ML HW6 report

資科工碩一

0756138

黃伯凱

我有試過很多 gamma，也有記錄每一筆的時間，所以有非常多圖以及 csv 檔。
下面連結包含了所有圖片、時間紀錄、影片以及此次作業的 code
https://drive.google.com/drive/folders/1x-4ZbrU0_ssNhtwNbpOY1mkfoOLWmPwi?usp=sharing

Code Explanation

一些基本的 function

```
def rand_float(a, b):                                    回傳介於 a~b 之間的小數
    return random.uniform(a, b)

def pts_dis(A, B):                                       計算點 A 和點 B 之間的距離
    return math.hypot(A[0] - B[0], A[1] - B[1])

def plot_cluster(D, pred_c, mu, c):                      不同 cluster 會依序畫出 blue, green, red,
    clr = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']       cyan, magenta, yellow, black, white 的點
    x = D[pred_c==c][:, 0]
    y = D[pred_c==c][:, 1]
    plt.scatter(x, y, color=clr[c], alpha=0.3, marker='.')
    if np.all(mu) != None:                               如果是 cluster mean，會畫出 *
        plt.scatter(mu[c][0], mu[c][1], color=clr[c], marker='*', s=100)

def rbf_kernel(a, b, gamma):
    return math.exp(-gamma * ((pts_dis(a, b))**2))       Kernel function 為 rbf

def gram_matrix(datas, gamma):                           Kernel K means 會用到的
    gram = np.zeros((len(datas), len(datas)))            gram matrix
    for p in range(len(datas)):
        for q in range(len(datas)):
            gram[p][q] = rbf_kernel(datas[p], datas[q], gamma)
    return gram

def cluster_count(c, k):
    return np.count_nonzero(c == k)                      計算每個 cluster 中有幾個 data
```

K means

K means 的做法是先 initial k 個 mean，並算出所有 data point 與這 k 個 mean 的距離，最後依照這些距離來 assign cluster 給所有的 data point。

```
def K_means(datas, k, dim, method="ClusterMedian", plot_iter=False):
    iteration = 0
    mu = choose_mu(datas, k, dim, method)  # initial mu
    pred_c = np.zeros(len(datas))  # initial prediction

    while(True):
        tmp_mu = np.zeros((k, dim))
        dist_mat = np.zeros((len(datas), k))

        # Show iteration status
        if plot_iter:
            for k_ in range(k):
                plot_cluster(moon, pred_c, mu, k_)
            title = "iteration : " + str(iteration)
            plt.title(title)
            plt.savefig("imgs/iteration/Kmeans/%s_K%d_iter%d.png" %(method, k, iteration+1))
            plt.show()
        for k_ in range(k):
            rep_mu = np.repeat([mu[k_]], len(datas), axis=0)
            delta = abs(np.subtract(datas, rep_mu))
            dist_mat[:, k_] = np.sqrt(np.square(delta).sum(axis=1))
        pred_c = np.argmin(dist_mat, axis=1)
        for k_ in range(k):
            if np.count_nonzero(pred_c==k_) > 0:
                tmp_mu[k_] = np.mean(datas[pred_c==k_], axis=0)
            else:
                tmp_mu[k_] = mu[k_]

        iteration += 1
        iter_flag = np.all(highdim_pts_dis(mu, tmp_mu, k).T) < 1e-5
        if iter_flag:
            print("Iteration ends, it takes %d iteration\n" %iteration)
            break

        mu = tmp_mu
    return mu, pred_c
```

初始化 mu，有三種初始化方法：

ClusterMedian,

RandomDataPoint,

Random

會在下一頁解釋

show 每個 iteration 的圖形

計算 data 與 mu 之間的距離

重新計算 cluster 的中心點 mu

新的 mu 和舊的 mu 距離小於 1e-5 的時候才收斂

我在這次作業裡使用了三種 initial mean 的方式，分別為：

ClusterMedian、RandomDataPoint、Random

```python
def choose_mu(datas, k, dim, method):

    # Pick the median of each cluster
    if method == "ClusterMedian":
        # If each cluster has same number of data
        # gap will be the number of data points in each cluster
        gap = len(datas) / k

        # argsort according to the first column
        sort_datas = datas[datas[:, 0].argsort()]

        # argsort for each column according to find different cluster
        # (each gap contains one cluster)
        # If data points belong to same cluster, they will have similar coordinates
        for col in range(1, dim):
            tmp = sort_datas[int(gap*col) : ].copy()
            tmp = tmp[tmp[:, col].argsort()]
            sort_datas[int(gap*col) : ] = tmp
        # Pick median from each gap
        mu = np.zeros((k, dim))
        for i in range(k):
            idx = int(i*gap + 1/2 *gap)
            mu[i] = sort_datas[idx]
        return mu

    # Randomly pick points from datas
    if method == "RandomDataPoint":
        while(True):
            idx = np.random.randint(len(datas), size=k)
            idx.sort()
            # Check if it picked the same data points
            if np.all(np.unique(idx) == idx):
                break
        mu = datas[idx]
        return mu
    # Pick points within the range of data points
    if method == "Random":
        return np.random.uniform(datas.min(), datas.max(), (k, dim))
```

ClusterMedian：

事先把 data 依據座標位址大略分成 k 個 cluster，再將這 k 個 cluster 中的每一個 data point 依據座標大小排序，最後選出每一個 cluster 最中間的 data point 當作 cluster 中心 mu

依照座標大小排序，排完的 cluster 就不再排

RandomDataPoint：

隨機挑 k 個 data point 當作 cluster 的中心 mu

Random：

隨機挑 k 個點當作 Cluster 的中心，挑選範圍為 data 的 min 和 Max 之間

Kernel K means

Kernel K means 是把原來的 Data point 都先轉換到 Kernel Space 上，最後再在 Kernel Space 上做 K means。但在這邊的做法是跟老師上課的作法一樣，沒有求出每個 data 在 Kernel Space 相對應的座標，而是直接算出每個 data 在 Kernel Space 當中與 mean 的距離。

```
def Kernel_K_means(datas, k, gamma, method="Original"):
    print("Data Training...")
    iteration = 0
    assign = np.zeros((len(datas), k))
    pred_c = init_cluster(datas, k, method)
    gram = gram_matrix(datas, gamma)

    for i in range(len(datas)):
        assign[i][pred_c[i]] = 1

    print("K = %d" %k)
    while(True):
        tmp_assign = np.copy(assign)
        tmp_pred_c = np.copy(pred_c)
        for i, data in enumerate(datas):
            dist = np.zeros(k)
            for k_ in range(k):
                k_i_j = 0
                for n in range(len(datas)):
                    k_i_j += rbf_kernel(data, datas[n], gamma) * assign[n,k_]

            mem_count = cluster_count(tmp_pred_c, k_)
            k_i_i = rbf_kernel(data, data, gamma)
            a_k_p = assign[:, k_].T
            a_k_q = assign[:, k_]
            dist[k_] = k_i_i - 2/mem_count * k_i_j + 1/(mem_count**2) * np.matmul(np.ma

            c = np.argmin(dist)
            tmp_assign[i] = np.zeros(k)
            tmp_assign[i][c] = 1
            tmp_pred_c[i] = c

        iteration += 1
        if np.count_nonzero(pred_c == tmp_pred_c) >= len(pred_c)*0.9:
            print("\nIteration ends, it takes %d iteration" %iteration)
            break

        assign = tmp_assign
        pred_c = tmp_pred_c
    return pred_c
```

有三種 initial 每一個 data 屬於哪過 cluster 的方法，在下一頁解釋

計算 data 的 gram matrix

當第 i 筆 data 屬於 cluster j 時，assign[i][j] = 1

老師講義中的距離公式

np.matmul(a_k_p, gram), a_k_q)

更新 cluster 中心 mu 的值

當舊的 prediction 與新的 prediction 有 9 成相似的時候即收斂

我在這次作業做了 3 種 initial cluster 的方法，分別是：

Random, Original, DataCenter

```
def init_cluster(datas, k, method):
    if method == "Random":
        return np.random.randint(0, k, len(datas))


    if method == "Original":
        ori = np.zeros((len(datas), 2))
        delta = abs(np.subtract(ori, datas))
        dist = np.sqrt(np.square(delta).sum(axis=1))

        # Split into k clusters
        split_mu = []
        data_sum = np.asscalar(np.sum(dist, axis=0))
        for i in range(1, k):
            split_mu.append(data_sum / len(datas) * 2 * i / k)
        pred_c = np.zeros(len(datas))
        if k<3:
            pred_c[dist >= split_mu[0]] = 1
        else:
            for i in range(1, len(split_mu)+1):
                pred_c[dist >= split_mu[i-1]] = i
        pred_c = pred_c.astype(int)
        return pred_c


    if method == "DataCenter":
        center = np.mean(datas, axis=0)
        delta = abs(np.subtract(center, datas))
        dist = np.sqrt(np.square(delta).sum(axis=1))
        # Split into k clusters
        split_mu = []
        data_sum = np.asscalar(np.sum(dist, axis=0))
        for i in range(1, k):
            split_mu.append(data_sum / len(datas) * 2 * i / k)

        pred_c = np.zeros(len(datas))
        if k<3:
            pred_c[dist >= split_mu[0]] = 1
        else:
            for i in range(1, len(split_mu)+1):
                pred_c[dist >= split_mu[i-1]] = i
        pred_c = pred_c.astype(int)
        return pred_c
```

Random :

隨機分配所有 data 一個介於 0~k-1 之間的值當作他的 cluster number

Original :

算出所有 data 與原點的距離差，依照距離遠近分成 k 群

DataCenter :

先算出整個 data 的中心點，再算出所有 data 與該中心點的距離差，依照距離遠近分成 k 群

Spectral Clustering

Spectral Clustering 是利用 Similarity, Degree 建立出 Laplacian Matrix，可以利用 Laplacian Matrix 來算出 minimum cut cost。透過 Rayleigh Quotient 知道要找出 Laplacian Matrix 的 Eigenvalue, Eigenvector。用 Laplacian Matrix 的 Eigenvector 來當作新的座標軸，並在這個空間使用 K means 找出每個 data point 相對應的 Cluster。

而 Eigenvector 的挑選則是挑出第 2 小 ~ 第 k+1 小的 Eigenvalue 相對應的 Eigenvector，不挑最小 Eigenvalue 的 Eigenvector 是因為這一個 Eigenvector 是 1 矩陣。

```
def build_similarity(datas, gamma):
    similarity = np.zeros((len(datas), len(datas)))
    for i in range(len(datas)):
        for j in range(i, len(datas)):
            similarity[i][j] = rbf_kernel(datas[i], datas[j], gamma)
    for i in range(1, len(datas)):
        for j in range(i):
            similarity[i][j] = similarity[j][i]
    return similarity

def build_degree(datas, similarity):
    degree = np.zeros((len(datas), len(datas)))
    for i in range(len(datas)):
        degree[i][i] = np.sum(similarity[i])
    return degree

def build_laplacian(similarity, degree):
    return degree - similarity
```

