clusters.shape[0] - has_outlier
        if file is not None:
            print(f'{cluster_num} clusters found', file=file)
            print(f'{np.sum(labels==0)} outliers', file=file)
        else:
            print(f'{cluster_num} clusters found')
            print(f'{np.sum(labels==0)} outliers')
    else:
        for i, c in enumerate(cluster_idx):
            color = colors[i%8]
            c_i = points[np.asarray(c)]
            plt.scatter(c_i[:, 0], c_i[:, 1], c=color, s=15)

def get_members(cluster):
    q = Queue()
    q.put(cluster)
    member = []
    while not q.empty():
        c = q.get()
        for m in c:
            if type(m) == list:
                q.put(m)
            elif type(m) == tuple:
                member.append(m[0])
    return member

def single_link(cluster_1, cluster_2, dis_mat):

```

```

members_c1 = get_members(cluster_1)
members_c2 = get_members(cluster_2)
min_dis = INF
for m_c1 in members_c1:
    for m_c2 in members_c2:
        dis = dis_mat[m_c1, m_c2]
        min_dis = min(dis, min_dis)
return min_dis

def complete_link(cluster_1, cluster_2, dis_mat):
    members_c1 = get_members(cluster_1)
    members_c2 = get_members(cluster_2)
    max_dis = 0
    for m_c1 in members_c1:
        for m_c2 in members_c2:
            dis = dis_mat[m_c1, m_c2]
            max_dis = max(dis, max_dis)
    return max_dis

def average_link(cluster_1, cluster_2, dis_mat):
    members_c1 = get_members(cluster_1)
    members_c2 = get_members(cluster_2)
    dis = 0
    for m_c1 in members_c1:
        for m_c2 in members_c2:
            dis += dis_mat[m_c1, m_c2]
    return dis / (len(members_c1)*len(members_c2))

def distance(cluster_1, cluster_2, dis_mat, method):
    if method == 'single':
        return single_link(cluster_1, cluster_2, dis_mat)
    elif method == 'complete':
        return complete_link(cluster_1, cluster_2, dis_mat)
    elif method == 'average':
        return average_link(cluster_1, cluster_2, dis_mat)

def get_mis_dis(clusters, dis_mat, method):
    n = len(clusters)
    min_dis = INF
    min_dis_pair = None
    for i in range(n):
        for j in range(i, n):
            if i == j:
                continue
            else:
                dis = distance(clusters[i], clusters[j], dis_mat,
method)

```

```

        if dis < min_dis:
            min_dis = dis
            min_dis_pair = (i, j)
    return min_dis_pair, min_dis

def clustering(clusters, dis_mat, method):
    global THRES, VERTICAL
    merge_pair, min_dis = get_mis_dis(clusters, dis_mat, method)
    THRES.append(min_dis)
    VERTICAL.append((min_dis, clusters[merge_pair[0]][0][-1]))
    VERTICAL.append((min_dis, clusters[merge_pair[1]][0][-1]))
    target = clusters.pop(merge_pair[1])
    clusters[merge_pair[0]].append(target[0])
    clusters[merge_pair[0]] = [[clusters[merge_pair[0]], min_dis]]


def hc(clusters, dis_mat, method):
    cluster_num = len(clusters)
    while cluster_num > 1:
        clustering(clusters, dis_mat, method)
        cluster_num = len(clusters)
    return clusters


def data_mapping(clusters):
    global ENCODE, MAPPING
    _clusters = clusters[0]
    if type(_clusters) == list:
        data_mapping(_clusters[0])
        data_mapping(_clusters[1])
    else:
        MAPPING[str(_clusters[0])] = ENCODE
        ENCODE += 1


def get_center(clusters):
    _clusters = clusters[0]
    if type(_clusters) == list:
        left_center = get_center(_clusters[0])
        right_center = get_center(_clusters[1])
        return (left_center+right_center) / 2
    else:
        return MAPPING[str(_clusters[0])]


def draw_dendrogram(clusters):
    _clusters = clusters[0]
    if type(_clusters) == list:
        left_center = get_center(_clusters[0])
        right_center = get_center(_clusters[1])

```

