HW06 KMeans/Kernel KMeans/Spectral Clustering

所有 Clustering 的影片可以在 video 資料夾中找到。

KMeans/Kernel KMeans code

在程式中我把 Kmeans 和 kernel KMeans 寫在同一個 class 中，其主要運算流程是輪流做 Estep(算 data 的 class)跟 Mstep(算新的 mean)：

```
156     def clusterProcess(self):
157         stepIndex=0
158         while(1):
159             self.plotData(stepIndex)
160             stepIndex+=1
161             self.eStep()
162             self.plotData(stepIndex)
163             self.mStep()
164             if self.kernelFunction=='Euclidean':
165                 stepIndex+=1
166             if(self.isEndCondition()):
167                 break
168         self.plotData(stepIndex)
```

Estep：

算這個點跟各個 class 的 mean 的距離，選擇距離最近的作為這次的 class。

Mstep:

根據上次分類的結果，計算新的 mean。

注意在 kernel kmeans 中不會實際運算在 feature space 中的 mean，因為做成 kernel 就是為了避免做 space 轉換的運算。在計算公式中，kernel kmeans 可以直接跳過算 mean 的步驟對各個點做分類。而此處的 mStep()在我的程式中實際功用是儲存 data 值的 Mean(非 feature space 中) 來確認 clustering 是否結束。

```
129     def eStep(self):
130         for dataIndex in range(self.dataSize):
131             minDistance=-1
132             clusterResult=-1
133             for clusterIndex in range(self.clusterCount):
134                 distanceToMean=self.distance(dataIndex,self.mean[clusterIndex,:],clusterIndex=clusterIndex)
135                 #if there is a closer class, choose it
136                 if minDistance==-1 or distanceToMean < minDistance:
137                     minDistance=distanceToMean
138                     clusterResult=clusterIndex
139             self.classArray[dataIndex]=clusterResult
140         return
141     def mStep(self):
142         #save oldmean to determine end conditions in isEndCondition()
143         self.oldMean=np.copy(self.mean)
144         for clusterIndex in range(self.clusterCount):
145             dataClusterIndices=np.where(self.classArray==clusterIndex)[0]
146             self.mean[clusterIndex,:]=calculateMean(self.dataArray[dataClusterIndices,:])
```

這就是確認是否結束 clustering，若結束的話 mean 基本上不會有任何改變。
END_DIFFERENCE 我定義為 1e-3。

```
147     def isEndCondition(self):
148         if np.sum(np.absolute(self.oldMean-self.mean)) < self.END_DIFFERENCE:
149             return 1
150         return 0
```

接下來是計算距離的函式

```
102 def distance(self, aPointIndex, meanPoint,clusterIndex):
103     if self.kernelFunction=='Euclidean':
104         resultDistance=np.sum(np.absolute(self.dataArray[aPointIndex,:]-meanPoint))
105     elif self.kernelFunction=='RBF':
106         selfDistance=self.gramMatrix[aPointIndex,aPointIndex]
107         neighborDistanceSum=0
108         neighborGramMatrixSum=0
109         dataIndices=np.where(self.classArray == clusterIndex)[0]
110         clusterDataCount=dataIndices.shape[0]
111         for dataIndex in dataIndices:
112             neighborDistanceSum+=self.gramMatrix[aPointIndex,dataIndex]
113             for dataIndex2 in dataIndices:
114                 neighborGramMatrixSum += self.gramMatrix[dataIndex, dataIndex2]
115         resultDistance=selfDistance-2*neighborDistanceSum/max(clusterDataCount,1e-6)+neighborGramMatrixSum/(max(clusterDataCount,1e-6)**2)
116     #print(resultDistance)
117     return resultDistance
```

若是 KMeans 的話就單純算點與點的距離。
若是用 RBF 算距離，結果對應的是這個

$$\left\|\phi(x_j)-\mu_k^{\phi}\right\| = \left\|\phi(x_j)-\frac{1}{|C_k|}\sum_{n=1}^{N}\alpha_{kn}\phi(x_n)\right\|$$
$$= \mathbf{k}(x_j,x_j)-\frac{2}{|C_k|}\sum_{n}\alpha_{kn}\mathbf{k}(x_j,x_n)+\frac{1}{|C_k|^2}\sum_{p}\sum_{q}\alpha_{kp}\alpha_{kq}\mathbf{k}(x_p,x_q)$$