建立 Similarity Matrix

建立 Degree Matrix

建立 Laplacian Matrix

```
def Spectral_Clustering(datas, k, gamma, method="ClusterMiddle", plot_iter=False):
    similarity = build_similarity(datas, gamma)
    degree = build_degree(datas, similarity)
    laplacian = build_laplacian(similarity, degree)

    e_val, e_vec = LA.eig(laplacian)
    tmp_e_val = np.copy(e_val)
    ind_vec = e_vec[:, 1:k+1]
    return K_means(ind_vec, k, k, method, plot_iter)
```

利用 numpy 的 linalg.eig 來算出 Eigenvalue 和 eigenvector

Spectral Clustering Coordinate Discussion

找出 Graph Laplacian 前 k 個最小的特徵值與對應的特徵向量，將原本的資料點投影到這 k 個特徵向量上，再將投影的結果進行 k means，就可以找出原本每個集群都是完全連通的分割。

換句話說，只要找出每一個 data point 的 indicator vector 就可以拿去做 K means 分群。而每一個 data 的 indicator vector 其實就是 eigenvector 的每一列。

將 Spectral Clustering 每一個 data 的 indicator vector (左) 以及最後預測的 cluster number (右) 印出來：

```
Data 0 : [0.02581992 0.02581985]
Data 1 : [-0.02581985  0.02581992]
Data 2 : [-0.02581986  0.02581992]
Data 3 : [-0.02581985  0.02581992]
Data 4 : [0.02581991 0.02581985]
Data 5 : [-0.02581986  0.02581992]
Data 6 : [-0.02581985  0.02581992]
Data 7 : [0.02581992 0.02581985]
Data 8 : [-0.02581986  0.02581992]
Data 9 : [-0.02581986  0.02581992]
Data 10 : [-0.02581985  0.02581992]
Data 11 : [0.02581992 0.02581985]
Data 12 : [0.02581993 0.02581985]
Data 13 : [0.02581993 0.02581985]
Data 14 : [0.02581992 0.02581985]
Data 15 : [-0.02581986  0.02581992]
Data 16 : [-0.02581985  0.02581992]
Data 17 : [0.02581993 0.02581985]
Data 18 : [0.02581992 0.02581985]
Data 19 : [0.02581992 0.02581985]
```

```
Data 0 pred : 1
Data 1 pred : 0
Data 2 pred : 0
Data 3 pred : 0
Data 4 pred : 1
Data 5 pred : 0
Data 6 pred : 0
Data 7 pred : 1
Data 8 pred : 0
Data 9 pred : 0
Data 10 pred : 0
Data 11 pred : 1
Data 12 pred : 1
Data 13 pred : 1
Data 14 pred : 1
Data 15 pred : 0
Data 16 pred : 0
Data 17 pred : 1
Data 18 pred : 1
Data 19 pred : 1
```

發現在投影到 eigenvector 後的空間當中，大致上可以分為第一象限以及第二象限的 data point，相比預測的 cluster number 發現確實第 1 象限是 cluster 1, 第 2 象限是 cluster 0。

Discussion

作業結果都在下一頁的 result 裡，由於圖片太多，故擺在做後才顯示

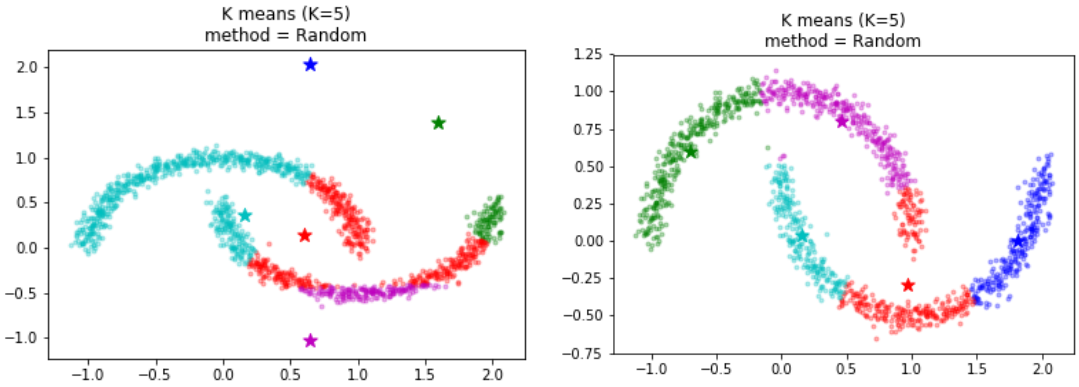
K means

i. 只有辦法把圖形區分成線性的

因為餵 n dimension 的資料給 K means，他就只能判斷 n dimension 的結果，沒有辦法轉換到高維來判斷。所以 moon 跟 circle 圖形都沒有辦法完美的區分

ii. Random initial mean 的方式每次跑出來的結果會不一樣

因為初始 mean 不會相同，所以結果當然會有差異

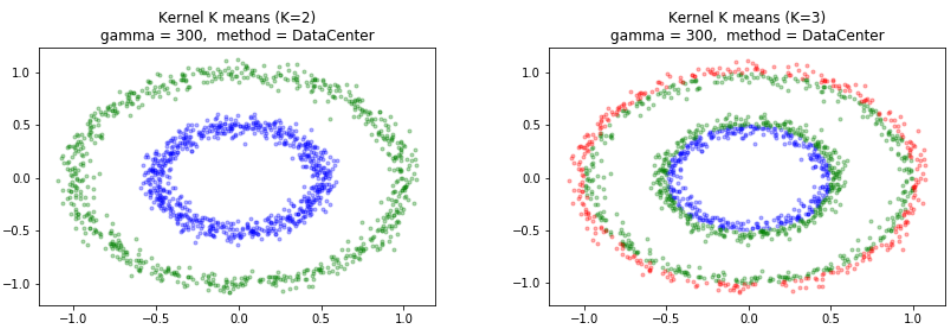


Kernel K means

一開始只有寫 random initial cluster，但是發現這樣寫的話還是沒有辦法把 moon 和 circle 分成兩群，所以才想到要用 Origin 和 DataCenter 來解決。

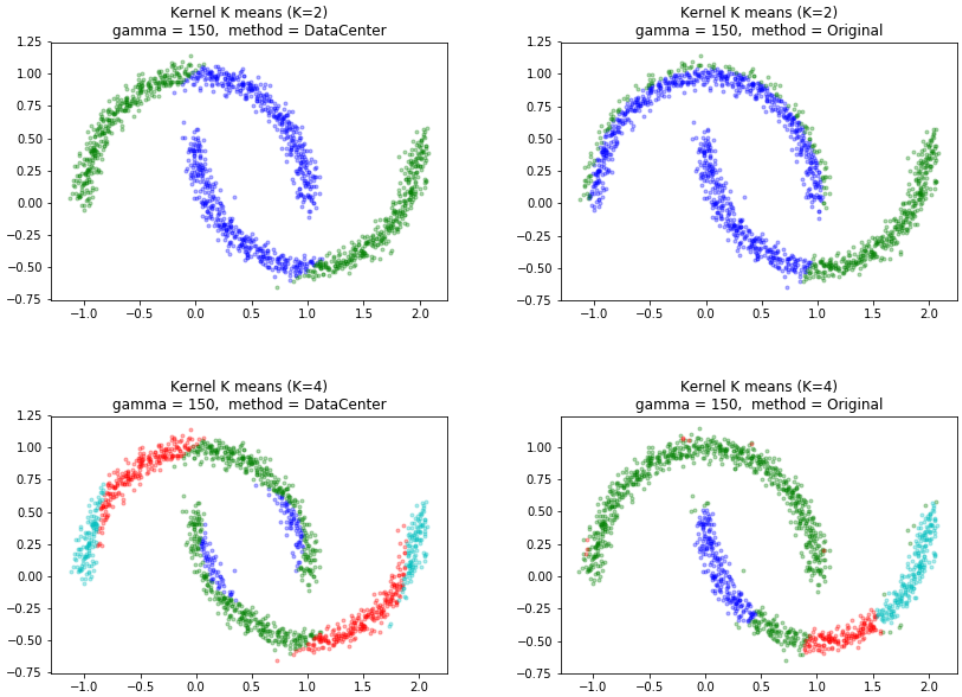
i. Origin 和 DataCenter 可以把 circle 分成內外圈兩群

因為 circle 是對稱且中心點在原點的圖形，所以用這兩種解法是最合適的。



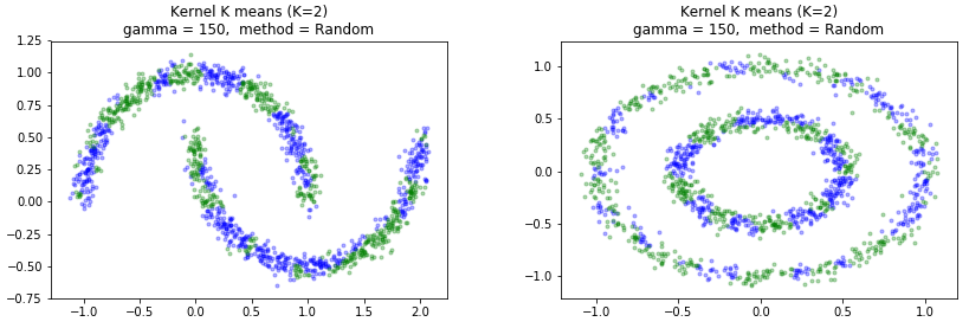
ii. 但是 Origin 和 DataCenter 依然沒辦法把 moon 分成兩群

這兩個 initial 方式結果已經不再像 k means 一樣是線性的，很明顯看得出來是離中心點越遠會是不同的 cluster，但還是沒有辦法把它變成明顯的兩群。



iii. Random 完全分不出群

從下面的 result 看得出來，random 根本是在亂分，原因也是因為他是隨機 initial cluster，所以從一開始的時候 cluster 就已經被打亂了。

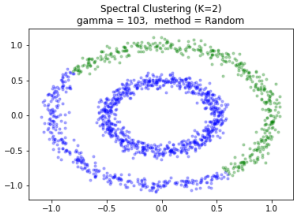


iv. 計算時間很長

因為每一次都要算 gram matrix，並再利用 gram matrix 做運算，所以整體時間變長。Original 和 DataCenter 平均需要 15.5 秒左右，而 Random 就要 27 秒。

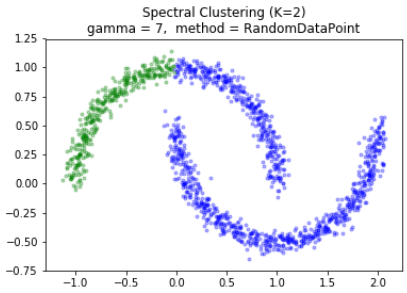
Spectral Clustering

一開始只用 random initial cluster mean 的時候一樣沒有辦法把 cluster 分乾淨，所以想到用看看抓出每一個 cluster 當中的其中一個 data point 來當 mu 試看看，所以就有了 ClusterMedian 和 RandomDataPoint。