```

        center = (left_center+right_center) / 2
        plt.plot([left_center, right_center], [clusters[1],
clusters[1]], c='blue', lw=2)
        plt.plot([left_center, left_center], [_clusters[0][1],
clusters[1]], c='blue', lw=2)
        plt.plot([right_center, right_center], [_clusters[1][1],
clusters[1]], c='blue', lw=2)
        draw_dendrogram(_clusters[0])
        draw_dendrogram(_clusters[1])

def get_opt_thres():
    global THRES
    THRES.append(0)
    THRES = sorted(THRES, reverse=True)
    max_gap = 0
    opt_thres = None
    for i in range(1, len(THRES)):
        gap = THRES[i-1] - THRES[i]
        if gap > max_gap:
            max_gap = gap
            opt_thres = (THRES[i-1]+THRES[i]) / 2
    return opt_thres

def get_opt_clusters(clusters_idx, clusters, thres):
    _clusters = clusters[0]
    if type(_clusters) == list:
        upper_bound = clusters[1]
        left_lower_bound = _clusters[0][1]
        right_lower_bound = _clusters[1][1]
        if thres<upper_bound and thres>left_lower_bound:
            clusters_idx.append(get_members(_clusters[0]))
        else:
            get_opt_clusters(clusters_idx, _clusters[0], thres)
    if thres<upper_bound and thres>right_lower_bound:
        clusters_idx.append(get_members(_clusters[1]))
    else:
        get_opt_clusters(clusters_idx, _clusters[1], thres)

def find_leader(clusters, idx):
    leader = idx
    while True:
        if type(clusters[leader]) == int:
            leader = clusters[leader]
        else:
            return leader

def mst(clusters, dis_mat):

```

```

global THRES, VERTICAL
q = PriorityQueue()
for i in range(0, dis_mat.shape[0]):
    for j in range(i+1, dis_mat.shape[1]):
        q.put((dis_mat[i, j], (i, j)))
cluster_num = len(clusters)
label = np.arange(cluster_num)
while cluster_num > 1:
    merge_pair = q.get()
    dis = merge_pair[0]
    pair = merge_pair[1]
    leader_0 = find_leader(clusters, pair[0])
    leader_1 = find_leader(clusters, pair[1])
    if label[leader_0] == label[leader_1]:
        continue
    label[leader_1] = leader_0
    THRES.append(dis)
    VERTICAL.append((dis, clusters[leader_0][0][-1]))
    VERTICAL.append((dis, clusters[leader_1][0][-1]))
    clusters[leader_0].append(clusters[leader_1][0])
    clusters[leader_0] = [[clusters[leader_0], dis]]
    clusters[leader_1] = leader_0
    cluster_num -= 1
for t in clusters:
    if type(t) != int:
        return [t]

def complete_hc(clusters, dis_mat):
    global THRES, VERTICAL
    dis_list = []
    edge_map = {}
    for i in range(dis_mat.shape[0]):
        for j in range(i, dis_mat.shape[1]):
            if i == j:
                continue
            dis_list.append((dis_mat[i, j], (i, j)))
            edge_map[(i, j)] = True
            edge_map[(j, i)] = True
    dis_list = sorted(dis_list, key=lambda t: t[0], reverse=True)
    cluster_num = len(clusters)
    label = np.arange(cluster_num)
    while cluster_num > 1:
        (dis, pair) = dis_list.pop()
        if not edge_map[pair]:
            continue
        leader_0 = find_leader(clusters, pair[0])
        leader_1 = find_leader(clusters, pair[1])
        if label[leader_0] == label[leader_1]:
            continue
        label[leader_1] = leader_0

```

```

        THRES.append(dis)
        VERTICAL.append((dis, clusters[leader_0][0][-1]))
        VERTICAL.append((dis, clusters[leader_1][0][-1]))
        clusters[leader_0].append(clusters[leader_1][0])
        clusters[leader_0] = [[clusters[leader_0], dis]]
        clusters[leader_1] = leader_0
        cluster_num -= 1