第二項 Sigma 對應的是 neighborDistanceSum，第三項對應的是 neighborGramMatrixSum。其中加了一個最小量 1e-6 以免出現除 0 的狀況。
其中的 self.gramMatrix 就是講義中的符號 W，含點與點的 similiarity，在_init_中就會先算好，如以下。

```
61      def __init__(self,dataArray,clusterCount=3,gamma=80,kernelFunction='Euclidean',name="KMmeansCluster",saveImages=True):
```

```
81          if kernelFunction=='RBF':
82              self.gramMatrix=np.zeros((self.dataSize,self.dataSize))
83              print("Initializing Gram Matrix...")
84              for dataIndex in range(self.dataSize):
85                  #show process
86                  processPercentage=int(dataIndex*100/self.dataSize)
87                  if processPercentage%10==0:
88                      print("progress: {} %".format(processPercentage))
89                  for dataIndex2 in range(dataIndex,self.dataSize):
90                      self.gramMatrix[dataIndex][dataIndex2]=kernelRBF(self.dataArray[dataIndex],self.dataArray[dataIndex2],gamma=gamma)
91                      self.gramMatrix[dataIndex2][dataIndex]=self.gramMatrix[dataIndex][dataIndex2]
```

最後要提的是把圖片存下來的部分

```
154     def plotData(self,imageIndex,forceSaveImage=False):
155         if forceSaveImage==False and self.saveImages == False:
156             return
157         fig = plt.figure()
158         ax = fig.add_subplot(111)
159         dataX=self.dataArray[:,0]
160         dataY=self.dataArray[:,1]
161         scatter = ax.scatter(dataX, dataY, c=self.classArray, s=50)
162         for clusterIndex in range(self.clusterCount):
163             ax.scatter(self.mean[clusterIndex,0], self.mean[clusterIndex,1], s=50, c='red', marker='+')
164         ax.set_xlabel('x')
165         ax.set_ylabel('y')
166         plt.colorbar(scatter)
167         imageName=self.name+'_'+str(imageIndex)+'.png'
168         folderPath=Path('images/'+self.name)
169         print(imageName)
170         if not os.path.exists(str(folderPath)):
171             os.mkdir(str(folderPath))
172         fig.savefig(str(folderPath / imageName))
```

我使用的是 matplotlib，把當下的狀態圖片存下來，之後再手動做成 gif。

Spectral Clustering Code

```
class SpectralCluster:
    def __init__(self,dataArray,clusterCount=3,gamma=10,name="SpectralClustering",saveEigenImage=False):
        self.dataArray=dataArray
        self.clusterCount=clusterCount
        self.saveEigenImage=saveEigenImage
        self.gamma=gamma
        self.name=name
        #to be more comprehensive
        self.dataSize,self.dataDimension=self.dataArray.shape
        self.graphMatrix = np.zeros((self.dataSize, self.dataSize))
        self.degreeMatrix = np.zeros((self.dataSize,self.dataSize))
        self.graphLaplacian = np.zeros((self.dataSize,self.dataSize))
        print("Initializing Graph Matrix...")
        for dataIndex in range(self.dataSize):
            #show progess
            processPercentage = int(dataIndex * 100 / self.dataSize)
            if processPercentage % 10 == 0:
                print("progress: {} %".format(processPercentage))
            #calculate RBF
            for dataIndex2 in range(dataIndex, self.dataSize):
                self.graphMatrix[dataIndex,dataIndex2] = kernelRBF(self.dataArray[dataIndex], self.dataArray[dataIndex2],
                                                                   gamma=gamma)
                self.graphMatrix[dataIndex2,dataIndex] = self.graphMatrix[dataIndex,dataIndex2]
        print('Calculating degree and graph laplacian...')
        flatDegree=np.sum(self.graphMatrix,axis=1)
        #calculate degree D
        for dataIndex in range(self.dataSize):
            self.degreeMatrix[dataIndex,dataIndex]=flatDegree[dataIndex]
        #calculate L= D-W
        self.graphLaplacian=self.degreeMatrix-self.graphMatrix
```

上面是一些變數，中間式計算 W 的值和 Kernel Kmeans 一模一樣，後面算 D 是將各軸加起來然後放到斜角上，graph laplacian 用的是 D-W。

```
        print('Calculating eigenvalues and eigenvectors')
        self.eigenValue,self.eigenVector=lng.eig(self.graphLaplacian)
        #get first few eigenvalue and eigenvectors
        sortIndices=self.eigenValue.argsort()
        self.eigenValue=self.eigenValue[sortIndices[:self.clusterCount]]
        for index in range(self.eigenValue.shape[0]):
            if self.eigenValue[index]<1e-10:
                self.eigenValue[index]=0
        self.eigenVector=self.eigenVector[:,sortIndices[:self.clusterCount]]
        #self.eigenSpace=np.transpose(self.eigenVector)\
        print('Start Kmean cluster...')
        self.kmeansSolver=KMeansCluster(dataArray=self.eigenVector,clusterCount=self.clusterCount,gamma=self.gamma,kerne
        self.kmeansSolver.clusterProcess()
        #plot result
        self.kmeansSolver.dataArray=self.dataArray
        self.kmeansSolver.plotData(-1,forceSaveImage=True)
```

