Homework# 6 Clustering

0756079 陳冠聞

1. Clustering Procedure for k-means/kernel k-means for k = 2, 3, 4

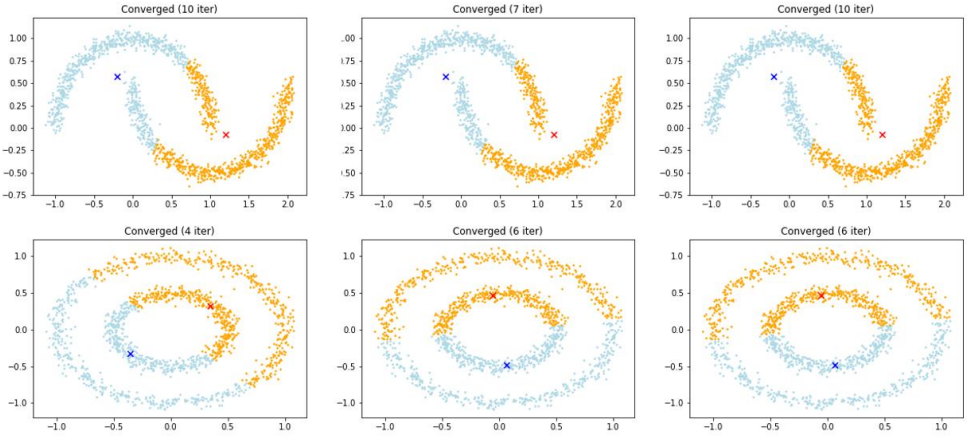
Please see ***kmeans.mp4*** & ***kernel-kmeans.mp4*** in detail.

2. Different initialization of k-means/kernel k-means

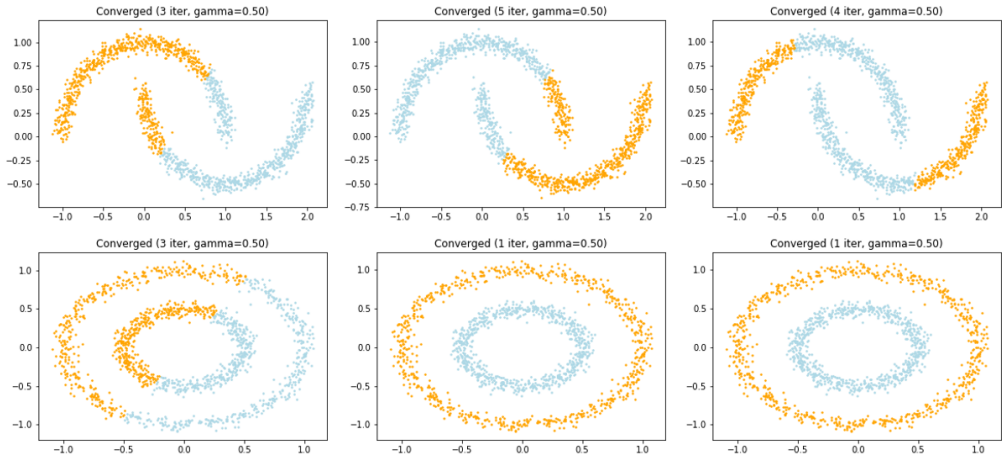
We use three different methods (Random, Distance-To-Origin and Distance-To-Center) for initialization. And we can see that for these two dataset, k-means can not get desired result for any initialization method, and kernel k-means can get desired result of circle dataset if we use Dist2Origin or Dist2Center for initialization.

Following are figures of clustering result for k=2, and from left to right are Random, Distance-To-Origin and Distance-To-Center method respectively.

