0410024 余東儒

1. Kmeans

(1)開檔 將 data point 分別存入 x 和 y,size 存#data points

```
fp=open("test1_data.txt","r")
str=fp.read()
x=[]
y=[]
i=0
size=0;
while str.find(' ',i)!=-1:
    x_end=str.find(' ',i)
    x.append(float(str[i:x_end-1]))
    y_end=str.find('\n',x_end)
    y.append(float(str[x_end+1:y_end-1]))
    i=y_end
    size=size+1
```

(2)ux,uy 為 u 的初始值，這邊的初始方法為隨機挑選 k 個資料點

```
ux=[]
uy=[]
i=0
k=int(input("k clustering:"))
for i in range(k):
    rd=rand.randint(0,size-1)
    ux.append(x[rd])
    uy.append(y[rd])
```

(3)開始 kmeans clustering

r 為每個資料點的判斷分類

ax,ay 為幫助最後計算 u 的暫存,分別($\Sigma r_{nk}x_n$, Σr_{nk})

xx,yy 存分類後的資料點,xx[k],yy[k]存分類 k 的資料點

checkux,checkuy 為計算收斂,當 checku 跟 u 一樣時代表收斂,結束 clustering

```
"""start clustering"""
while 1:
    checkux=copy.copy(ux)
    checkuy=copy.copy(uy)
    r=[]
    ax=[]
    ay=[]
    xx=[]
    yy=[]
    for i in range(k):
        ax.append((0,0))
        ay.append((0,0))
        xx.append([])
        yy.append([])
```

(4)開始計算每個 data point 到 u 的 distance

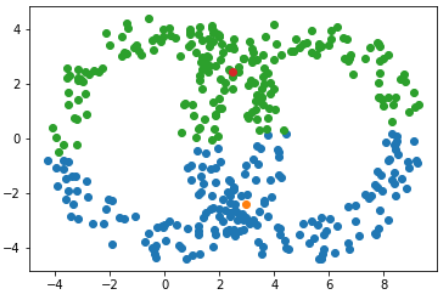
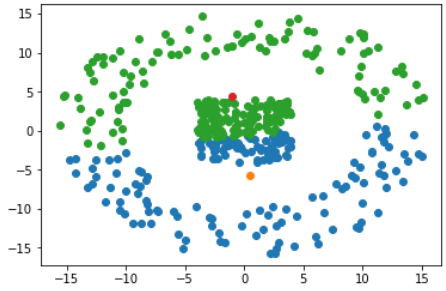
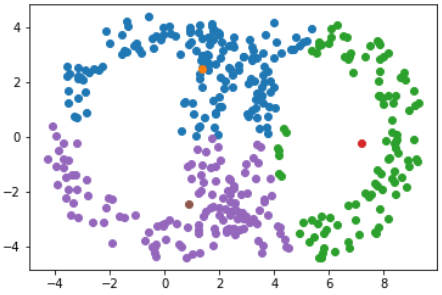
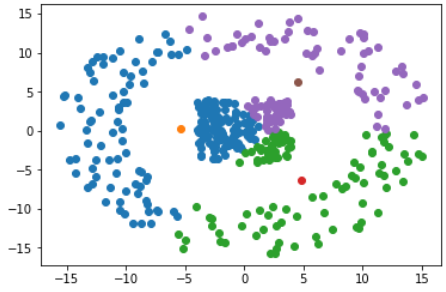
tmp 紀錄每個 data point 離最近 u 的分類
d 為 distance 初始值設成非常大
算完之後,r 存最後的 tmp,
xx.yy 將資料點存進第 tmp 個 list,
ax.ay 更新