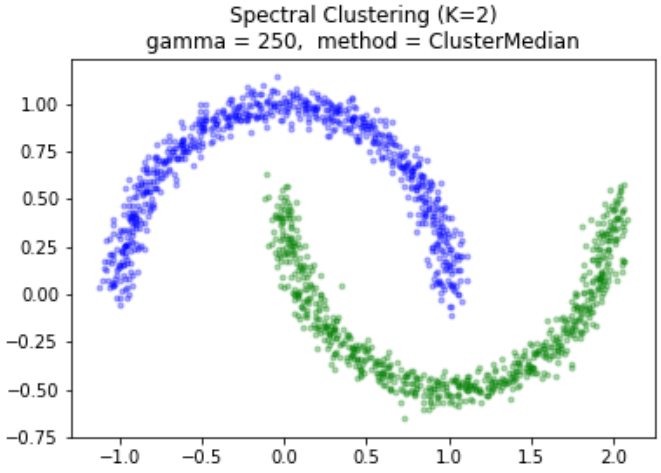


i. 當中最準的是 ClusterMedian

先將 indicator vector 依序排序好之後，再取出最中間的那一個 data point 當作 mean。當初在做 moon 的 clustering 的時候發現有時候兩端會被分到不同 cluster(如右圖)



想說會不會是因為他在綠色的 cluster 剛好取到靠左端的 data point 當作 mean，所以試試看取最中間的 data point。結果發現用 ClusterMedian 的方法比起另外兩種穩定非常多，一直到接近 300 才會誤判。



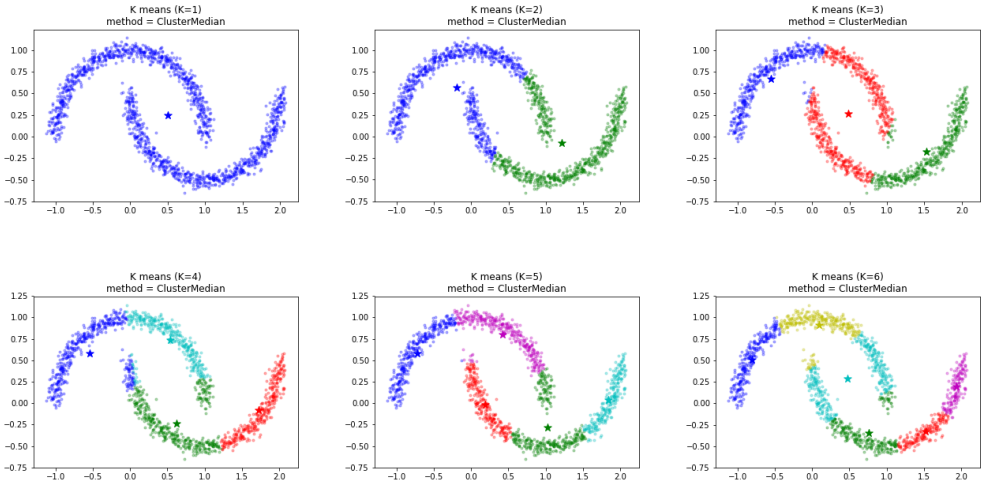
Result

因為圖片太多，所以只放部分，如果要看圖第一頁有連結

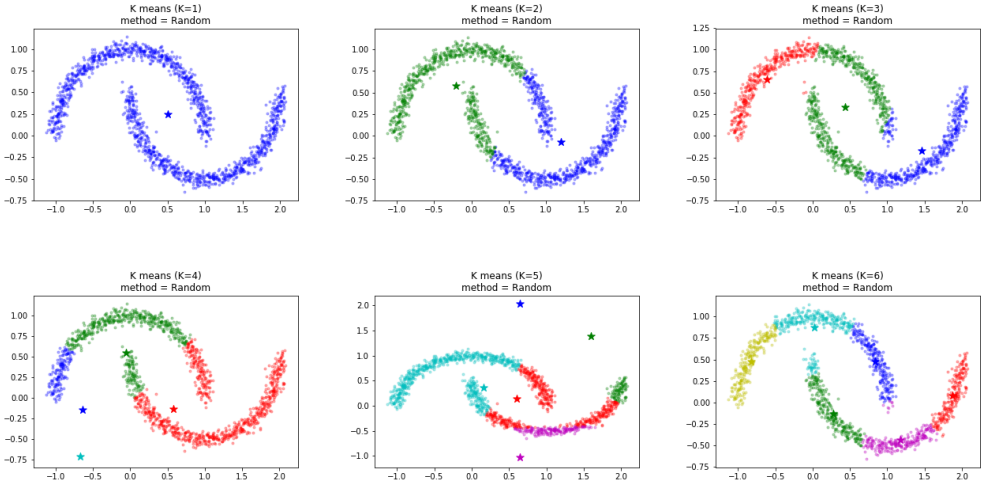
K means

Moon

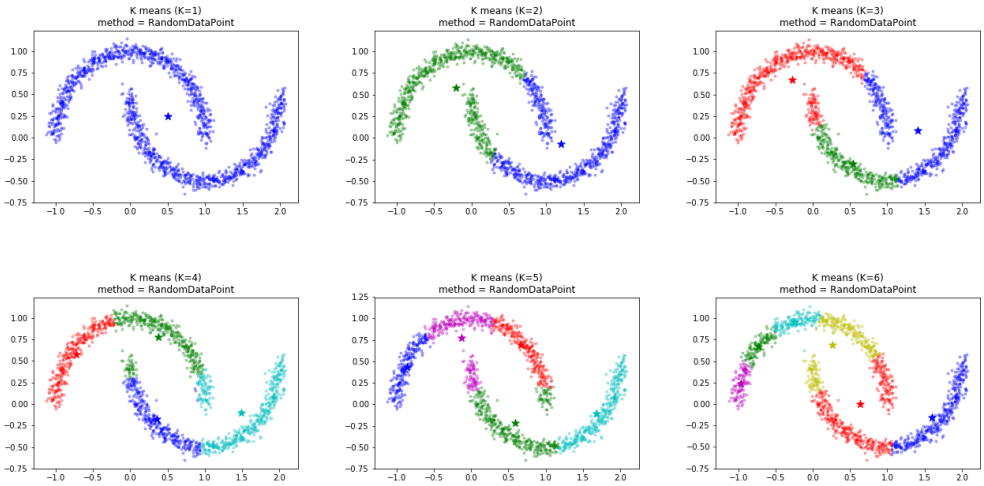
ClusterMedian



Random

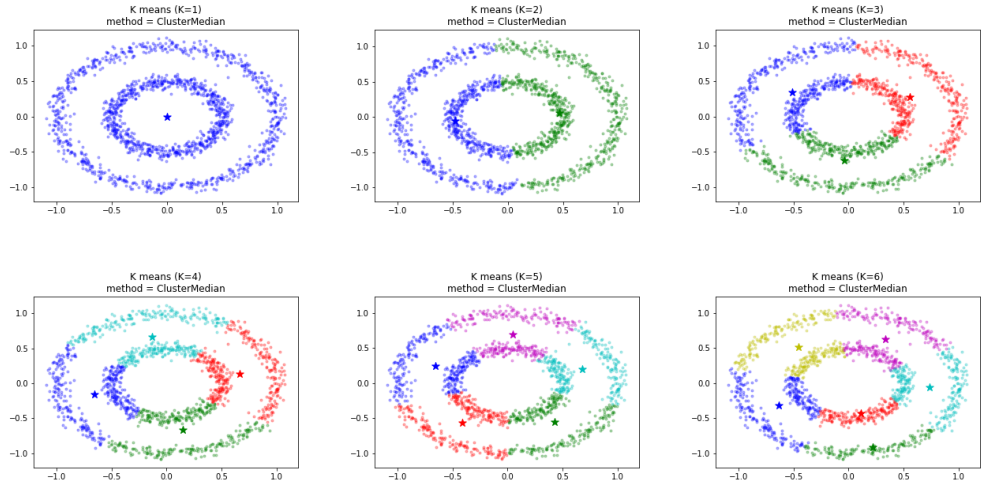


RandomDataPoint

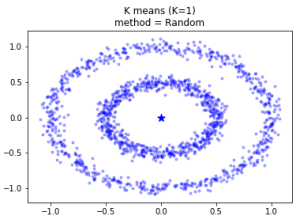


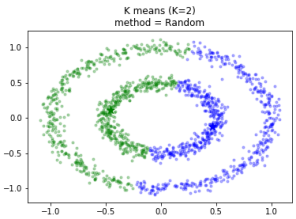
Circle

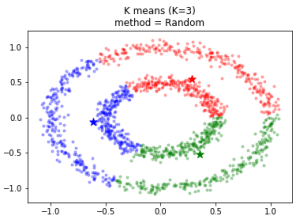
ClusterMedian

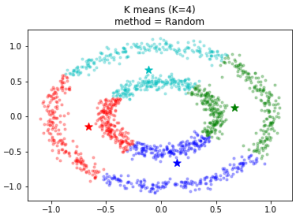


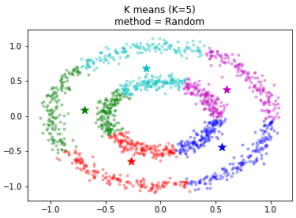
Random

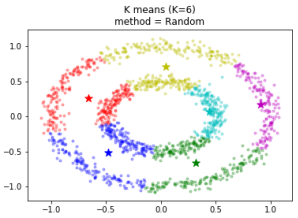




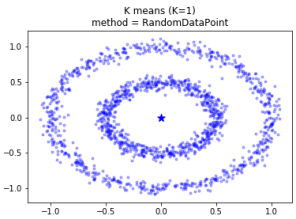


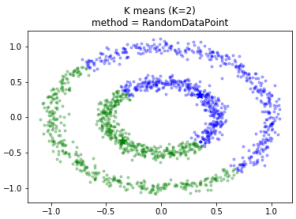


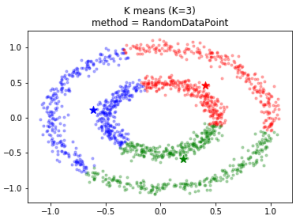


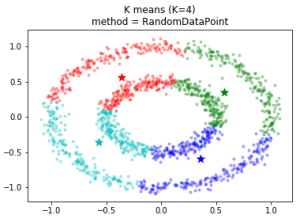


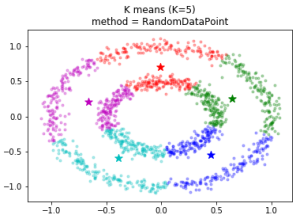
RandomDataPoint

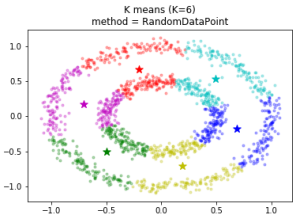








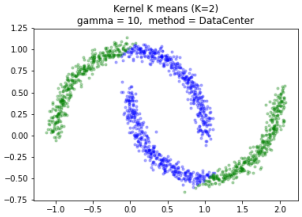


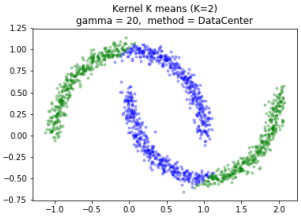


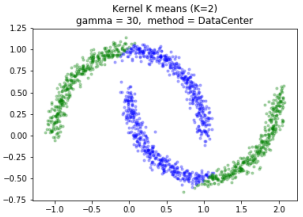
Kernel K means

Moon

DataCenter (k=2)

