HW3

本次作業需使用 K-means (Euclidean Distance & Manhatten Distance)，逐步將 cluster 中的 point 進行分群。以下為實作步驟：

1. **分別讀入 `data.txt` / `c1.txt` / `c2.txt` 。其中，對於 `data.txt` / `c1.txt` / `c2.txt` ，會再進行 `expandPointData` FlatMap 的處理：**

   `expandPointData` ，重整資料，以方便後續計算。將資料處理成：

   $$(dimension_i, (point_i, dimension_{value}))$$
   $$(2, (1, 273.34))$$

   同時，再針對`data.txt` ，進行以下 FlatMap 處理：

   `tidyPointData` ，負責紀錄每一點的全部 Dimemsion 資訊。將資料處理成

   $$(point_i, [dimen1_{value}, dimen2_{value}, dimen3_{value} ...])$$
   $$(0, [0, 0.64, 0.64, ...])$$

2. **進入 for-loop 後，開始指派 Point 到最近的 cluster centroid：**

   ```
   eucliResult
   = pointData.join(cData).map(minimizeFunc).reduceByKey(lambda x, y: x+y)
   ```

   將各 point data ，與 centroid data 進行 join ，讓每個 Point 的各 Dimensions ，與每個 Centroid Point 的各 Dimension 的交叉合併。接著，在進入 `minimizeFunc` 的處理。

   `minimizeFunc` 共有 `Euclidean` 版本與 `Manhatten` 版本，此步驟會試圖讓每個 point 與 centroid point ，套用各自的 Distance 公式並找出 cost。

   最終再用 reducer 將計算結果，以 `((point_idx, c_point_idx), cost)` 方式整合。因此，我們可以得到每個點與各個 centroid 點的 cost 。

   ```
   .map(lambda val: (val[0][0], [(val[0][1], val[1])])).reduceByKey(lambda x,
   y: x+y)
   ```

此步驟試著將資料整理成 `(currentPoint, [ (cPoint1, CPoint1DimemTotalDistance), (cPoint2, CPoint2DimemTotalDistance) ..... ])` 的版本，方便後續比較。

3. **依據各 `k-means` 的算法，找出最近的 centroid_point：**

```
# Map would return "point_id: cluster_id"
midResult = eucliResult.flatMap(findOptimalCPoint)

def findOptimalCPoint(val):
    clusterMap = []
    smallestCIdx = 0
    smallestCVal = 9999999999
    for perVal in val[1]:
        if (perVal[1] < smallestCVal):
            smallestCIdx = perVal[0]
            smallestCVal = perVal[1]
    clusterMap.append((val[0], (smallestCIdx, smallestCVal)))
    return clusterMap
```

透過 `findOptimalCPoint` ，跑 for-loop 找出 cost 最低的 centroid point ，最佳的 cluster ，並輸出 `(point_idx, (最佳的 c_point_id, 最佳的 c_point_value))`

4. **重新計算各 centroid 的 all dimensions：**

```
finalResult = midResult.join(pointAllDimems).map(retidyMergeData)
```

將剛才的結果，與 point 進行 join mapping 後，再透過 `retidyMergeData` 將資料處理成， `(cPointIdx, myPointVal)` ex: (2, [0, 0.64, 0.64, ...])。

```
.reduceByKey(lambda x, y: x + y).map(calculateMean)

def calculateMean(val):
    totalPointNum = 0
    pointData = [0]*totalDimemsions
    iterationTimes = int(len(val[1])/totalDimemsions)
    for idx in range(iterationTimes):
        for dimemIdx in range(totalDimemsions):
            pointData[dimemIdx] = pointData[dimemIdx] + float(val[1][idx*t
otalDimemsions + dimemIdx])
        totalPointNum = totalPointNum + 1
    for dimemIdx in range(totalDimemsions):
```

```
        pointData[dimemIdx] = (pointData[dimemIdx]/totalPointNum)
    return (val[0], pointData)
```

此步驟，會將集中至任一 cluster 的 point ，全部 dimension 進行平均計算，再更新各自的 centroid dimension 值。

5. **重新將各 centroid 的結果，改成 for-loop 一開始支援的格式：**

   重新將資料整理成

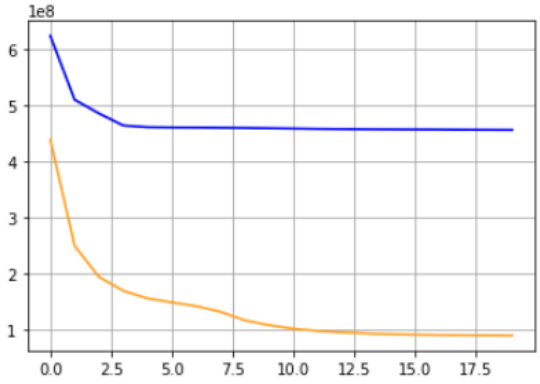
$$(dimension_i, (point_i, dimension_{value}))$$
$$(2, (1, 273.34))$$

```
cData =
finalResult.flatMap(lambda val: [(dimemIdx, (val[0], perDimem)) for dimemI
dx, perDimem in enumerate(val[1])] )
```