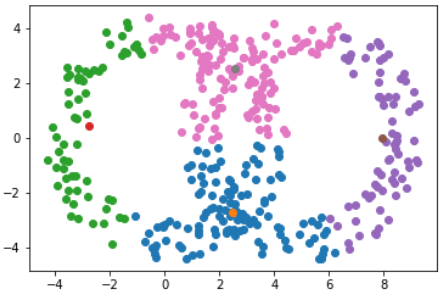
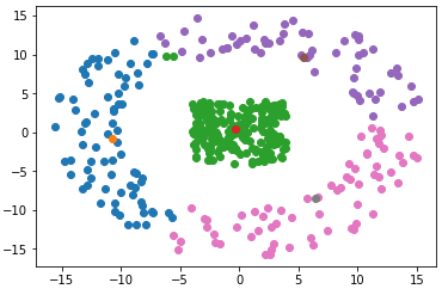
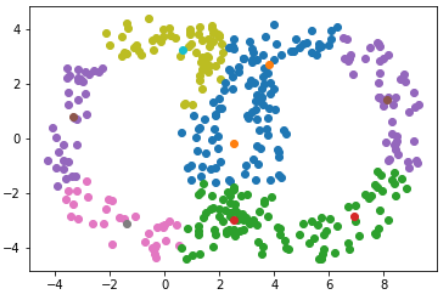
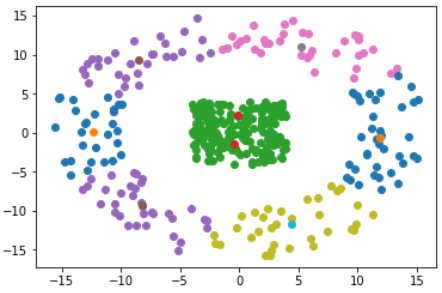
```
for i in range(size):
    tmp=0
    d=100000
    for j in range(k):
        if (x[i]-ux[j])*(x[i]-ux[j])+(y[i]-uy[j])*(y[i]-uy[j])<d:
            d=(x[i]-ux[j])*(x[i]-ux[j])+(y[i]-uy[j])*(y[i]-uy[j])
            tmp=j
    r.insert(i,tmp)
    xx[tmp].append(x[i])
    yy[tmp].append(y[i])
    ax[tmp]=(ax[tmp][0]+x[i],ax[tmp][1]+1)
    ay[tmp]=(ay[tmp][0]+y[i],ay[tmp][1]+1)
```

(5)做完 step(4)之後,用 ax,ay 算最新的 u
再來判斷是否跟 check 一樣,如果一樣就結束,不一樣就繼續執行程式

```
for i in range(k):
    ux[i]=ax[i][0]/ax[i][1]
    uy[i]=ay[i][0]/ay[i][1]
if ux==checkux and uy==checkuy:
    break
```

Result:

	Test1	Test2
K=2		
K=3		

K=4		
K=8		

2. kernel kmeans

(1) 跟 kmeans 一樣先讀檔,存進 data list

```
fp=open("test1_data.txt","r")
str=fp.read()
x=[]
y=[]
data=[]
i=0
size=0;
while str.find(' ',i)!=-1:
    x_end=str.find(' ',i)
    x.append(float(str[i:x_end-1]))
    y_end=str.find('\n',x_end)
    y.append(float(str[x_end+1:y_end-1]))
    data.append((float(str[i:x_end-1]),float(str[x_end+1:y_end-1])))
    i=y_end
    size=size+1
```

(2) Initial

這裡用的初始方法為隨機把所有 data points 指定成 k 類

xx 和 yy 一樣是存分類後的 data point

r 則是存分類的類別

count 紀錄每個類別的數量

$$\mu_k^{\phi} = \frac{1}{|C_k|}\sum_{n=1}^{N}\alpha_{kn}\phi(x_n)$$

a 則是計算 u 時用的參數,在這裡直接將 a_{kn} 跟 c_k 結合

c 則是 check 的功用,當 a 和 c 一樣時則代表收斂

```
xx=[]
yy=[]
for i in range(k):
    xx.append([])
    yy.append([])
for i in range(size):
    rd=rand.randint(0,k-1)
    r[i]=rd
    count[rd]=count[rd]+1
for i in range(size):
    tmp=[0]*k
    tmp[r[i]]=1/count[rd]
    a[i]=tmp
    xx[r[i]].append(data[i][0])
    yy[r[i]].append(data[i][1])
for i in range(k):
        plt.plot(xx[i],yy[i],'o')
plt.show()

c=copy.deepcopy(a)
```

(3) RBF kernel function:d1.d2 為兩資料點,

```
def kernel(d1,d2):
    return np.exp((pow(d1[0]-d2[0],2)+pow(d1[1]-d2[1],2))*0.05*(-0.5))
```

(4)開始進入 loop

r,xx,yy 每經過一次 iteration 都要歸零,只有 a 要留著上次 iteraion 的結果

third 為計算以下式子

$$\frac{1}{|C_k|^2}\sum_p\sum_q \alpha_{kp}\alpha_{kq}\mathbf{k}(x_p,x_q)$$

```
while 1:
    r=[0]*size
    xx=[]
    yy=[]
    for i in range(k):
        xx.append([])
        yy.append([])
    count=[0]*k
    third=[0]*k
    for j in range(k):
        for p in range(size):
                    for q in range(size):
                        third[j]=third[j]+a[p][j]*a[q][j]*kernel(data[p],data[q])
```