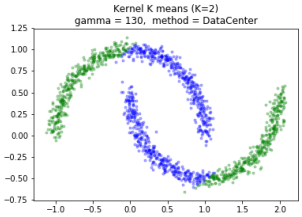


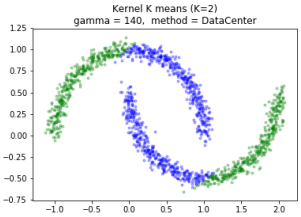


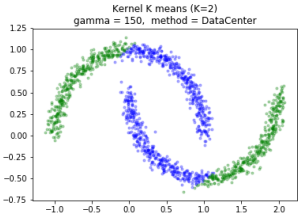


.

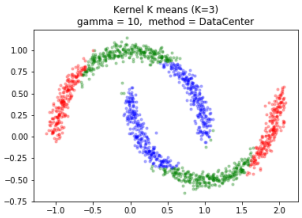
.

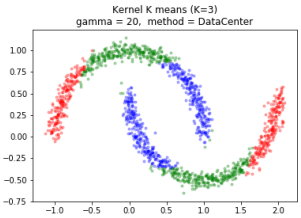


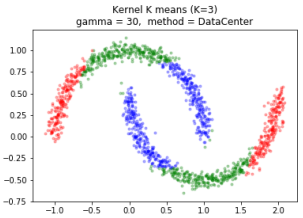




DataCenter (k=3)

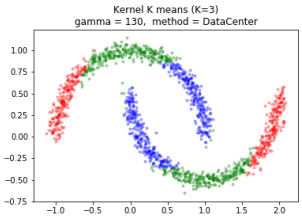


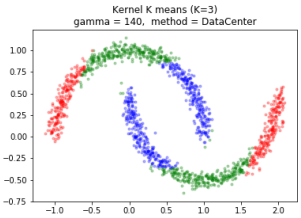


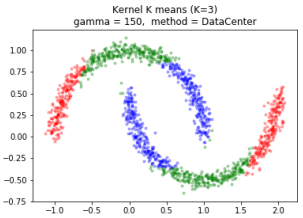


.

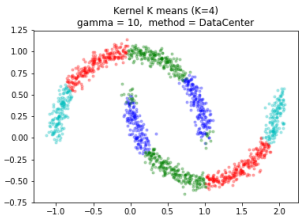
.

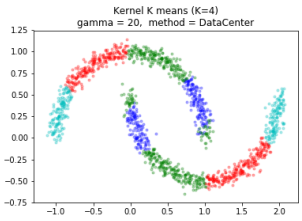


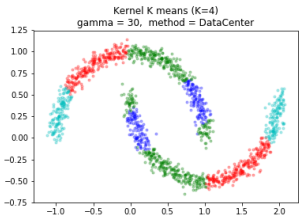




DataCenter (k=4)

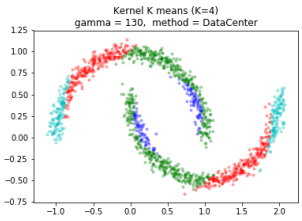


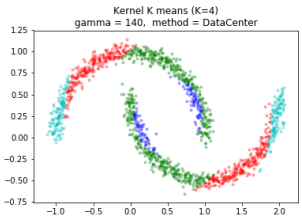


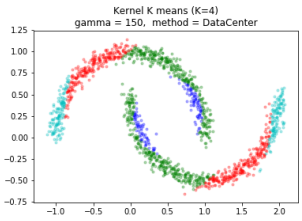


.

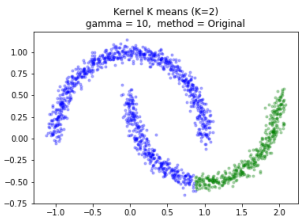
.

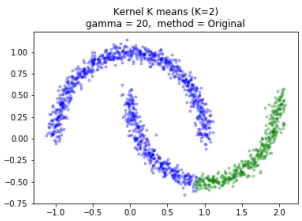


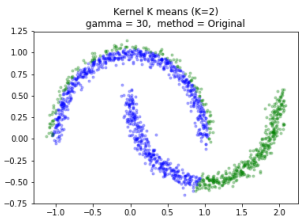




Original (k=2)

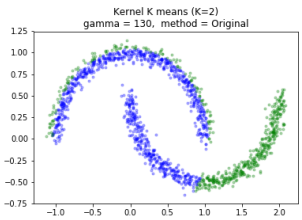


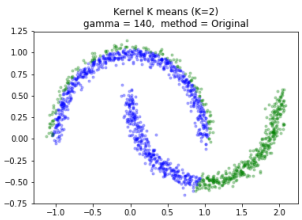


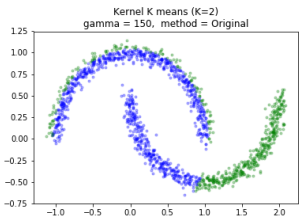


.

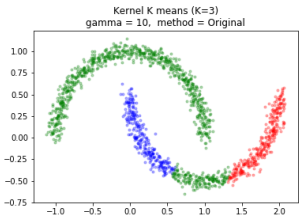
.

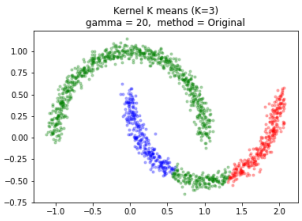


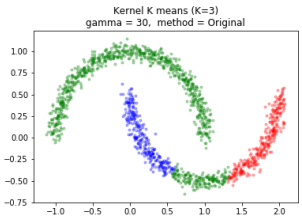




Original (k=3)

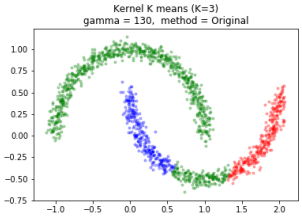


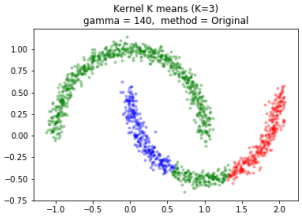


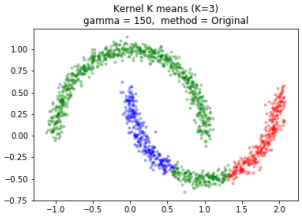


.

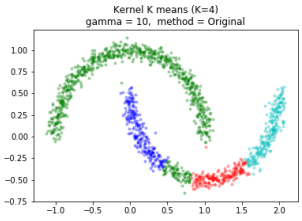
.

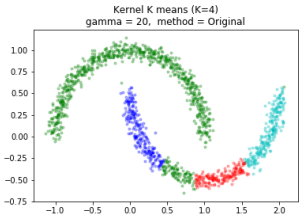


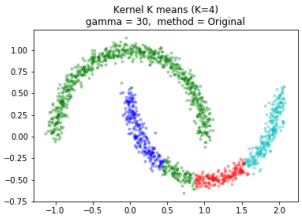




Original (k=4)

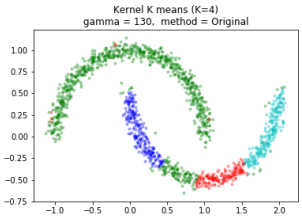


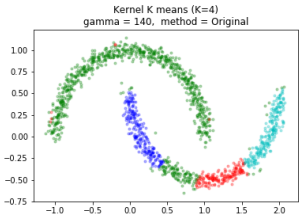


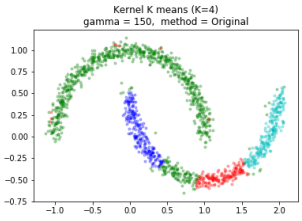


.

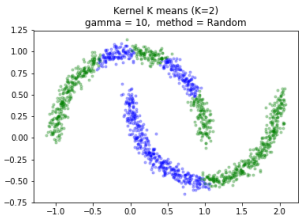
.

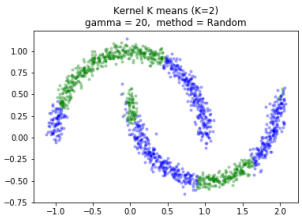


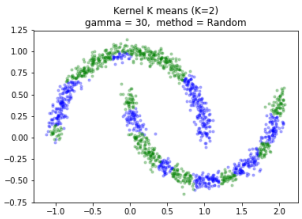




Random (k=2)

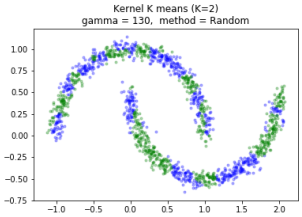


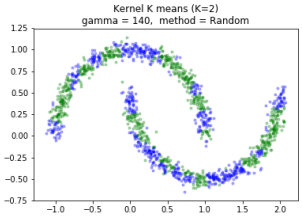


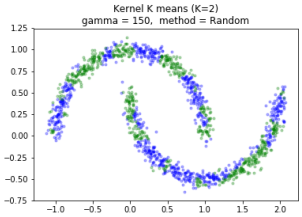


.

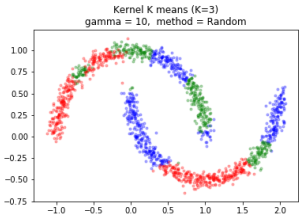
.

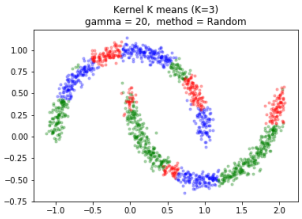


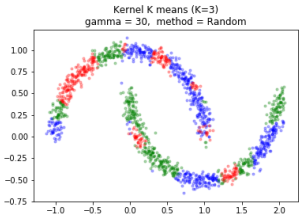




Random (k=3)

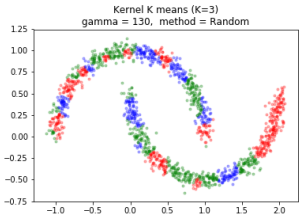


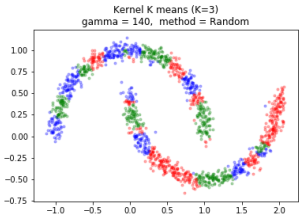


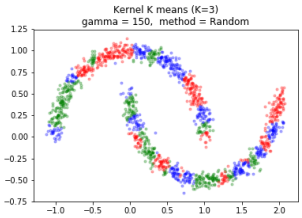


.

