巨量資料 HW3

108062566 資工碩一 陳法佑

一、演算法

```python
def get_init_centroids(file_path):
    centroids = []
    with open(file_path, "r") as file:
        for line in file.readlines():
            centroid = list(map(float, line.split(' ')))
            centroids.append(centroid)
    return centroids


def classify_and_get_loss(point, centroids, method):
    min_loss = float("inf")
    in_cluster = None
    point = list(map(float, point.split(' ')))
    for centroid_idx, centroid in list(enumerate(centroids)):
        point_loss = 0
        for dim in range(len(point)):
            if method == 'Euclidean':
                loss = math.pow(point[dim] - centroid[dim], 2)
            elif method == 'Manhattan':
                loss = abs(point[dim] - centroid[dim])
            else:
                raise NameError
            point_loss += loss
        # if method == 'Euclidean':
        #     point_loss = math.sqrt(point_loss)
        if point_loss < min_loss:
            in_cluster = centroid_idx
            min_loss = point_loss
    return [(in_cluster, (min_loss, point))]


def list_add_reducer(x, y):
    list_add = [0] * len(x)
    for i in range(len(x)):
        list_add[i] = x[i] + y[i]
    return list_add
```

```python
def iterative_k_means(sc,  max_iter, init_centroids_file_path, data_file_path, method):
    centroids = get_init_centroids(init_centroids_file_path)
    losses = []
    for iter in range(1, max_iter + 2):
        print('iteration: {}'.format(iter))
        clusters_with_loss_and_points = sc.textFile(data_file_path).flatMap(
            lambda x: classify_and_get_loss(x, centroids, method)
        )
        clusters_with_loss = clusters_with_loss_and_points.map(lambda x: (x[0], x[1][0]))
        loss = sum(clusters_with_loss.reduceByKey(lambda x, y: x + y).values().collect())
        print('\tloss: {}'.format(loss))
        losses.append(loss)
        if iter == max_iter + 1:
            break
        clusters_with_points = clusters_with_loss_and_points.map(lambda x: (x[0], x[1][1]))
        clusters_points_sum = clusters_with_points.reduceByKey(list_add_reducer).collect()
        clusters_points_num_dict = dict(clusters_with_points.countByKey())
        for cluster_idx, cluster_points_sum in clusters_points_sum:
            cluster_points_num = clusters_points_num_dict[cluster_idx]
            centroids[cluster_idx] = [x / cluster_points_num for x in cluster_points_sum]
    improvement = (losses[0] - losses[-1]) / losses[0] * 100.
    losses = losses[1:]
    return losses, centroids, improvement
```

Algorithm 1 Iterative k-Means Algorithm

```
1: procedure ITERATIVE k-MEANS
2:     Select k points as initial centroids of the k clusters.
3:     for iterations := 1 to MAX_ITER do
4:         for each point p in the dataset do
5:             Assign point p to the cluster with the closest centroid
6:         end for
7:         for each cluster c do
8:             Recompute the centroid of c as the mean of all the data points assigned to c
9:         end for
10:    end for
11: end procedure
```

參考講義上的演算法進行實作，流程如下：

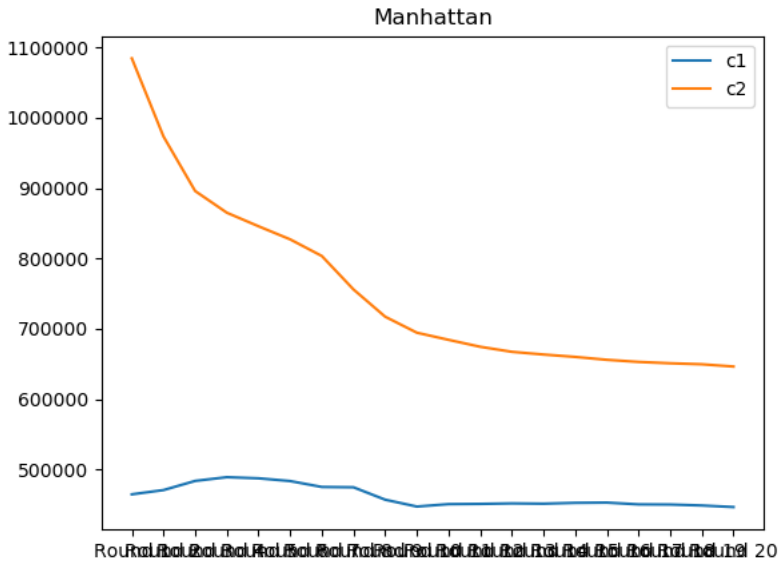
1. 從 “data.txt” 讀取所有點，並在此時計算每個點歸類到哪個 cluster 同時保存 loss，形式為 (in_cluster, (min_loss, point))，其中 in_cluster 表示此點被歸類在哪一個 cluster，min_loss 表示其與該 cluster 的 loss，point 表示此點的座標。