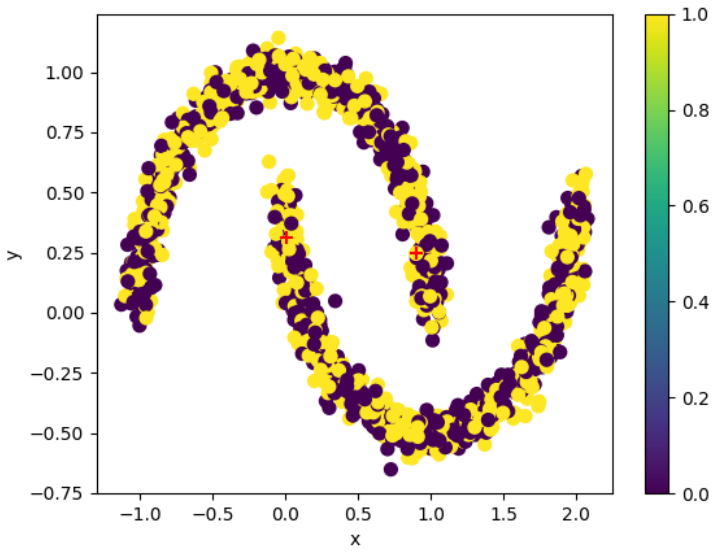
算 eigenvector 和 eigenvalue 用的是 numpy 中的 linalg，我使用 argsort 選到前 k 個(clusterCount 個)eigenvector，再丟到更前面的 KMeans class 中解完，最後面是為了產生分類完的圖，將 eigenvector 替換成原本的 data 再 plot 一次。

Initialization

主要可以調整的 Initialization 有兩個，第一個是各個資料點的初始 class，第二個是 mean 點的決定。

初始 class

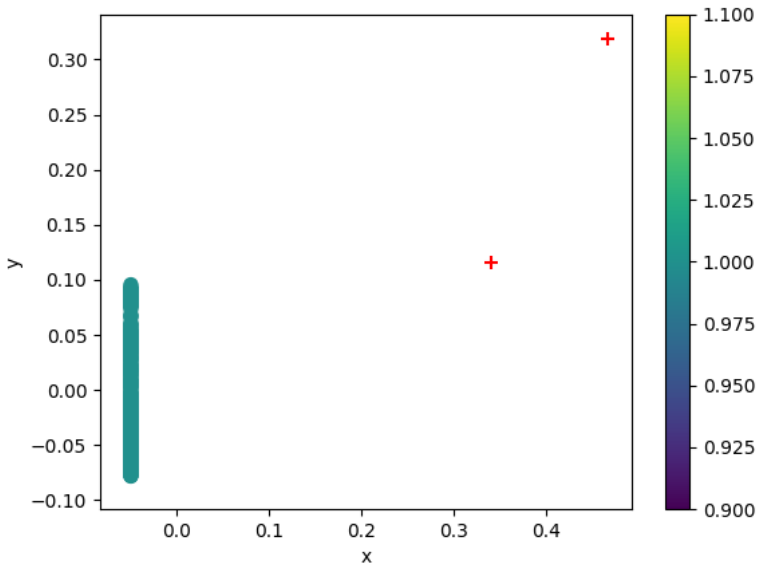
因為 KMeans 和 Kernel KMeans 都會先對點做分類的步驟，因此這個的調整對他們沒有用處，都設為 0 也沒關係。
在 Spectral Clustering 中因為不會有直接計算 mean 的階段，所以這個的初始值就很重要。



