KÜMELEME ALGORİTMALARI

Ömer Yılmaz

İÇİNDEKİLER

Özet..... I

BÖLÜM 1

1.1 Kümeleme Nedir? 1
1.2 Kümeleme Algoritmaları Hangi Alanlarda Kullanılır? 2
1.3 Kümeleme Algoritmaları Nelerdir? 3
1.3.1 K-Means 4
1.3.2 Affinity Propagation 5
1.3.3 Mean-shift 6
1.3.4 Spectral Clustering 7
1.3.5 Hierarchical Clustering 9
1.3.6 DBSCAN 12

BÖLÜM 2

2.1 K-Means Uygulama 14
2.2 Affinity Propagation Uygulama 18
2.3 Mean-Shift Uygulama 24
2.4 Spectral Clustering Uygulama 28
2.5 Fuzzy C Means Uygulama 34
2.6 DBSCAN Uygulama 38

KAYNAKÇA

KÜMELEME ALGORİTMALARI

Ömer YILMAZ

Kümeleme algoritmaları, veri madenciliği ve makine öğrenmesi gibi alanlarda kullanılan ve benzer özelliklere sahip verileri gruplamak için kullanılan bir dizi tekniklerdir. Bu algoritmalar, verileri kümeler halinde gruplandırmak için kullanılır ve veriler arasındaki farklılıkları ve benzerlikleri analiz ederler.

1.1 Kümeleme Nedir?

Kümeleme algoritmaları, veri madenciliği ve makine öğrenmesi gibi alanlarda kullanılan ve benzer özelliklere sahip verileri gruplamak için kullanılan bir dizi tekniklerdir. Bu algoritmalar, verileri kümeler halinde gruplandırmak için kullanılır ve veriler arasındaki farklılıkları ve benzerlikleri analiz ederler.

Kümeleme algoritmaları, verileri benzer özelliklere sahip kümeler halinde gruplandırarak daha fazla anlam çıkarmamızı sağlar. Buna göre, bu algoritmalar:

1-Verileri belirli özelliklerini bir araya toplar.
2-Benzer verileri belirli bir mesafe ölçeği bazında gruplandırır.
3-Gruplar arasındaki benzerlikleri belirler.

Popüler kümeleme algoritmaları arasında k-Means, Hiyerarşik Kümeleme, DBSCAN, Mean-Shift ve Spektral Kümeleme bulunur.

k-Means, verileri belirli sayıda kümeye bölerek her kümenin merkezini bulmayı amaçlar. Hiyerarşik Kümeleme, küme sayısını değil de veriler arasındaki benzerlikleri baz alarak kümeler oluşturur. DBSCAN, gürültü içeren verileri tamamen silerek kümeleme yapar. Mean-Shift, veriler arasındaki yoğunluk farklılıklarını temel alarak kümeler oluşturur. Spektral Kümeleme, doğrudan veriler arasındaki benzerliği değerlendirir.

Kümeleme algoritmaları, pazarlama, sosyal medya, biyoistatistik, finans ve diğer pek çok alanda yaygın olarak kullanılmaktadır. Bu algoritmalar, büyük veri kümeleri içindeki desenleri ve ilişkileri tespit edebilme yeteneğinde oldukları için işletmeler için büyük bir fırsat sunarlar.

1.2 Kümeleme Algoritmaları Hangi Alanlarda Kullanılır?

Kümeleme algoritmaları, birbiriyle benzer özellikleri olan verileri gruplamak için kullanılır. Bu nedenle, birçok farklı uygulama alanında kullanılırlar:

Pazarlama: Müşterileri belirli özelliklerine göre gruplamak ve alışveriş alışkanlıklarına göre pazarlama stratejileri geliştirmek için kullanılır.

Tıp: Hastalıkları benzer özelliklere göre sınıflandırmak ve ilaç araştırmalarında kullanılır.

Sosyal medya: İçeriklerin benzer özelliklerine göre gruplandırılması, tüketici davranışları hakkında bilgi toplamak ve kişiselleştirilmiş öneriler sağlamak için kullanılır.

Finans: Benzer özelliklere sahip müşterileri gruplandırıp, portföy yönetimi ve risk yönetimi için kullanılır.

Coğrafi Bilgi Sistemleri: Coğrafik özelliklere göre verilerin gruplandırılması, şehir planlaması ve afet yönetimi için kullanılır.

Biyoinformatik: Gendenetik verileri benzer özelliklere göre sınıflandırmak ve tıbbi teşhislerde kullanmak için kullanılır.

Tarım: Bitki özelliklerine göre gruplandırılır ve tarım alanlarının verimlilik analizi yapılır.

Veri Madenciliği: Verilerin benzer özelliklerine göre sınıflandırılıp, keşfedilmesi ve analizi için kullanılır.

1.3 Kümeleme Algoritmaları Nelerdir?

1. K-Means
2. Affinity Propagation
3. Mean-shift
4. Spectral Clustering
5. Hierarchical Clustering
6. DBSCAN

1.3.1 K-Means

K-Means
Temelinde, verileri benzerliklerine göre kümelemeyi amaçlar. Benzerlik, veriler arasındaki uzaklığa göre belirlenmektedir. Uzaklığın az olması benzerliği arttırır.

Giriş parametresi olarak ‘k’ sayısı verilmelidir. Bu sayı örneklemin kaç adet kümeye ayrılacağını belirtir.

En sık kullanılan kümeleme algoritmalarındandır. Uygulanması oldukça kolaydır. Büyük ölçekli verileri hızlı ve etkin bir şekilde kümeleyebilir.