(5)

tmp 和 d 的作用一樣是存分類跟紀錄 distance:

second 則是

$$\frac{2}{|C_k|}\sum_n \alpha_{kn}\mathbf{k}(x_j,x_n)$$

隨後將 data point 紀錄進 xx 和 yy 裡,更新 r 和 count

```
for i in range(size):
        tmp=0
        d=100000
        for j in range(k):
            second=0
            for n in range(size):
                second=second+a[n][j]*kernel(data[i],data[n])
            if kernel(data[i],data[i])-2*second+third[j] < d:
                d=kernel(data[i],data[i])-2*second+third[j]
                tmp=j
        xx[tmp].append(data[i][0])
        yy[tmp].append(data[i][1])
        r[i]=tmp
        count[tmp]=count[tmp]+1
```

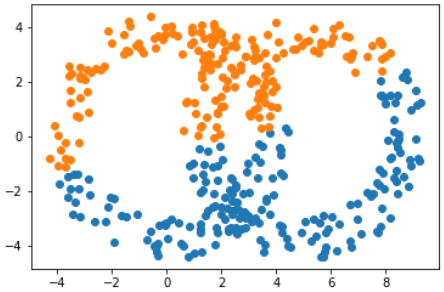
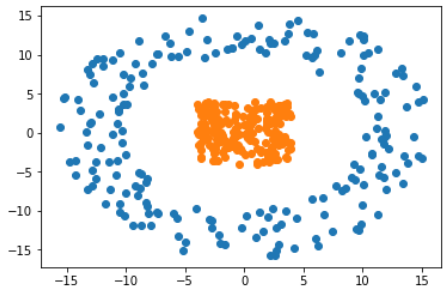
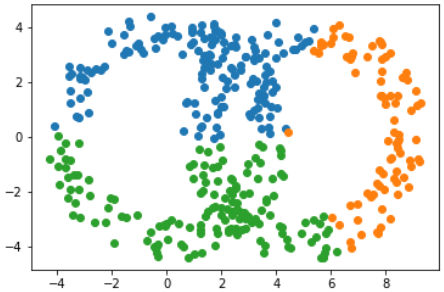
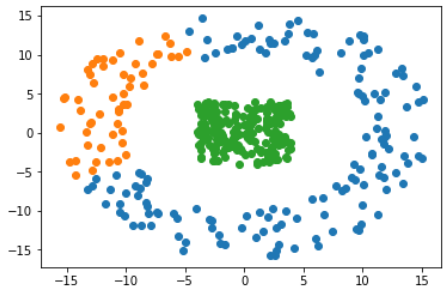
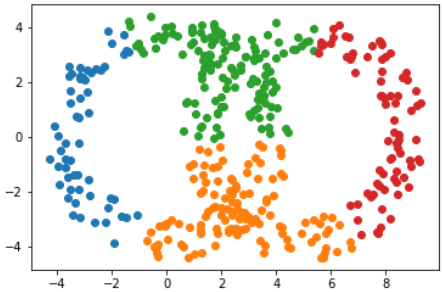
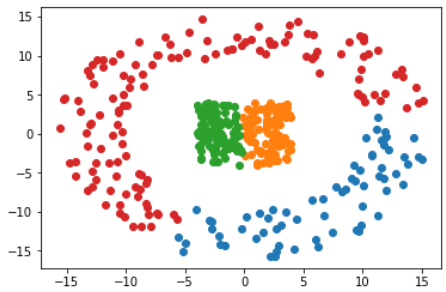
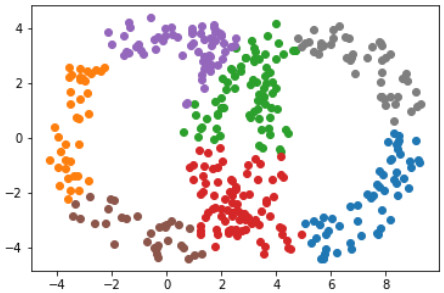
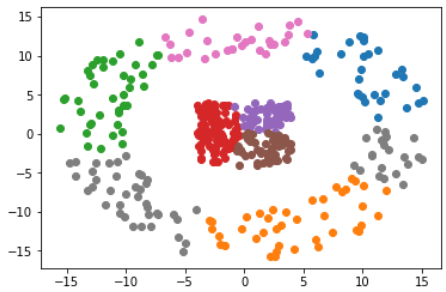
(6)