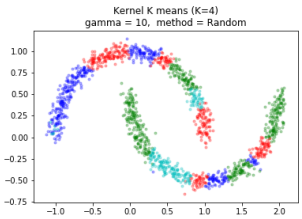
.

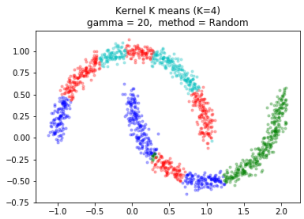


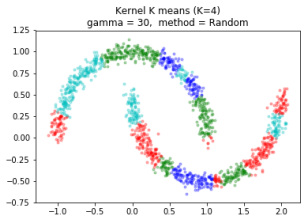




Random (k=4)

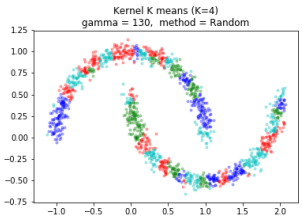


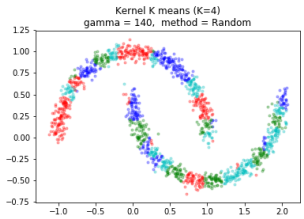


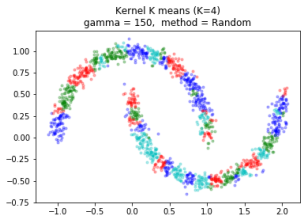


.

.

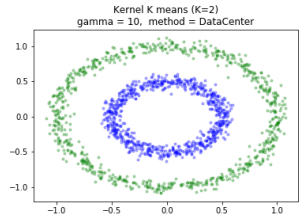


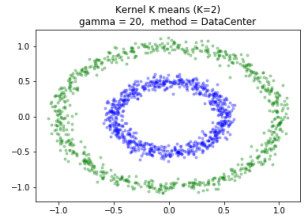


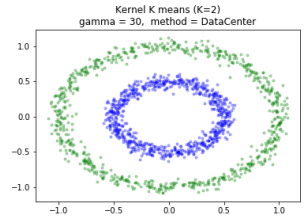


Circle

DataCenter (k=2)

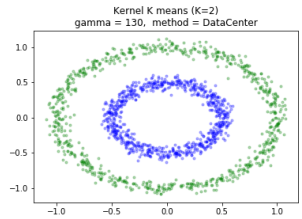


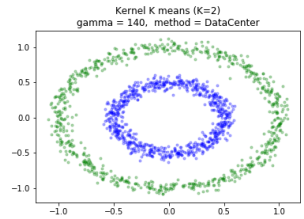


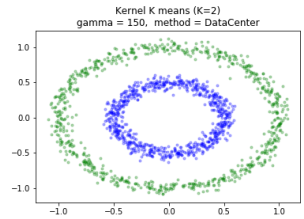


.

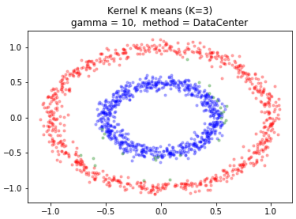
.

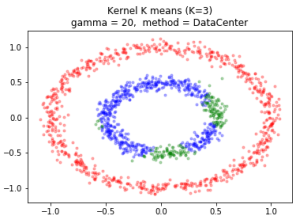


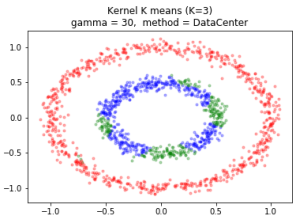




DataCenter (k=3)

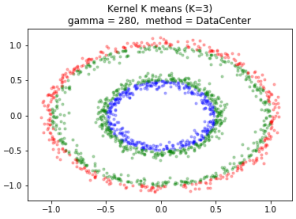


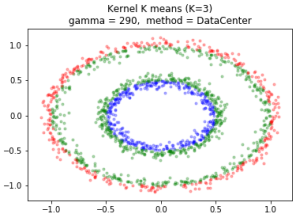


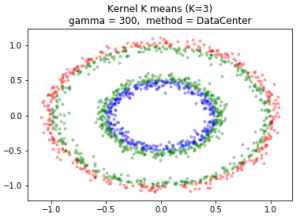


.

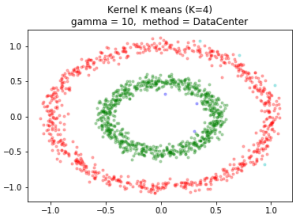
.

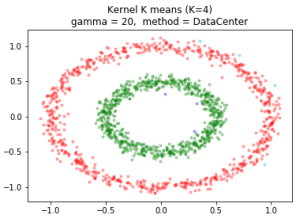


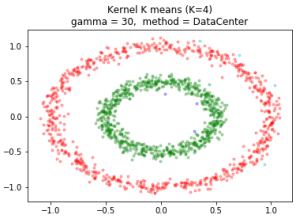




DataCenter (k=4)

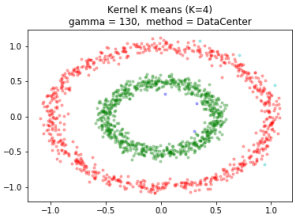


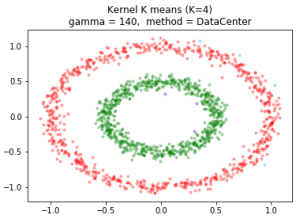


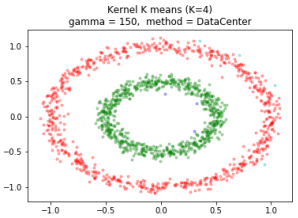


.

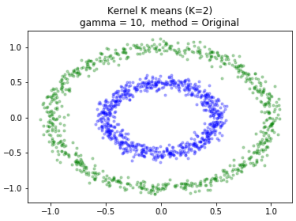
.

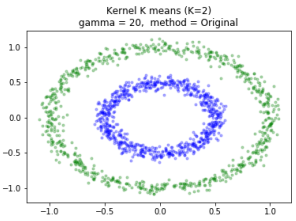


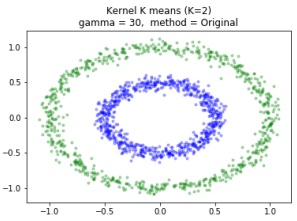




Original (k=2)

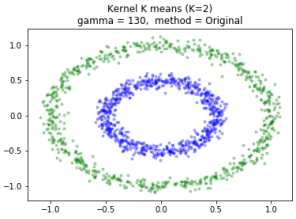


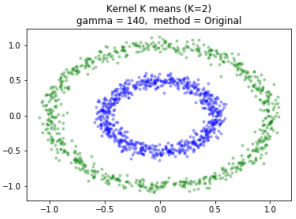


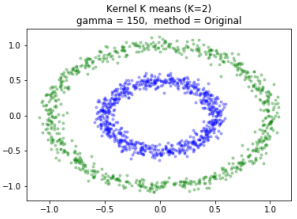


.

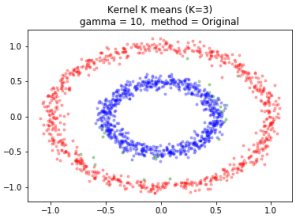
.

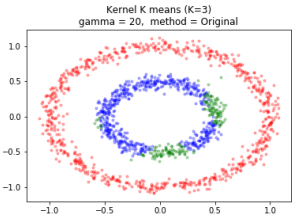


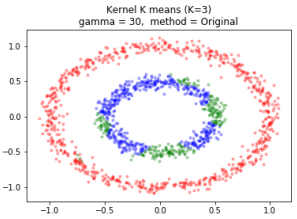




Original (k=3)

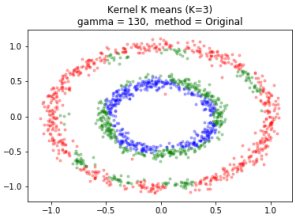


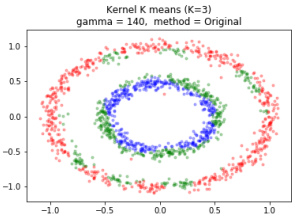


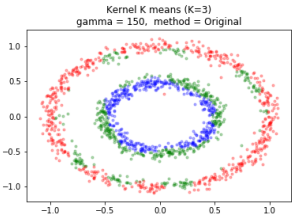


.

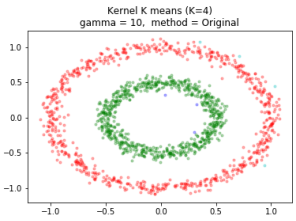
.

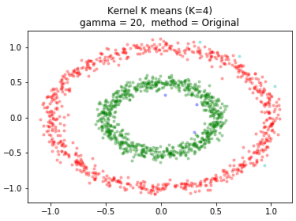


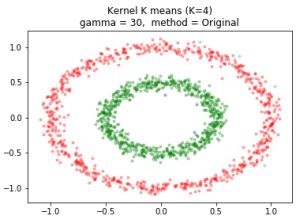




Original (k=4)

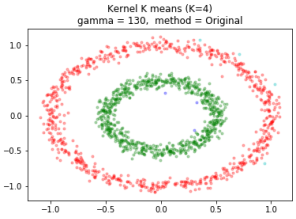


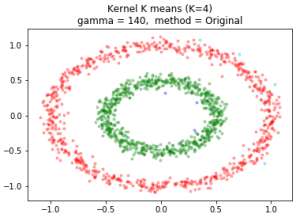


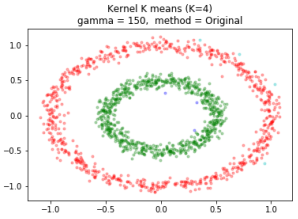


.

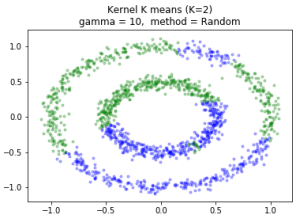
.