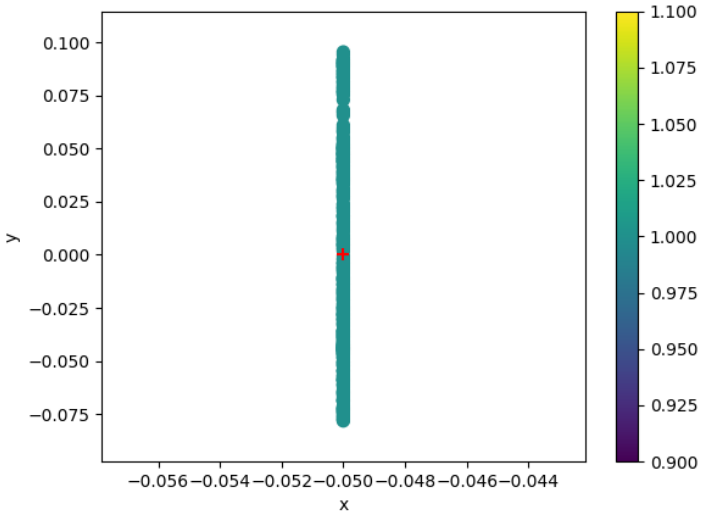
這是最沒有爭議的選擇，因為對資料點沒有了解所以讓他隨機分類。這一共花了 3 步到達收斂。
若是對資料有點概念的話，若將部分的知識導入是否可以加快收斂速度?像是這張圖我們會想要分上下部分，因此我讓>0.5 的歸類為 0，<-0.25 的歸類為 1，其他隨機。這樣的方法讓他同樣花 3 步到收斂，沒有差別。
之後試著讓 class 失衡，若是 10:1 的比例產生完全不會影響步數。
全部指定給一個 class 他也只花 5 步。
由此可見對於 spectral clustering 來說這部分的 initialization 只要不是極端值就沒問題。

初始 Mean

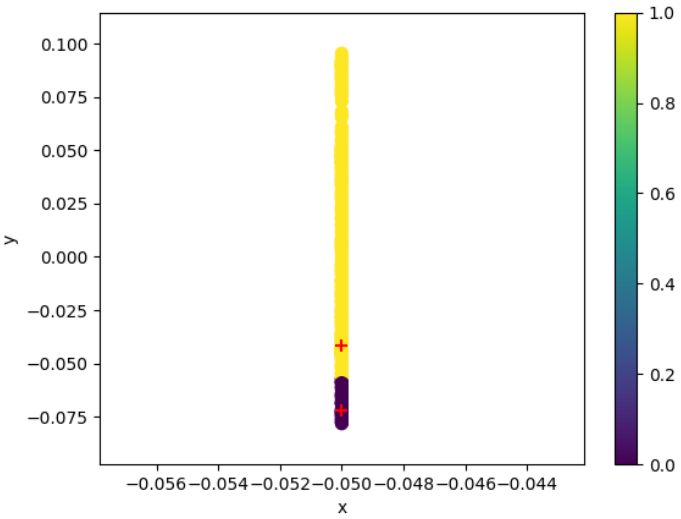
最先使用的是在一個範圍內隨機產生 mean 點，另一個是從 data 中選取一個點作為 mean。在 KMeans 中這兩個方法並不會造成明顯的收斂時間變化，但是後者會是個比較好的方法。