實驗數據

Euclidean

	C1	C2
Round1	623660345.306	438747790.028
Round2	509862908.298	249803933.626
Round3	485480681.872	194494814.406
Round4	463997011.685	169804841.452
Round5	460969266.573	156295748.806
Round6	460537847.983	149094208.109
Round7	460313099.654	142508531.620
Round8	460003523.889	132303869.407
Round9	459570539.318	117170969.837
Round10	459021103.342	108547377.179
Round11	458490656.192	102237203.318
Round12	457944232.588	98278015.750
Round13	457558005.199	95630226.122
Round14	457290136.352	93793314.051
Round15	457050555.060	92377131.968
Round16	456892235.615	91541606.254
Round17	456703630.737	91045573.830
Round18	456404203.019	90752240.101
Round19	456177800.542	90470170.181
Round20	455986871.027	90216416.176



Euclidean - cost v.s. iteration

Percentage Improvement:

c1: 26.88538329 %

c2: 79.43775029 %

由圖表可知， C1 的收斂較 C2 差。由於 Euclidean 對於距離的敏感度較高，同時 C1 相對於 C2 而言資料更集中，導致最後收斂時，C1 無法最佳的分散到額外的 cluster ，而落入區域的最佳解。

C1 - Euclidean

	0	1	2	3	4	5	6	7	8	9
1	0	692.158	3490.259	205.750	346.719	512.612	444.731	566.202	1282.771	307.669
2		0	2798.801	897.659	1038.827	1204.078	1136.327	1257.450	669.890	412.076
3			0	3695.114	3836.907	4002.689	3934.872	4056.136	2294.580	3195.924
4				0	142.439	309.506	241.730	363.263	1474.945	504.634
5					0	167.150	99.546	220.902	1615.852	646.931
6						0	67.912	53.790	1782.203	814.076
7							0	121.634	1715.253	746.336
8								0	1835.640	867.823
9									0	975.320
10										0

C2 - Euclidean

	0	1	2	3	4	5	6	7	8	9
1	0	15760.122	14110.834	9045.320	5567.685	1924.624	1100.859	402.891	2105.443	3169.004
2		0	11524.506	6743.884	10192.525	14455.119	14682.451	15362.418	13674.708	12597.040
3			0	9545.879	10883.382	12233.960	13208.003	13786.484	12508.957	11938.376
4				0	3494.222	7718.222	7957.776	8644.807	6947.821	5876.330
5					0	4404.563	4492.458	5169.937	3488.159	2407.919
6						0	1182.864	1615.788	1313.327	2153.771
7							0	698.488	1010.198	2085.461
8								0	1702.793	2768.608
9									0	1080.535
10										0

C1 - Manhattan

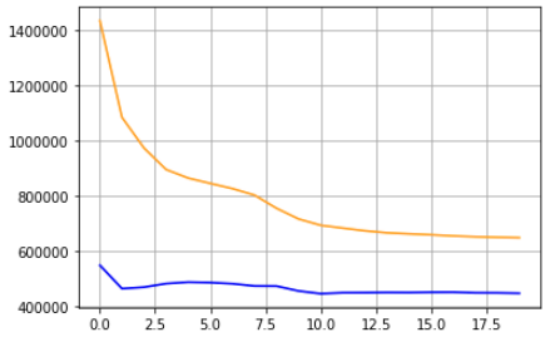
	0	1	2	3	4	5	6	7	8	9
1	0	728.924	3797.899	212.181	374.890	577.402	499.158	645.770	1731.064	406.701
2		0	3072.889	935.885	1100.833	1303.896	1225.352	1372.092	1005.293	490.928
3			0	4001.038	4170.305	4372.789	4294.953	4440.720	2513.423	3396.420
4				0	171.365	375.248	296.255	443.498	1934.087	609.749
5					0	204.523	125.597	272.935	2102.865	779.397
6						0	79.402	69.590	2306.380	983.020
7							0	147.866	2227.556	904.370
8								0	2374.545	1050.916
9									0	1327.584
10										0