    merge_members = get_members(clusters[leader_0])
    other_clusters = []
    check = {}
    for i in range(len(label)):
        leader = find_leader(clusters, i)
        if leader_0!=leader and leader_1!=leader:
            if check.get(leader):
                continue
            other_clusters.append(get_members(clusters[leader]))
            check[i] = True
    for cluster_members in other_clusters:
        dis_block = dis_mat[np.array(cluster_members), :]
        dis_block = dis_block[:, np.array(merge_members)]
        max_pos = np.unravel_index(np.argmax(dis_block),
dis_block.shape)
        for m1 in cluster_members:
            for m2 in merge_members:
                edge_map[(m1, m2)] = False
                edge_map[(m2, m1)] = False
                edge_map[(cluster_members[max_pos[0]], merge_members[max_pos[1]]]] = True
                edge_map[(merge_members[max_pos[1]], cluster_members[max_pos[0]]]] = True
        for t in clusters:
            if type(t) != int:
                return [t]

def plot_ground_truth(points, y):
    colors = [
        'red', 'blue', 'gold', 'green', 'purple'
    ]
    labels = np.unique(y)
    for l in labels:
        p = points[y==l]
        plt.scatter(p[:, 0], p[:, 1], c=colors[l%5])

def close_fig(filename=None):
    plt.tight_layout()
    if filename is None:
        plt.show()
    else:

```

```

        plt.savefig(filename, dpi=300, transparent=True)
        plt.clf()

def initialize_hc():
    global ENCODE, MAPPING, THRES, VERTICAL
    ENCODE = 1
    MAPPING = {}
    THRES = []
    VERTICAL = []

if __name__ == '__main__':
    DBSCAN = True
    HC = False
    if DBSCAN:
        for shape in ['circle', 'moon', 'varied']:
            for point_num in [100, 200]:
                if shape=='circle' or shape=='moon':
                    for noise in [0, 0.05, 0.1]:
                        points =
    np.load(f'./data/{shape}_{point_num}_{noise}.npy')
    y =
    np.load(f'./data/{shape}_{point_num}_{noise}_y.npy')
    plot_ground_truth(points, y)

close_fig(f'./dbscan/{shape}_{point_num}_{noise}_gt.png')
    if shape == 'circle':
        r = [0.1, 0.2, 0.3, 0.4, 0.5]
    else:
        r = [0.1, 0.3, 0.5, 0.7]
    for radius in r:
        for density_thres in [5, 10, 15]:
            start = datetime.now()
            log = open('dbscan.log', 'a+')
            print(f'{shape}, {point_num} data
points, noise={noise}, radius={radius}, density_thres={density_thres}', file=log)

            point_labels = dbscan(points, radius,
density_thres)
            plot_clusters(points, point_labels,
None, log)

close_fig(f'./dbscan/{shape}_{point_num}_{noise}_{radius}_{density_thres}.png')

        y_pred = np.arange(point_num)
        for i, l in enumerate(point_labels):
            y_pred[i] = -1 if l==0 else l - 1
        ari_score = adjusted_rand_score(y,

```

```

y_pred)

                print(f'ARI score: {ari_score}',

file=log)
                print(f'spend {datetime.now() - start}',

file=log)
                print('=====', file=log)
log.close()

else:
    points = np.load(f'./data/{shape}_{point_num}.npy')
    y = np.load(f'./data/{shape}_{point_num}_y.npy')
    plot_ground_truth(points, y)
    close_fig(f'./dbSCAN/{shape}_{point_num}_gt.png')
    for radius in [1, 2, 3, 4]:
        for density_thres in [5, 10, 15]:
            start = datetime.now()
            log = open('dbSCAN.log', 'a+')
            print(f'{shape}, {point_num} data points,
radius={radius}, density_thres={density_thres}', file=log)
            point_labels = dbSCAN(points, radius,
density_thres)
            plot_clusters(points, point_labels, None,
log)

close_fig(f'./dbSCAN/{shape}_{point_num}_{radius}_{density_thres}.png')

y_pred = np.arange(point_num)
for i, l in enumerate(point_labels):
    y_pred[i] = -1 if l==0 else l - 1
ari_score = adjusted_rand_score(y, y_pred)

print(f'ARI score: {ari_score}', file=log)
print(f'spend {datetime.now() - start}',

file=log)
print('=====', file=log)
log.close()

if HC:
    for shape in ['circle', 'moon', 'varied']:
        for point_num in [100, 200]:
            if shape=='circle' or shape=='moon':
                for noise in [0, 0.05, 0.1]:
                    points =
np.load(f'./data/{shape}_{point_num}_{noise}.npy')
                    y =
np.load(f'./data/{shape}_{point_num}_{noise}_y.npy')
                    plot_ground_truth(points, y)

close_fig(f'./{shape}/{shape}_{point_num}_{noise}_gt.png')
                    dis_mat = cdist(points, points)
                    for method in ['single', 'complete', 'average']:
                        initialize_hc()

```