這張圖是在 Spectral clustering 中出現的，可以發現到因為一個 mean 明顯比較遠，最後結果就一定會變成全部的點被分類到同一個 class：

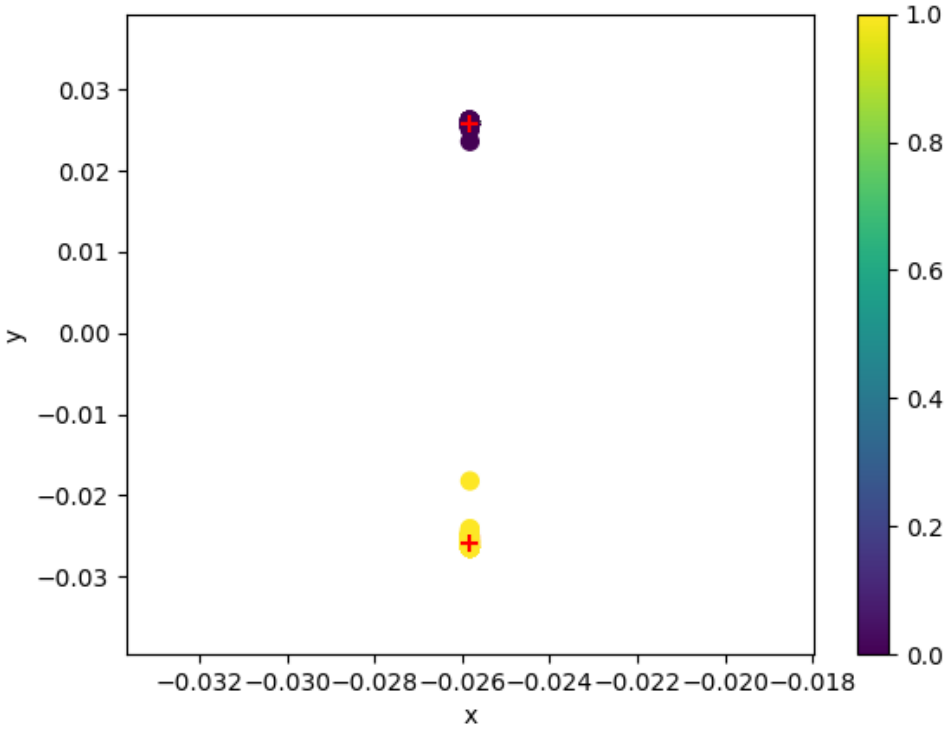


這不是我們想要的，但若將 mean 選在 data 中的話會得到：

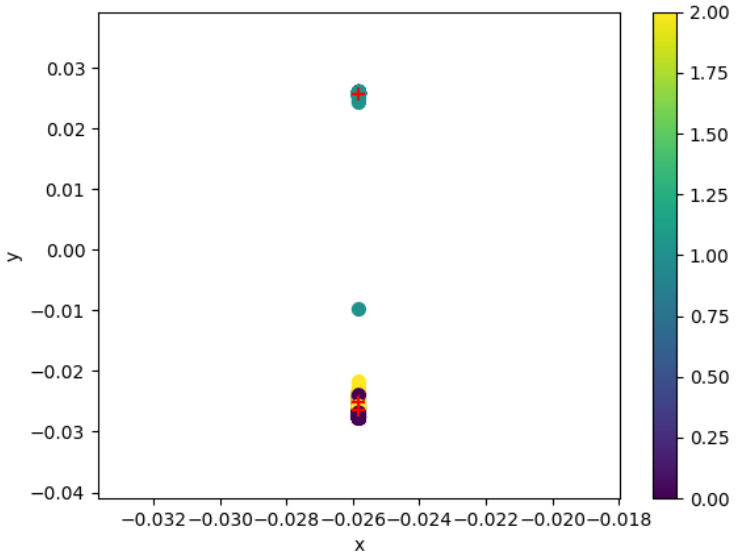


最後會平分這條線，這樣比較好。
另外要提一下雖然這種連成一條線的狀況是因為 gamma 值過小所造成的，但若用正確的 gamma 值仍可能遇到這種問題！

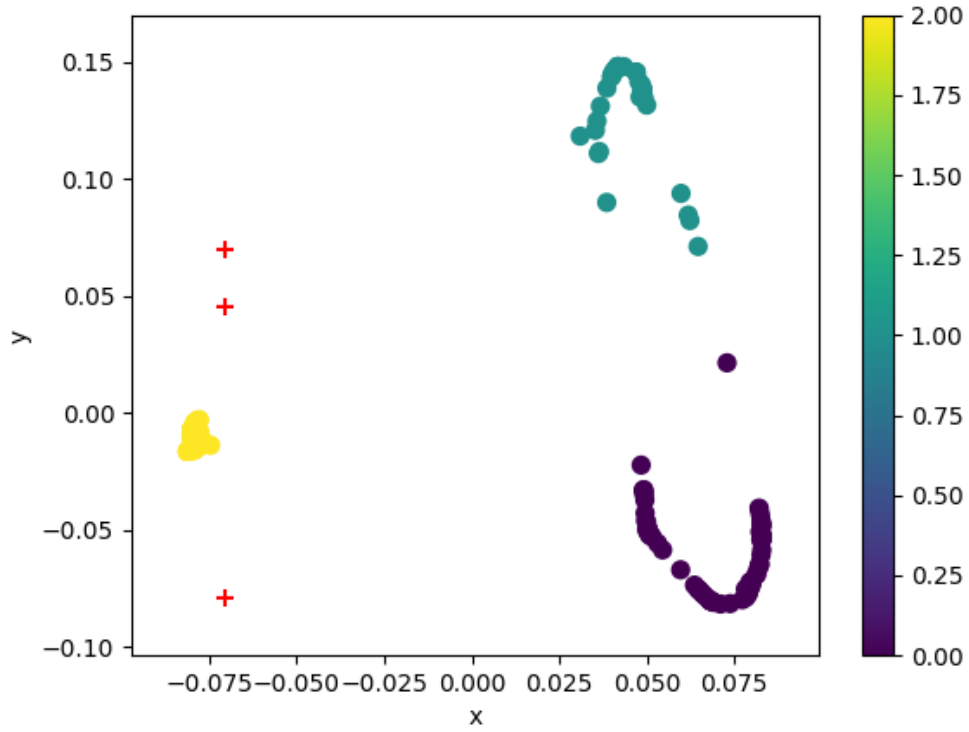
In EigenSpace



從這張圖中我們可以觀察到 eigenvector 的確把資料點分開來使得 kmeans 容易運作。接下來看一張有 3 個 cluster 的。第 0 個 cluster 的確是有被分開來，但 1 和 2 卻混雜在一起。