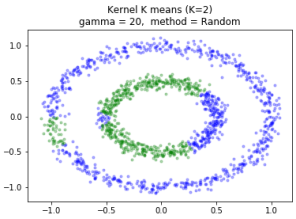


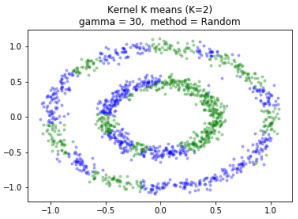




Random (k=2)

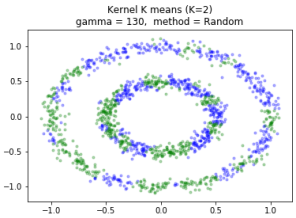


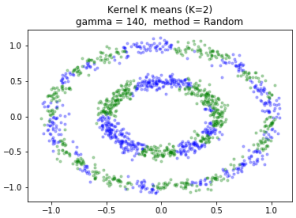


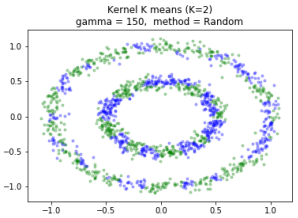


.

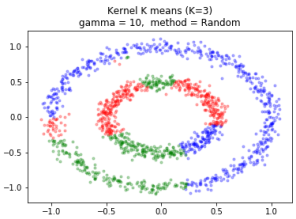
.

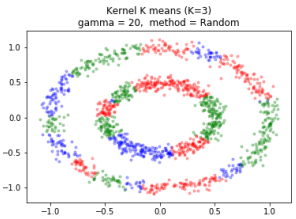


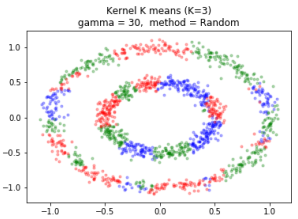




Random (k=3)

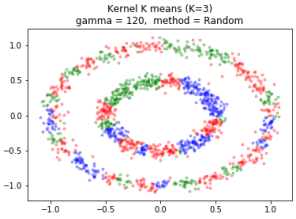


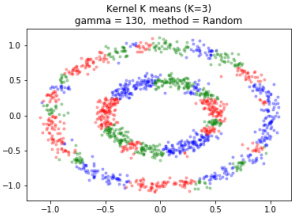


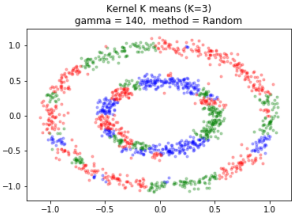


.

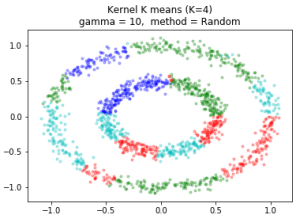
.

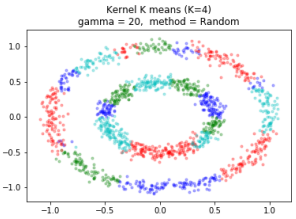


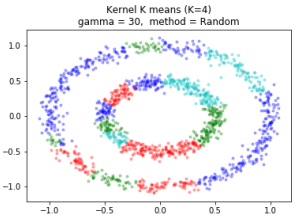




Random (k=4)

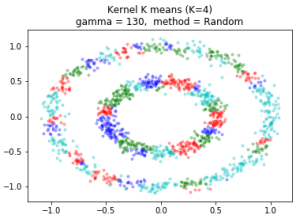


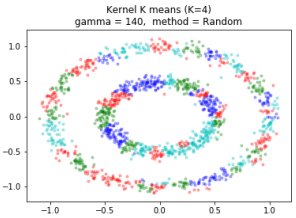


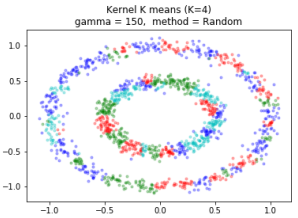


.

.



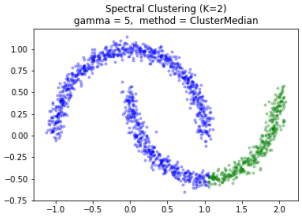


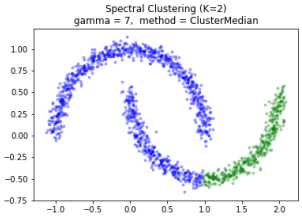


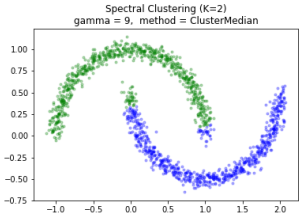
Spectral Clustering

Moon

ClusterMedian (k=2)

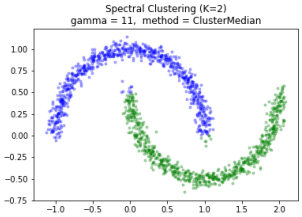


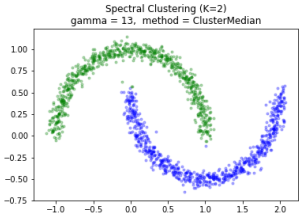


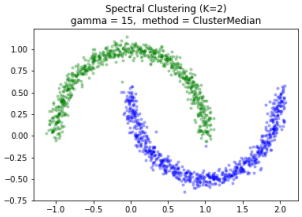


.

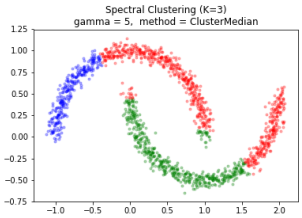
.

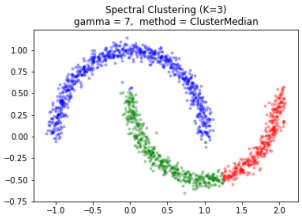


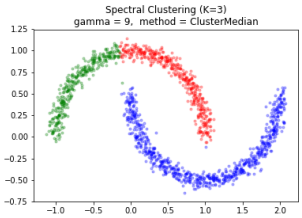




ClusterMedian (k=3)

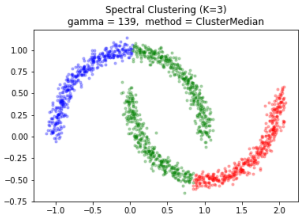


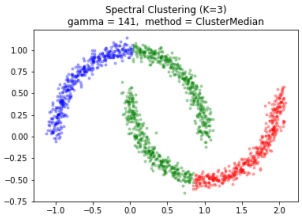


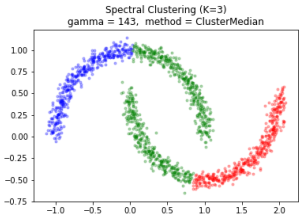


.

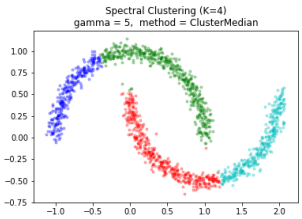
.

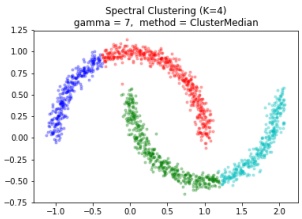


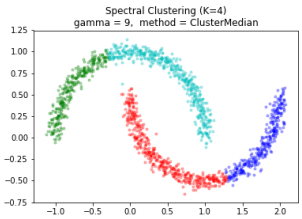




ClusterMedian (k=4)

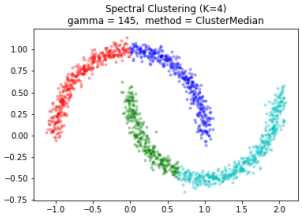


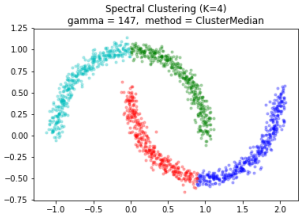


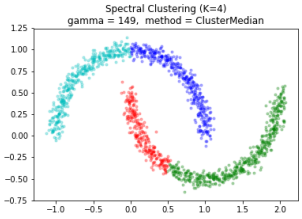


.

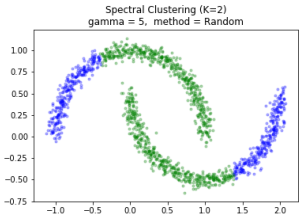
.

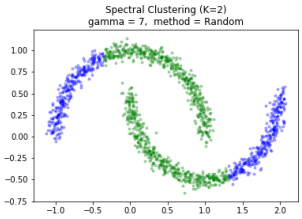


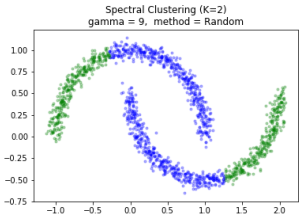




Random (k=2)

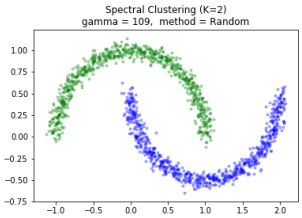


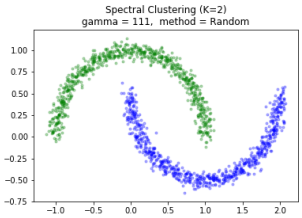


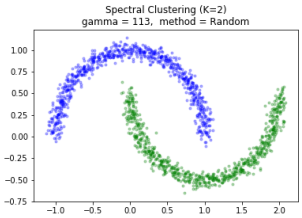


.

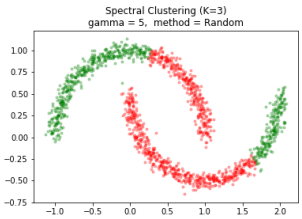
.

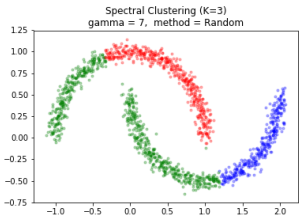


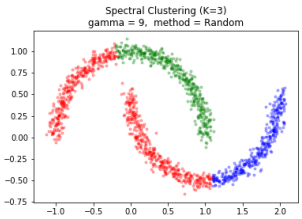




Random (k=3)

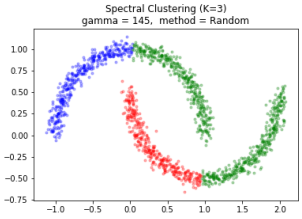


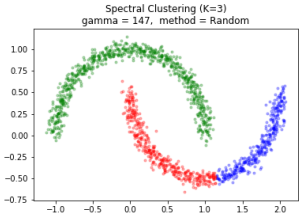


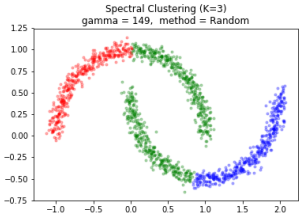


.

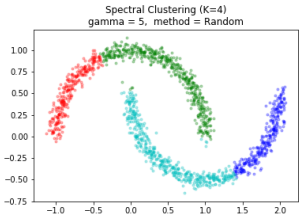
.