K-means

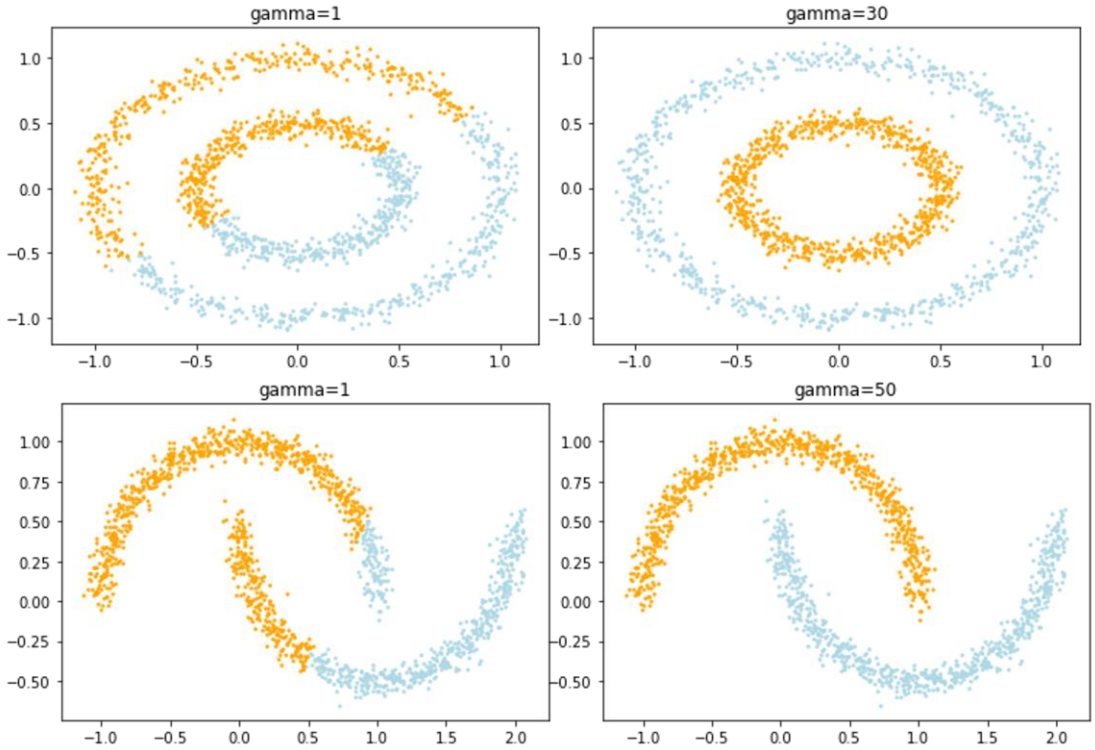


Kernel K-means



3. Spectral clustering

We found that the value of gamma for RBF kernel will effect the result of Spectral clustering. Following are figures of clustering result for Spectral clustering of different gamma.



For spectral clustering, you can see if data points within the same cluster do have the same coordinates in the eigenspace of graph Laplacian, discuss in the report.

Because the geometric meaning of spectral clustering is to use Laplacian Eigenmap to do dimension reduction, and then do k-means clustering. Since k-means will assign the membership of a data point by the closest distance to the cluster, the data points which have the same cluster will have similar coordinate.

Reference

Belkin, M., & Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, *15*(6), 1373-1396.

4. Code : RBF Kernel K-means

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from math import exp
from tqdm import tqdm_notebook as tqdm

def read_data(file_name):
    datas = []
    with open(file_name) as f:
        for line in f:
            data = line.split(',')
            data[0] = float(data[0])
            data[1] = float(data[1])
            datas.append(data)
    return datas
```

Read Data and convert to matrix form

In [2]:
```
data_moon = read_data('moon.txt')
data_circle = read_data('circle.txt')

X_moon = np.array(data_moon)
X_circle = np.array(data_circle)
```

In [3]:
```
class RBF_kmeans_clustering:

    def __init__(self, k=1, gamma=1):
        self.k = k
        self.num_img = 0
        self.gamma = gamma

    def initMember(self, X, method='random'):
        N = X.shape[0]
        X_membership = np.zeros((N), dtype=np.int32)

        if method == 'random':
            for i in range(N):
                X_membership[i] = np.random.randint(self.k)

        elif method == 'Dist2Origin':
            origin = np.zeros(X[0].shape)
            X_to_origin =  [self.distance(x, origin) for x in X]
            X_to_origin_order = np.argsort(X_to_origin)
            num_data_per_cluster = N//self.k
            for cluster in range(self.k):
                for idx in X_to_origin_order[(cluster)*num_data_per_cluster:(cluster+1)*num_data_per_cluster]:
                    X_membership[idx] = cluster

        elif method == 'Dist2Center':
            center = np.mean(X, axis=0)
            X_to_origin =  [self.distance(x, center) for x in X]
            X_to_origin_order = np.argsort(X_to_origin)
            num_data_per_cluster = N//self.k
            for cluster in range(self.k):
                for idx in X_to_origin_order[(cluster)*num_data_per_cluster:(cluster+1)*num_data_per_cluster]:
                    X_membership[idx] = cluster
        return X_membership
```

Initialization Method

```
def compute_RBF_kernel(self, X, gamma=5):
    # k(x1,x2) = exp(-gamma*length(x1-x2)**2)
    RBF_kernel = np.zeros((X.shape[0], X.shape[0]))
    for i in range(X.shape[0]):
        for j in range(X.shape[0]):
            RBF_kernel[i][j] = exp(-gamma* np.dot((X[i]-X[j]), (X[i]-X[j])))
    return rbf_kernel(X, gamma=gamma)
```

Compute RBF kernel

```python
def precompute_third_term(self, X, a, RBF_kernrl):
    # Compute third term
    self.third_term = []
    for k in range(self.k):
        c = np.count_nonzero(a[k])
        third_term = 0
        for p in range(X.shape[0]):
            for q in range(X.shape[0]):
                third_term += a[k][p]*a[k][q]*RBF_kernrl[p][q]
        third_term *= (1/c)**2
        self.third_term.append(third_term)
```

```python
def distance_to_cluster(self, X, membershipMatrix, RBF_kernrl, x_j, cluster_k):
    a = membershipMatrix
    j = x_j
    k = cluster_k
    c = np.count_nonzero(a[k])