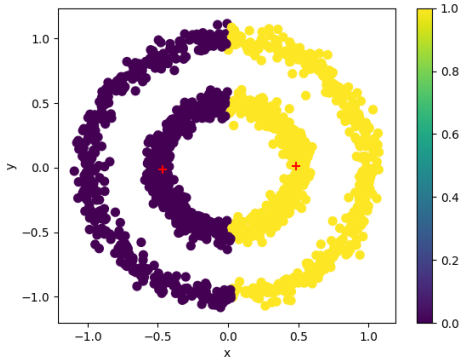
但一看到第三維的時候就會發現他有明顯的隔開，這就是 SpectralClustering 所想要達到的效果。

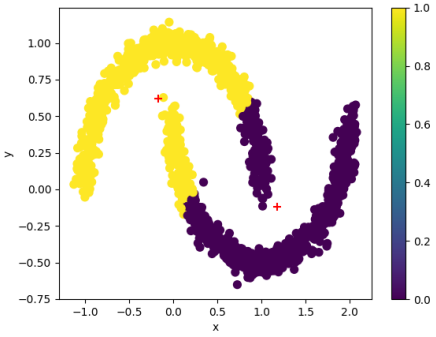


RESULTS

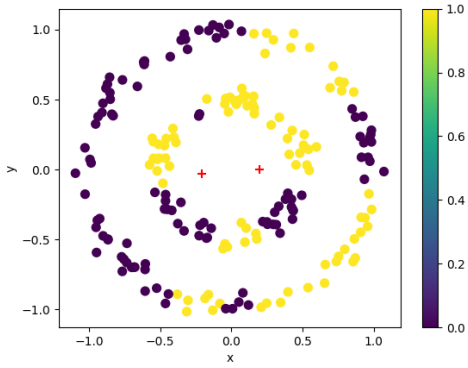
2 CLUSTERS

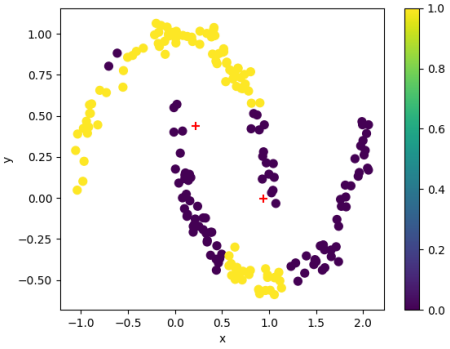
Kmeans



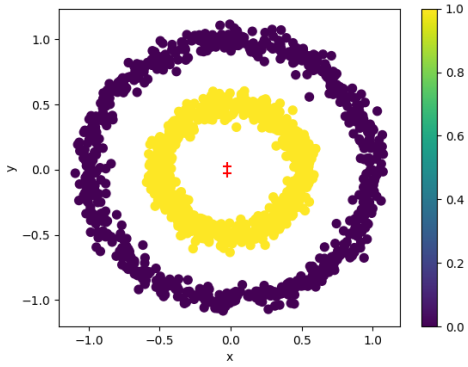