```

        clusters = [[[i, points[i].tolist()), 0]]
for i in range(points.shape[0])]:
    start = datetime.now()
    if method == 'single':
        clusters = mst(clusters, dis_mat)
    elif method == 'complete':
        clusters = complete_hc(clusters,
dis_mat)
    else:
        clusters = hc(clusters, dis_mat, method)
data_mapping(clusters[0][0])

        opt_thres = get_opt_thres()
        draw_dendrogram(clusters[0][0])
        plt.plot([0, points.shape[0]+1], [opt_thres,
opt_thres], c='red', ls='--', lw=3)

close_fig(f'./{shape}/{method}_{shape}_{point_num}_{noise}_dendrogram.png')
')

        opt_cluster_num = sum([1 if v[0]>opt_thres
and v[1]<opt_thres else 0 for v in VERTICAL])
        clusters_idx = []
        get_opt_clusters(clusters_idx,
clusters[0][0], opt_thres)
        plot_clusters(points, None, clusters_idx)

close_fig(f'./{shape}/{method}_{shape}_{point_num}_{noise}.png')

        y_pred = np.zeros(point_num, dtype=np.int32)
        for i, cluster in enumerate(clusters_idx):
            for member in cluster:
                y_pred[member] = i
        ari_score = adjusted_rand_score(y, y_pred)

        log = open(f'{shape}.log', 'a+')
        print(f'{shape}, {point_num} data points,
noise={noise}, {method} link', file=log)
        print(f'{opt_cluster_num} clusters found',
file=log)
        print(f'ARI score: {ari_score}', file=log)
        print(f'spend {datetime.now() - start}', file=log)
        print('=====', file=log)
        log.close()

else:
    points = np.load(f'./data/{shape}_{point_num}.npy')
    y = np.load(f'./data/{shape}_{point_num}_y.npy')
    plot_ground_truth(points, y)
    close_fig(f'./{shape}/{shape}_{point_num}_gt.png')
    dis_mat = cdist(points, points)

```

```

        for method in ['single', 'complete', 'average']:
            initialize_hc()
            clusters = [[[i, points[i].tolist()], 0]] for i
in range(points.shape[0])]
            start = datetime.now()
            if method == 'single':
                clusters = mst(clusters, dis_mat)
            elif method == 'complete':
                clusters = complete_hc(clusters, dis_mat)
            else:
                clusters = hc(clusters, dis_mat, method)
            data_mapping(clusters[0][0])

            opt_thres = get_opt_thres()
            draw_dendrogram(clusters[0][0])
            plt.plot([0, points.shape[0]+1], [opt_thres,
opt_thres], c='red', ls='--', lw=3)

close_fig(f'./{shape}/{method}_{shape}_{point_num}_dendrogram.png')

        opt_cluster_num = sum([1 if v[0]>opt_thres and
v[1]<opt_thres else 0 for v in VERTICAL])
        clusters_idx = []
        get_opt_clusters(clusters_idx, clusters[0][0],
opt_thres)
        plot_clusters(points, None, clusters_idx)

close_fig(f'./{shape}/{method}_{shape}_{point_num}.png')

        y_pred = np.zeros(point_num, dtype=np.int32)
        for i, cluster in enumerate(clusters_idx):
            for member in cluster:
                y_pred[member] = i
        ari_score = adjusted_rand_score(y, y_pred)

        log = open(f'{shape}.log', 'a+')
        print(f'{shape}, {point_num} data points,
{method} link', file=log)
        print(f'{opt_cluster_num} clusters found',
file=log)
        print(f'ARI score: {ari_score}', file=log)
        print(f'spend {datetime.now() - start}',
file=log)
        print('=====', file=log)
        log.close()

```