2. 將在同一個 cluster 的 points reduce，把他們的座標相加之後除以在同一個 cluster 裡的 points 總數（在實作中為 “clusters_with_points.countByKey()”），算出來的 list 便是各個 centroids 的新的座標，接著便回到第一步重複執行步驟。值得注意的是，實作中讓迴圈跑了 21 次，因為第一次是 initial（centroids 尚未更新），算出來的 loss 並不會記錄到圖片中，而第 21 次時算完 loss（也就是第 (1) 步時）就 break 掉。

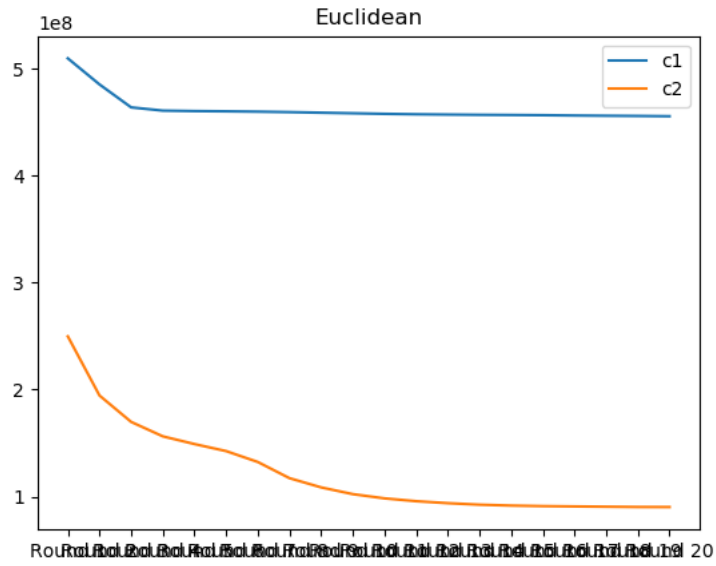
二、實驗數據

（2 - 1）loss 紀錄圖

Manhattan



Euclidean



(2 - 2) 與 centroids 距離圖

Loss Function：Euclidean

Distance Function: c1_Euclidean

Euclidean	1	2	3	4	5	6	7	8	9	10

1	0.0	692.16	3490.26	205.75	346.72	512.61	444.73	566.2	1282.77	307.67
2		0.0	2798.8	897.66	1038.83	1204.08	1136.33	1257.45	669.89	412.08
3			0.0	3695.11	3836.91	4002.69	3934.87	4056.14	2294.58	3195.92
4				0.0	142.44	309.51	241.73	363.26	1474.95	504.63
5					0.0	167.15	99.55	220.9	1615.85	646.93
6						0.0	67.91	53.79	1782.2	814.08
7							0.0	121.63	1715.25	746.34
8								0.0	1835.64	867.82
9									0.0	975.32
10										0.0

Distance Function: c2_Euclidean

Euclidean	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.0	15760.12	14110.83	9045.32	5567.68	1924.62	1100.86	402.89	2105.44	3169.0
2		0.0	11524.51	6743.88	10192.53	14455.12	14682.45	15362.42	13674.71	12597.04
3			0.0	9545.88	10883.38	12233.96	13208.0	13786.48	12508.96	11938.38
4				0.0	3494.22	7718.22	7957.78	8644.81	6947.82	5876.33
5					0.0	4404.56	4492.46	5169.94	3488.16	2407.92
6						0.0	1182.86	1615.79	1313.33	2153.77
7							0.0	698.49	1010.2	2085.46
8								0.0	1702.79	2768.61
9									0.0	1080.53
10										0.0