用 count 的結果更新 a

判斷 c 和 a 是否一樣，如果一樣則代表收斂，不一樣則繼續執行下個 iteration

```
for i in range(size):
    for j in range(k):
        if r[i]==j:
            a[i][j]=(1/count[r[i]])
        else:
            a[i][j]=0
if c==a:
    break
c=copy.deepcopy(a)
```

Result:

	Test1	Test2
K=2		
K=3		
K=4		
K=8		

3. spectral clustering

(1)開檔

```
fp=open("test2_data.txt","r")
str=fp.read()

x=[]
y=[]
data=[]
i=0
k=2
size=0;
while str.find(' ',i)!=-1:
    x_end=str.find(' ',i)
    y_end=str.find('\n',x_end)
    data.append((float(str[i:x_end-1]),float(str[x_end+1:y_end-1])))
    i=y_end
    size=size+1
```

(2)計算 L,D,W 矩陣(皆為(size*size)

```
W=np.zeros((size,size), dtype=np.float)
D=np.zeros((size,size), dtype=np.float)
for i in range(size):
    tmp=0
    for j in range(size):
        W[i][j]=weight(data[i],data[j])
        tmp=tmp+weight(data[i],data[j])
    D[i][i]=tmp
L=D-W
```

Weight 函式一樣是用 RBF kernel

```
def weight(d1,d2):
    if d1==d2:
        return 0
    else:
        return np.exp((pow(d1[0]-d2[0],2)+pow(d1[1]-d2[1],2))*0.1
                      *(-0.5))
```

(3) solve eigen problem

用 eig function 找 eigenvalue 和 eigenvector

用 argsort funtion 排序 eigen value 大小並且其相對應的 eigen vector

```
eigvalue,eigvector=eig(L)
newdata=list()
idx = eigvalue.argsort()
eigvec=list()
eigvalue = eigvalue[idx]
eigvector=eigvector[idx]
```

(4) U 矩陣(size*k)

將前 k 個 non-null eigenvalue 對應的 eigen vector 存入 U

```
U=np.zeros((size,k), dtype=np.float)
for i in range(size):
    for j in range(k+1):
        if j!=0:
            U[i][j-1]=eigvec[j][i]
```

(5) 做 kmeans clustering

u 為 k 個分類的初始中心點 u:k*k

以隨機的方式初始

```
u=[]
for i in range(k):
    u.append(())
for i in range(k):
        rd=rand.randint(0,size-1)
        uu=[]
        for j in range(k):
            uu.append(U[rd][j])
        u[i]=uu

kmeans(U,u,data,k)
```

Kmeans function

基本上做法跟第一題一模一樣

只是 a 的維度變成 k*k*2

a[分類][axis][計算 u 的分子和分母]

```
def kmeans(U,u,data,k):
        checku=copy.deepcopy(u)
        #z=[0,0]
        a=list()
        for i in range(k):
            a.append([])
            for j in range(k):
                a[i].append((0,0))
        #print(a[2][2])
        xx=[]
        yy=[]
#         ss=[]
#         dd=[]
        r=[0]*size
        for i in range(k):
            xx.append([])
            yy.append([])
#             ss.append([])
#             dd.append([])
        for i in range(size):
            tmp=0
            d=100000
            for j in range(k):
                dd=0
                for l in range(k):
                    dd=dd+(U[i][l]-u[j][l])*(U[i][l]-u[j][l])
                if dd<d:
                    d=dd
                    tmp=j
            r[i]=tmp
#             z[tmp]=z[tmp]+1
            #print(d)
            xx[tmp].append(data[i][0])
            yy[tmp].append(data[i][1])
#             ss[tmp].append(U[i][0])
#             dd[tmp].append(U[i][1])
            for j in range(k):
                a[tmp][j]=(a[tmp][j][0]+U[i][0],a[tmp][j][1]+1)
```

u 一樣是 k 分類的中心點

flag 判斷是否收斂，若收斂則結束，否則繼續 kmeans

```
for i in range(k):
    for j in range(k):
        u[i][j]=a[i][j][0]/a[i][j][1]
for i in range(k):
    plt.plot(xx[i],yy[i],'o')
plt.show()
flag=1
for i in range(k):
    for j in range(k):
        if checku[i][j]!=u[i][j]:
            flag=0
            break
    if flag==0:
        break
#print(u)


 ss=input("Enter:")
 if ss=='e':
     return
if flag==1:
    return
else:
    return kmeans(U,u,data,k)
```