C2 - Manhattan

	0	1	2	3	4	5	6	7	8	9
1	0	15772.615	20215.646	9533.171	5604.200	3088.054	1311.039	471.266	2369.412	3349.657
2		0	16003.499	7219.197	10221.031	16105.347	14909.170	15434.460	13950.576	12776.883
3			0	10690.484	14613.552	17509.903	18912.605	19748.936	17851.807	16873.244
4				0	3935.293	8896.389	8228.355	9065.404	7168.733	6190.679
5					0	5893.070	4696.975	5221.253	3737.707	2564.171
6						0	1781.823	2619.811	2162.802	3337.746
7							0	840.723	1068.940	2137.788
8								0	1901.209	2883.735
9									0	1176.450
10										0

Manhattan

	C1	C2
Round1	550117.142	1433739.310
Round2	464661.072	1084488.777
Round3	471200.049	973431.715
Round4	484160.694	895934.593
Round5	489251.723	865128.335
Round6	487564.741	845846.647
Round7	483404.056	827219.583
Round8	475365.341	803590.346
Round9	474924.050	756039.517
Round10	457233.640	717332.903
Round11	447495.090	694587.925
Round12	451004.306	684444.502
Round13	451222.093	674574.748
Round14	451973.846	667409.470
Round15	451585.355	663556.628
Round16	452756.645	660162.777
Round17	452893.792	656041.322
Round18	450382.234	653036.754
Round19	450023.968	651112.426
Round20	448929.474	649689.013



Manhattan - cost v.s. iteration

Percentage Improvement:

c1: 18.393840%

c2: 54.685694%

由圖表可知， C1 比 C2 收斂的更好。透過推論，Manhattan 算法對於部分離群值敏感度較低。

當使用 C1 進行 centroid 計算時，由於 point 的分佈是均勻分配，在 manhattan 算法下會忽略離群值問題，使大部分的點能更趨近 cluster centroid point ，讓最終結果更趨向全域最佳解。

儘管 M 算法對於離群值較低敏感，但處理 C2 centroid 時，由於整體資料過 Sparse，仍容易將值拉向離群，進入區域最佳解。

C1 - Euclidean

	0	1	2	3	4	5	6	7	8	9
1	0.	2214.960	9943.770	531.301	415.770	831.594	685.155	921.176	828.932	724.882
2		0.000	7767.950	2732.840	2626.580	3044.100	2898.580	3133.250	1812.450	1491.360
3			0.000	10432.000	10359.500	10773.100	10626.400	10862.800	9340.280	9236.840
4				0.000	221.316	375.438	249.613	457.816	1155.800	1249.910
5					0.000	417.519	272.536	506.763	1170.530	1135.220
6						0.000	146.795	89.666	1529.090	1552.740
7							0.000	236.438	1391.420	1407.270
8								0.000	1613.360	1641.910
9									0.000	709.408
10										0.000

C2 - Euclidean

	0	1	2	3	4	5	6	7	8	9
1	0	15747.234	14100.145	9032.333	5554.787	2006.703	1338.161	514.627	1571.243	3022.661
2		0	11524.506	6743.884	10192.525	14474.554	14412.057	15239.877	14328.226	12731.398
3			0	9545.879	10883.382	12167.794	13125.351	13684.607	12643.986	12006.395
4				0	3494.222	7742.628	7694.277	8521.198	7588.405	6009.820
5					0	4452.972	4219.761	5047.516	4167.637	2542.569
6						0	1405.109	1637.729	910.994	2124.263
7							0	827.841	566.551	1684.516
8								0	1081.379	2511.459
9									0	1649.389
10										0

C1 - Manhattan

	1	2	3	4	5	6	7	8	9	10
1	0.	2336.040	11924.100	652.125	499.653	952.467	775.805	1061.780	1255.350	732.704
2		0.000	9597.440	2777.740	2828.260	3279.860	3104.120	3388.740	2380.460	1605.270
3			0.000	12322.900	12419.400	12871.000	12695.400	12978.900	10775.900	11196.800
4				0.000	335.664	558.134	382.469	667.483	1653.440	1376.940
5					0.000	454.244	278.003	563.475	1753.200	1224.760
6						0.000	177.255	110.470	2204.800	1677.150
7							0.000	287.351	2028.740	1500.840
8								0.000	2314.430	1786.570
9									0.000	1006.370
10										0.000

C2 - Manhattan

	0	1	2	3	4	5	6	7	8	9
1	0	15757.691	20200.259	9517.668	5588.854	3281.488	1430.209	602.955	2102.554	3211.456
2		0	16003.499	7219.197	10221.031	16325.271	14506.486	15335.957	14980.056	12922.931
3			0	10690.484	14613.552	17521.518	18775.121	19602.263	18111.885	16995.134
4				0	3935.293	9116.024	8090.510	8918.813	7771.222	6312.530
5					0	6110.833	4293.502	5123.067	4768.923	2710.056
6						0	1855.580	2682.569	1358.796	3413.036
7							0	833.430	674.828	1784.512
8								0	1500.825	2613.997
9									0	2062.251
10										0