Distance Function: c1_Manhattan

Manhattan	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.0	728.92	3797.9	212.18	374.89	577.4	499.16	645.77	1731.06	406.7
2		0.0	3072.89	935.89	1100.83	1303.9	1225.35	1372.09	1005.29	490.93
3			0.0	4001.04	4170.3	4372.79	4294.95	4440.72	2513.42	3396.42

4				0.0	171.37	375.25	296.25	443.5	1934.09	609.75
5					0.0	204.52	125.6	272.93	2102.86	779.4
6						0.0	79.4	69.59	2306.38	983.02
7							0.0	147.87	2227.56	904.37
8								0.0	2374.55	1050.92
9									0.0	1327.58
10										0.0

Distance Function: c2_Manhattan

Manhattan	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.0	15772.61	20215.65	9533.17	5604.2	3088.05	1311.04	471.27	2369.41	3349.66
2		0.0	16003.5	7219.2	10221.03	16105.35	14909.17	15434.46	13950.58	12776.88
3			0.0	10690.48	14613.55	17509.9	18912.61	19748.94	17851.81	16873.24
4				0.0	3935.29	8896.39	8228.36	9065.4	7168.73	6190.68
5					0.0	5893.07	4696.98	5221.25	3737.71	2564.17
6						0.0	1781.82	2619.81	2162.8	3337.75
7							0.0	840.72	1068.94	2137.79
8								0.0	1901.21	2883.73
9									0.0	1176.45
10										0.0

Loss Function：Manhattan

Distance Function: c1_Euclidean

Euclidean	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.0	2219.18	9948.04	528.7	413.37	827.72	681.03	917.13	832.15	729.06
2		0.0	7767.95	2734.05	2628.49	3044.48	2898.71	3133.46	1812.45	1491.36
3			0.0	10433.06	10361.37	10773.53	10626.49	10862.97	9340.28	9236.84
4				0.0	221.37	375.16	249.38	457.26	1156.58	1251.16

5					0.0	415.99	270.75	505.07	1171.96	1137.14
6						0.0	147.05	89.49	1529.46	1553.12
7							0.0	236.51	1391.55	1407.4
8								0.0	1613.56	1642.13
9									0.0	709.41
10										0.0

Distance Function: c2_Euclidean

Euclidean	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.0	15747.23	14100.14	9032.33	5554.79	2006.7	1338.16	514.63	1571.24	3022.66
2		0.0	11524.51	6743.88	10192.53	14474.55	14412.06	15239.88	14328.23	12731.4
3			0.0	9545.88	10883.38	12167.79	13125.35	13684.61	12643.99	12006.39
4				0.0	3494.22	7742.63	7694.28	8521.2	7588.4	6009.82
5					0.0	4452.97	4219.76	5047.52	4167.64	2542.57
6						0.0	1405.11	1637.73	910.99	2124.26
7							0.0	827.84	566.55	1684.52
8								0.0	1081.38	2511.46
9									0.0	1649.39
10										0.0

Distance Function: c1_Manhattan

Manhattan	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.0	2341.02	11929.3	651.19	496.33	947.74	770.74	1056.8	1260.51	737.71
2		0.0	9597.44	2778.95	2830.14	3280.36	3104.29	3388.98	2380.46	1605.27
3			0.0	12323.29	12421.26	12871.48	12695.55	12979.13	10775.94	11196.79
4				0.0	335.95	558.47	382.46	667.53	1653.83	1379.17
5					0.0	452.86	276.33	561.85	1755.11	1226.66
6						0.0	177.59	110.22	2205.31	1677.67
7							0.0	287.43	2028.9	1500.99

8								0.0	2314.67	1786.81
9									0.0	1006.37
10										0.0

Distance Function: c2_Manhattan

Manhattan	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.0	15757.69	20200.26	9517.67	5588.85	3281.49	1430.21	602.95	2102.55	3211.46
2		0.0	16003.5	7219.2	10221.03	16325.27	14506.49	15335.96	14980.06	12922.93
3			0.0	10690.48	14613.55	17521.52	18775.12	19602.26	18111.89	16995.13
4				0.0	3935.29	9116.02	8090.51	8918.81	7771.22	6312.53
5					0.0	6110.83	4293.5	5123.07	4768.92	2710.06
6						0.0	1855.58	2682.57	1358.8	3413.04
7							0.0	833.43	674.83	1784.51
8								0.0	1500.82	2614.0
9									0.0	2062.25
10										0.0

三、Improvement