    # Compute second term
    second_term = 0
    for n in range(X.shape[0]):
        second_term += a[k][n] * RBF_kernrl[j][n]
    second_term *= 2/c
    third_term = self.third_term[k]
    return RBF_kernrl[j][j] - second_term + third_term
```

Compute Distance of x_j to cluster_k (precompute third term in formula to speed up)

```python
def assign_membership(self, X, membershipMatrix, RBF_kernrl):
    X_membership = np.zeros((X.shape[0]), dtype=np.int32)

    for ix, x in enumerate(X):
        min_dist = self.distance_to_cluster(X, membershipMatrix, RBF_kernrl, ix, 0)
        for cluster in range(self.k):
            cur_dist = self.distance_to_cluster(X, membershipMatrix, RBF_kernrl, ix, cluster)
            if cur_dist < min_dist:
                min_dist = cur_dist
                X_membership[ix] = cluster

    return X_membership
```

Assign membership by min distance to cluster

```python
def fit(self, X, method=None, visulaize=True, max_iter=100):

    # init cluster
    X_membership = self.initMember(X, method)
    old_membership = np.copy(X_membership)

    membershipMatrix = self.getMembershipMatrix(X_membership)

    # Compute kernel
    RBF_kernrl = self.compute_RBF_kernel(X, self.gamma)

    self.draw(X, X_membership, title='Init (%s) k=%d, gamma=%.2f'%(method, self.k, self.gamma))
    for it in range(max_iter):

        # assign new membership based on kernel (skip for compute centroid)
        self.precompute_third_term(X, membershipMatrix, RBF_kernrl)
        X_membership = self.assign_membership(X, membershipMatrix, RBF_kernrl)
        membershipMatrix = self.getMembershipMatrix(X_membership)

        if(visulaize):
            self.draw(X, X_membership, title='Interation %d'%(it+1))

        # Determine convergance
        if np.count_nonzero(old_membership == X_membership) >= X.shape[0]*0.99:
            break
        else:
            old_membership = np.copy(X_membership)
    self.draw(X, X_membership, title='Converged (%d iter, gamma=%.2f)'%(it+1, self.gamma))
    return X_membership
```

Initialize Membership

Compute Kernel

Compute new membership

Until Converge

```python
init_methods = ['random', 'Dist2Origin', 'Dist2Center']
k_range = [2,3,4]
gamma = 0.5
for k in k_range:
    for method in init_methods:
        model = RBF_kmeans_clustering(k=k, gamma=gamma)
        membership = model.fit(X_moon[:], method=method, visulaize=False)
```

Do kernel k-means clustering on different k and initialization method

5. Code : K-means

Only show codes which are different from kernel k-means

```python
def fit(self, X, method=None, visulaize=True, max_iter=100):
    # random init member
    X_membership = self.initMember(X, method)
    old_membership = np.copy(X_membership)
    self.draw(X, X_membership, title='Init (%s) k=%d'%(method, self.k))
    for it in range(max_iter):
        # calculate centroids based on membership
        self.centroids = self.calculate_centroids(X, X_membership)

        # assign membership based on centroids
        X_membership = self.assign_membership(X, self.centroids)

        if(visulaize):
            self.draw(X, X_membership, title='Interation %d'%(it+1))
        if np.count_nonzero(old_membership == X_membership) >= X.shape[0]*0.9999:
            break
        else:
            old_membership = np.copy(X_membership)

    self.draw(X, X_membership, title='Converged (%d iter)'%(it+1))
    return X_membership

def calculate_centroids(self, X, X_membership):
    centroids = np.zeros( (self.k, *(X.shape[1:])) )
    num_data = [0 for x in range(self.k)]

    for x, membership in zip(X, X_membership):
        # updata by weighted sum
        centroids[membership] = (num_data[membership]*centroids[membership] + x) / (num_data[membership]+1)
        num_data[membership] += 1
    return centroids
```

Update centroids by fixing memberships

Update memberships by fixing centroids

Calculate centroids of a cluster

6. Code : Spectral clustering

Only show codes which are different from kernel k-means

```python
def fit(self, X, gamma=1):
    # 0. Define similarity matrix W and D
    W = compute_RBF_kernel(X, gamma=self.gamma)
    D = np.zeros((X.shape[0], X.shape[0]))
    for d in range(X.shape[0]):
        D[d][d] = W[d].sum()

    # 1. Graph Laplacian L = D - W
    L = D - W

    # Get first k eigenvector
    eigenValues, eigenVectors = LA.eig(L)
    idx = eigenValues.argsort()
    eigenValues = eigenValues[idx]
    eigenVectors = eigenVectors[:,idx]
    U = eigenVectors[:, 1:self.n_cluster+1]

    # Do k-means
    kmeans = k_means_clustering(k=2)
    membership = kmeans.fit(U)
    print(membership)
    self.draw(X, membership)
```

Calculate Graph Laplacian

Calculate First K eigenvector of Graph Laplacian

Do k-means

For more detail, please see ***k-means_clustering.py***, ***RBF_k-means_clustering.py*** and ***spectral_clustering.py***