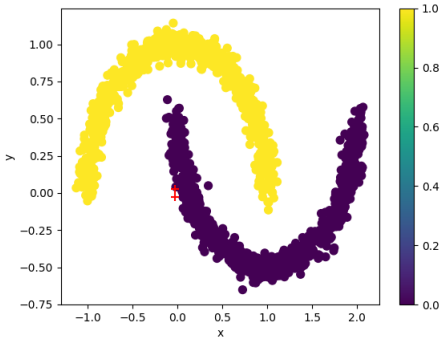
Kernel Kmeans





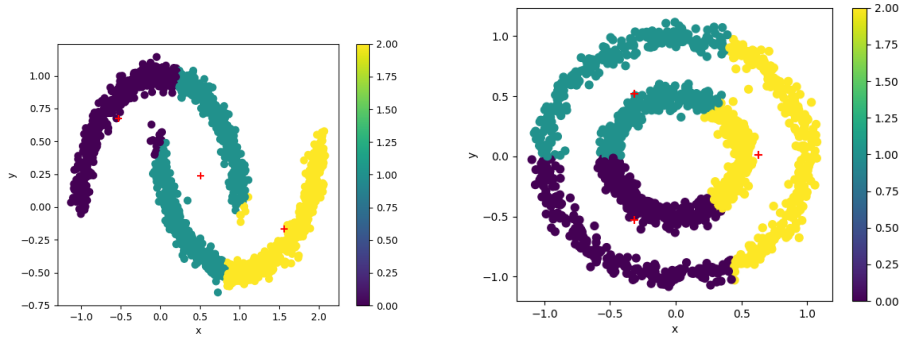
Spectral Clustering



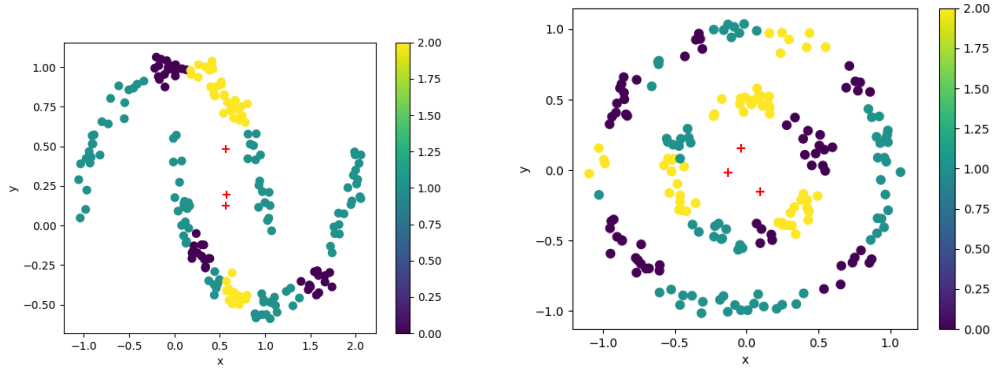


3 CLUSTERS

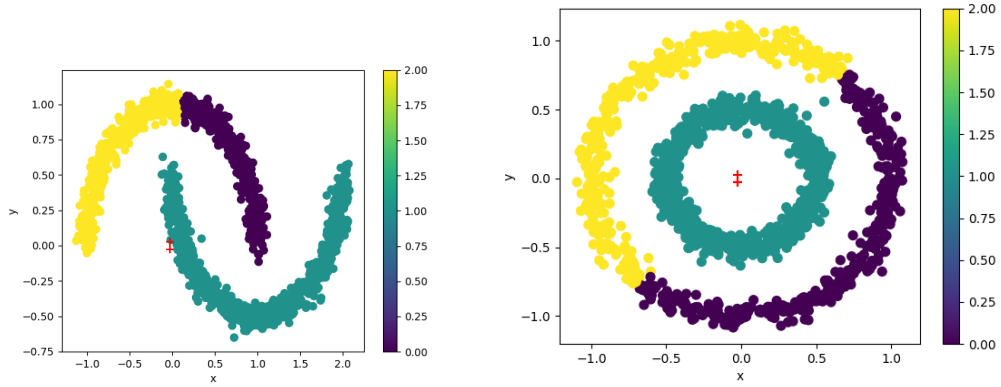
Kmeans



Kernel Kmeans

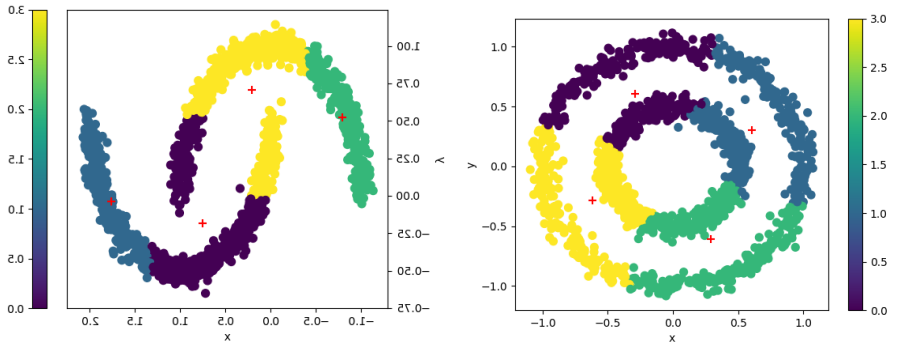


Spectral Clustering