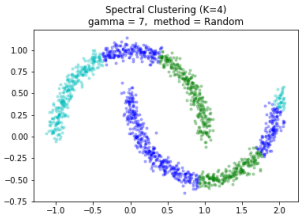


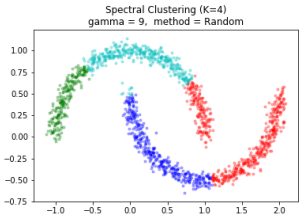




Random (k=4)

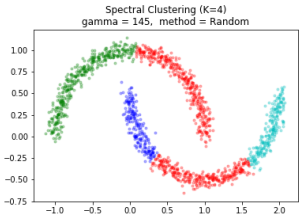


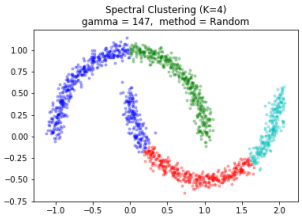


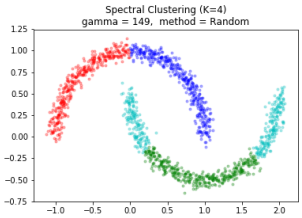


.

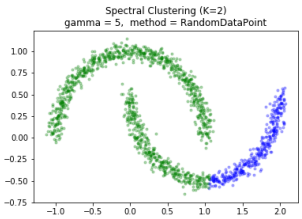
.

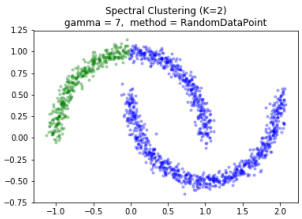


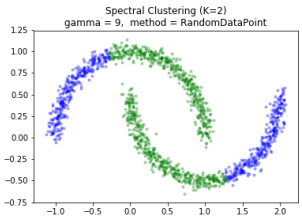




RandomDataPoint (k=2)

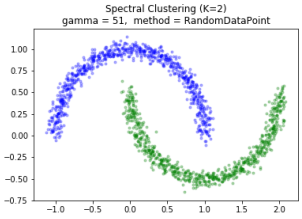


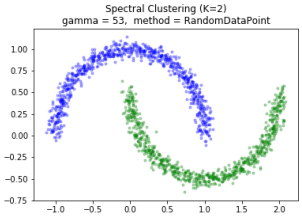


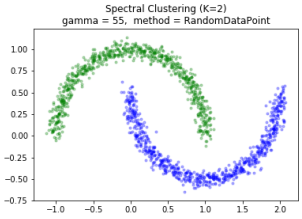


.

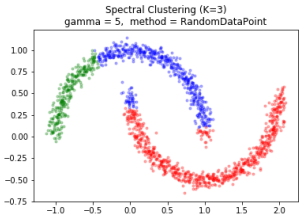
.

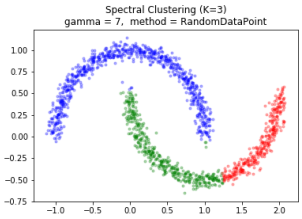


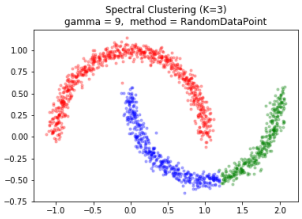




RandomDataPoint (k=3)

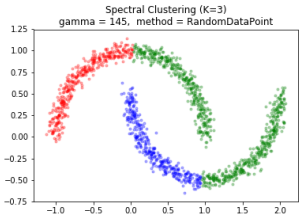


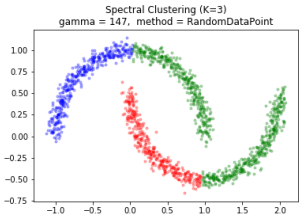


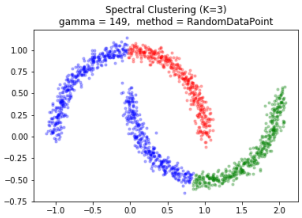


.

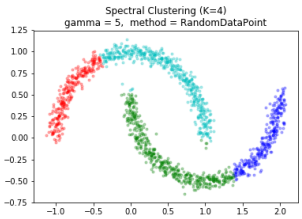
.

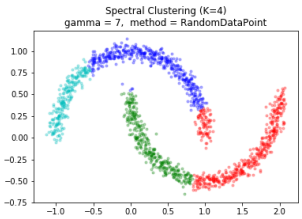


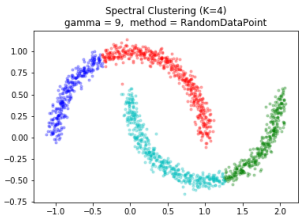




RandomDataPoint (k=4)

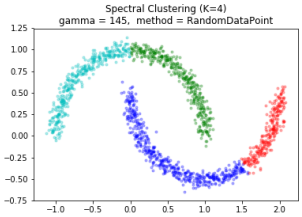


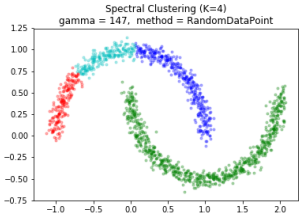


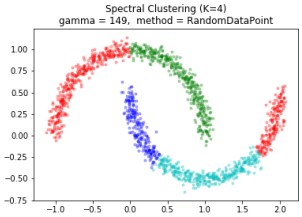


.

.

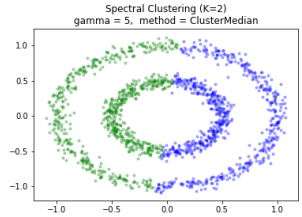


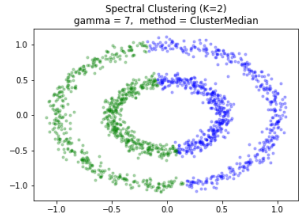


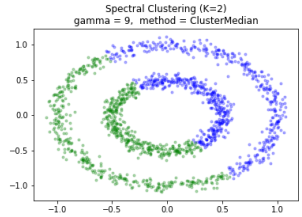


Circle

ClusterMedian (k=2)

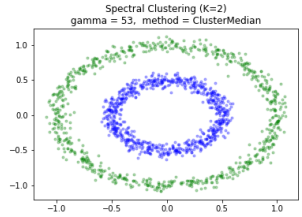


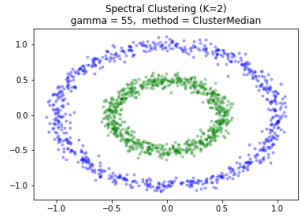


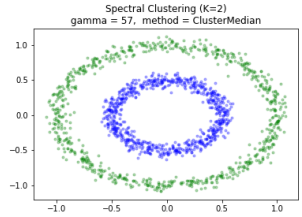


.

.

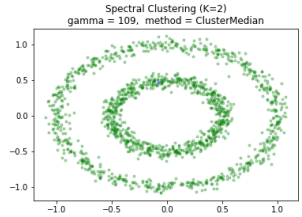


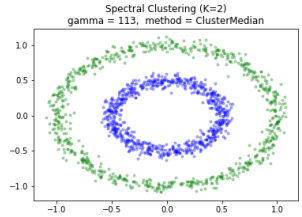


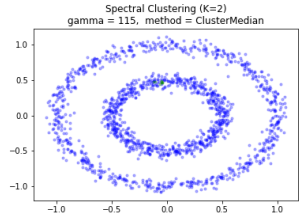


.

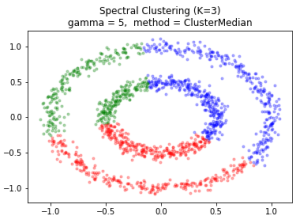
.

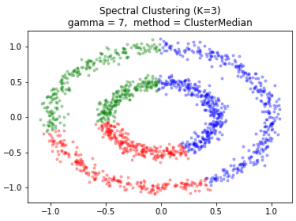


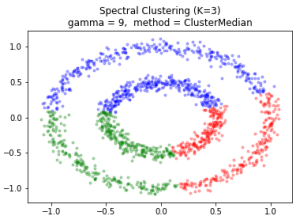




ClusterMedian (k=3)

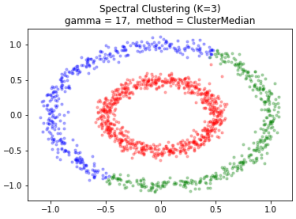


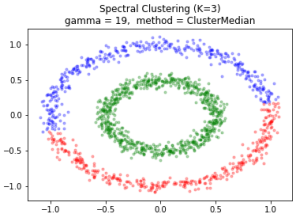


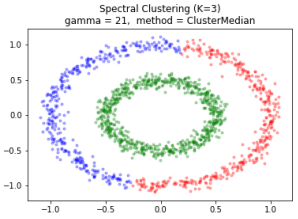


.

.

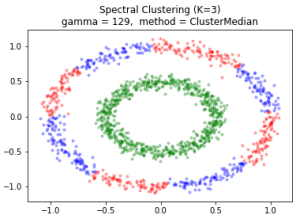


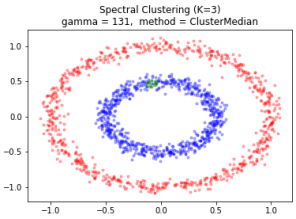


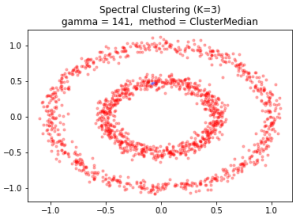


.

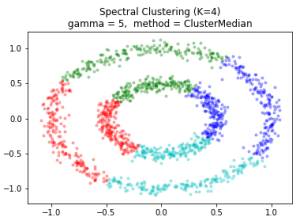
.

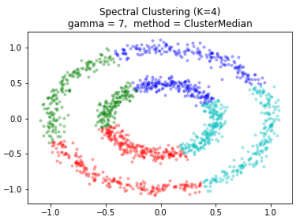


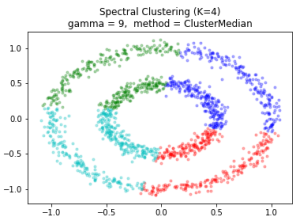




ClusterMedian (k=4)

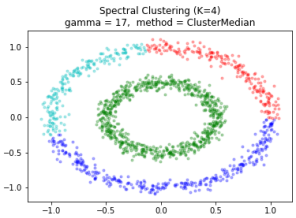


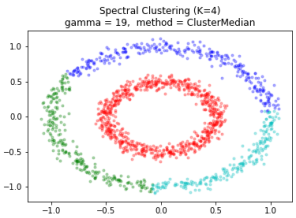


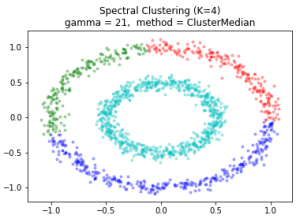


.

.