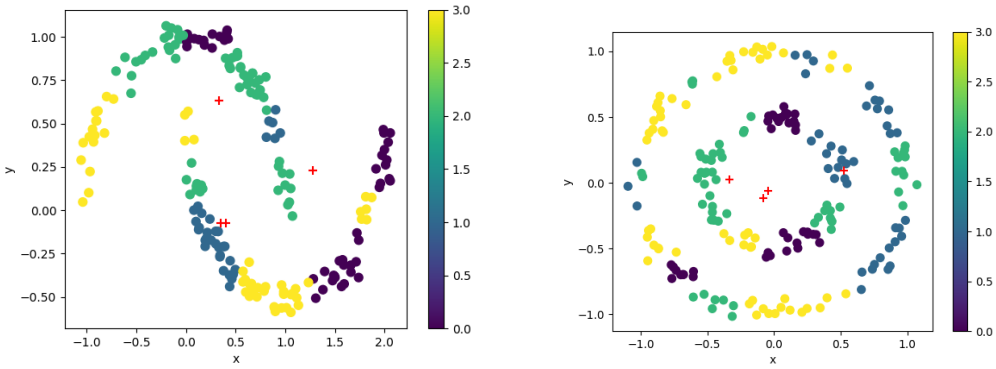


4 CLUSTERS

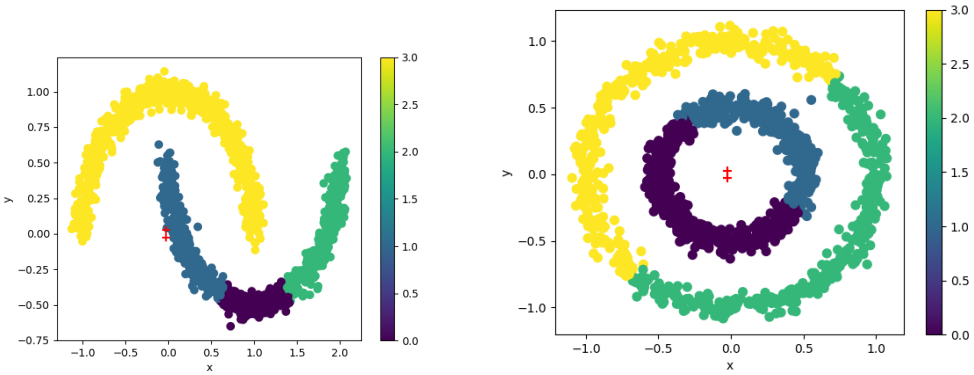
Kmeans

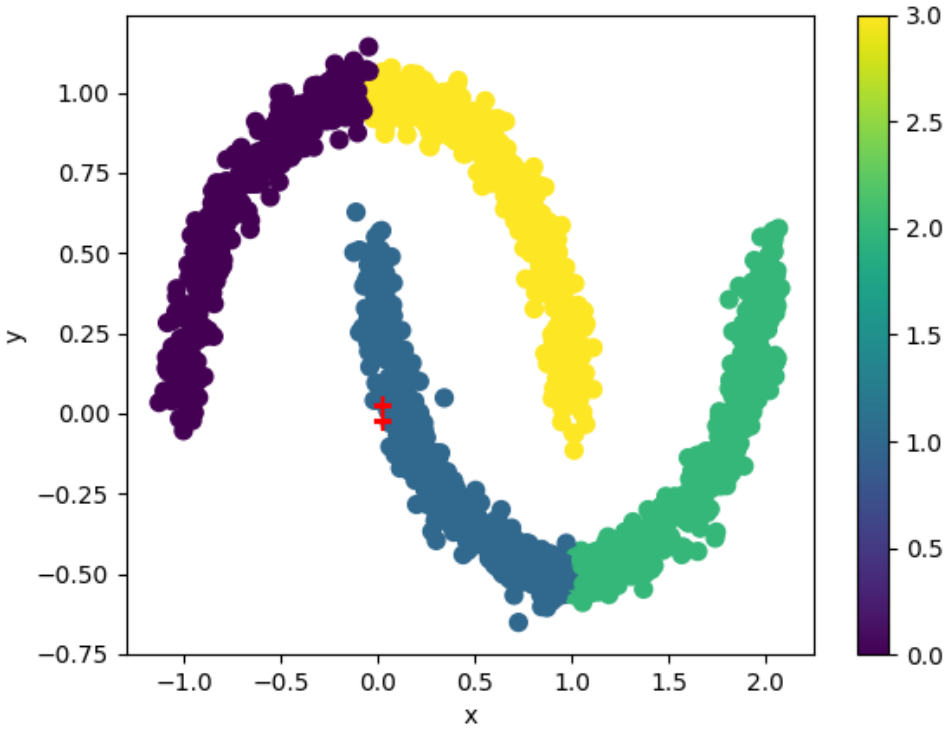


Kernel Kmeans



Spectral Clustering




1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
y
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x
3.0
2.5
2.0
1.5
1.0
0.5
0.0