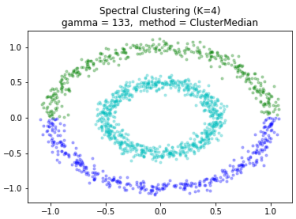


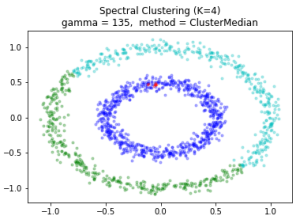


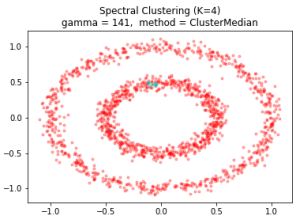


.

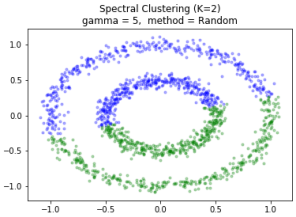
.

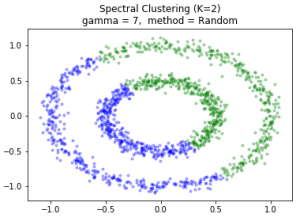

Spectral Clustering (K=4)
gamma = 133, method = ClusterMedian

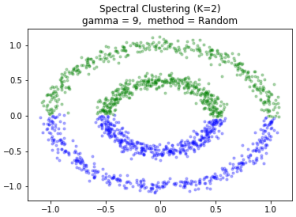

Spectral Clustering (K=4)
gamma = 135, method = ClusterMedian


Spectral Clustering (K=4)
gamma = 141, method = ClusterMedian

Random(k=2)

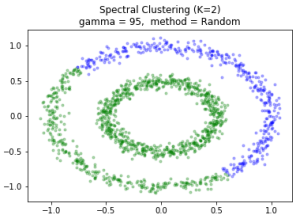

Spectral Clustering (K=2)
gamma = 5, method = Random

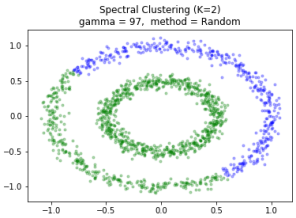

Spectral Clustering (K=2)
gamma = 7, method = Random

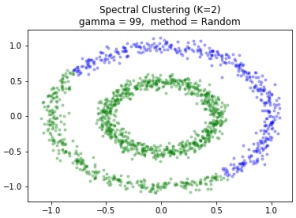

Spectral Clustering (K=2)
gamma = 9, method = Random

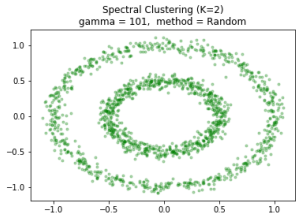
.

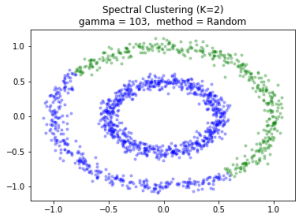
.

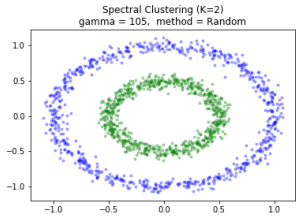

Spectral Clustering (K=2)
gamma = 95, method = Random


Spectral Clustering (K=2)
gamma = 97, method = Random

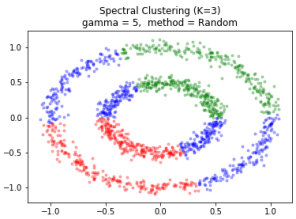

Spectral Clustering (K=2)
gamma = 99, method = Random

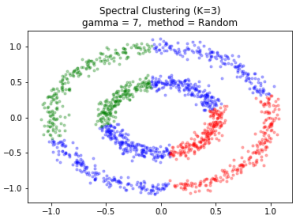

Spectral Clustering (K=2)
gamma = 101, method = Random

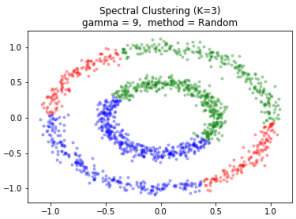

Spectral Clustering (K=2)
gamma = 103, method = Random


Spectral Clustering (K=2)
gamma = 105, method = Random

Random (k=3)

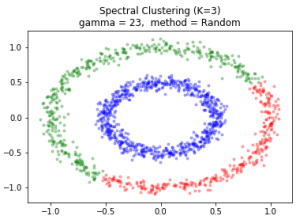


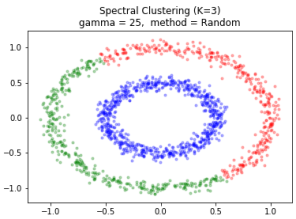


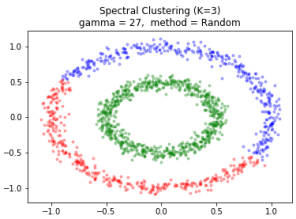


.

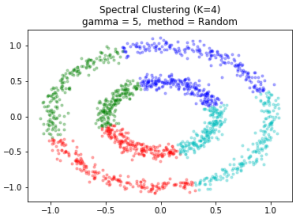
.

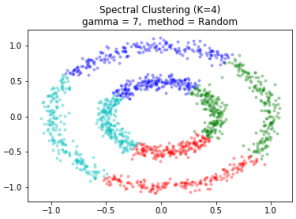


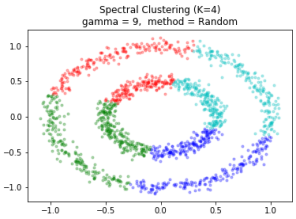




Random (k=4)

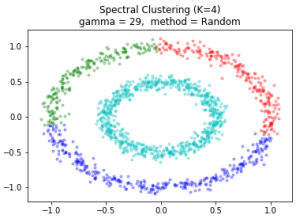


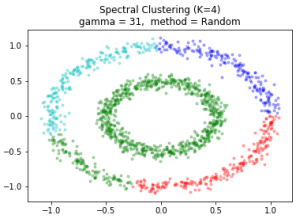


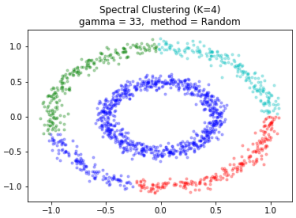


.

.

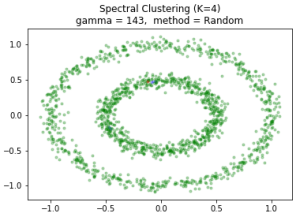


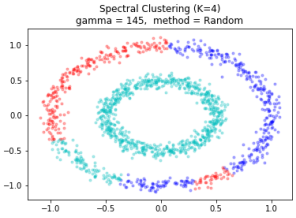


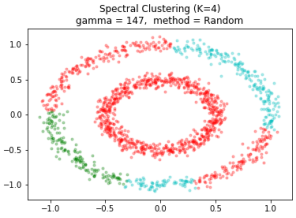


.

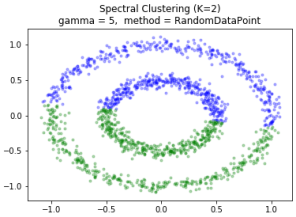
.

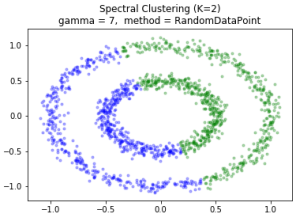


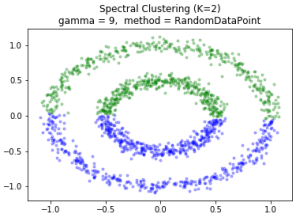




RandomDataPoint (k=2)

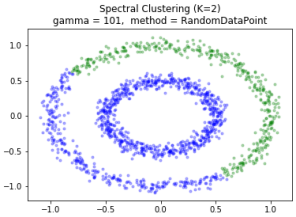


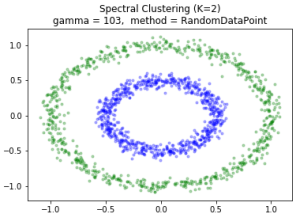


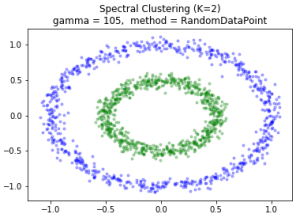


.

.

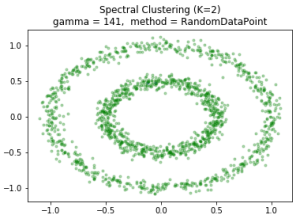


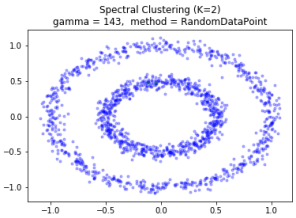


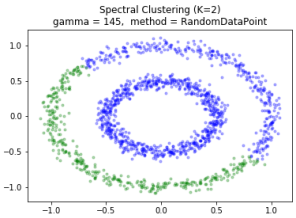


.

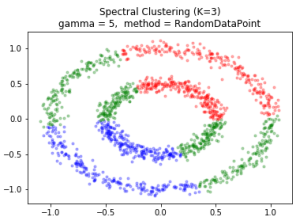
.

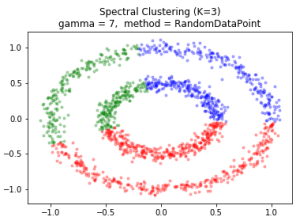


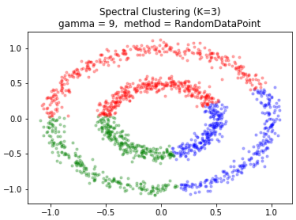




RandomDataPoint (k=3)

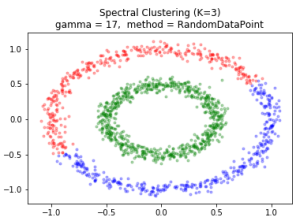


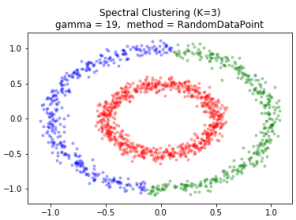


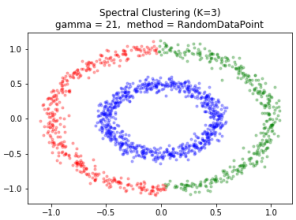


.

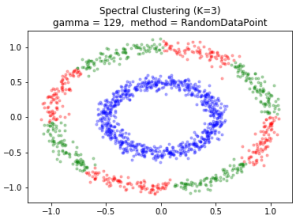
.

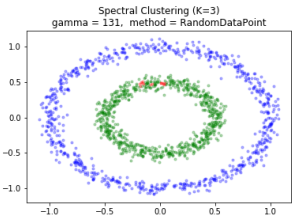


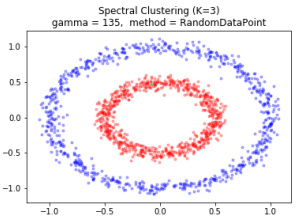




.







RandomDataPoint (k=4)