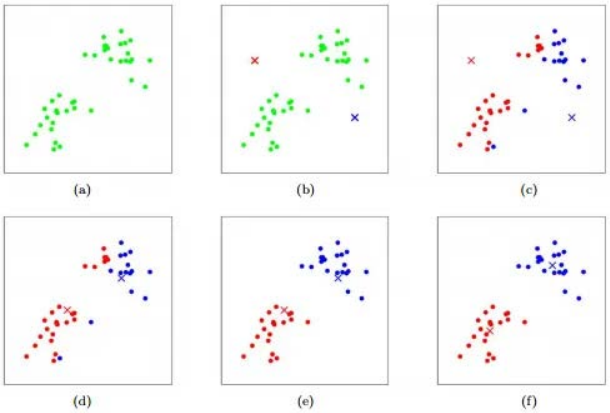
Çalışma mantığı şu şekildedir:

‘k’ adet küme merkezi rastgele seçilir.
2. Merkez dışındaki veriler mesafelerine göre kümelendirilir.

3. Yapılan kümelendirmeye göre yeni küme merkezleri belirlenir.

4. Kararlı hale gelene kadar 2. ve 3. adım tekrarlanır.

‘k’ sayısının işlem başlamadan belirlenmesi duruma göre avantaj ya da dezavantaj oluşturabilir.



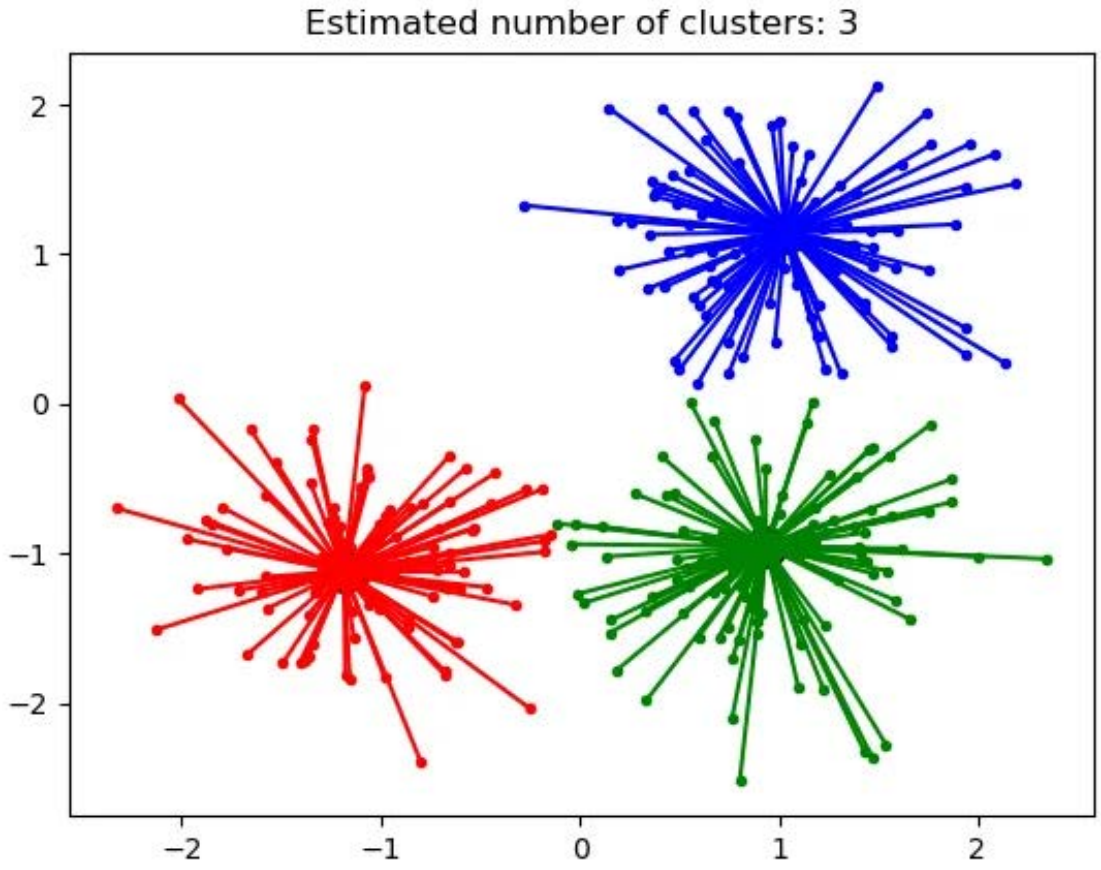
Şekil 1.1 K-Means uygulama aşamaları

1.3.2 Affinity Propagation

Veri noktaları arasında “mesaj göndererek” yakınsamaya dayalı kümeler oluşturur. Küme sayısının belirtilmesi gerekmez.

Veri çiftleri arasında mesajlar gönderilerek kümeler oluşturulur. Ardından, verileri en iyi temsil edebilecek az sayıda veri kümesi tanımlanır. Veri çiftleri arasında gönderilen mesajlar, bir verinin diğerinin kümesine uygun olup olmadığı yanıtlarıyla güncellenir. Bu aşamalar yakınsamaya kadar yinelemeli olarak gerçekleşir.

En büyük dezavantajı karmaşıklığıdır. Küçük ve orta ölçekli veri kümeleri için uygundur.



Şekil 1.2 Affinity Propagation uygulanan bir veri seti

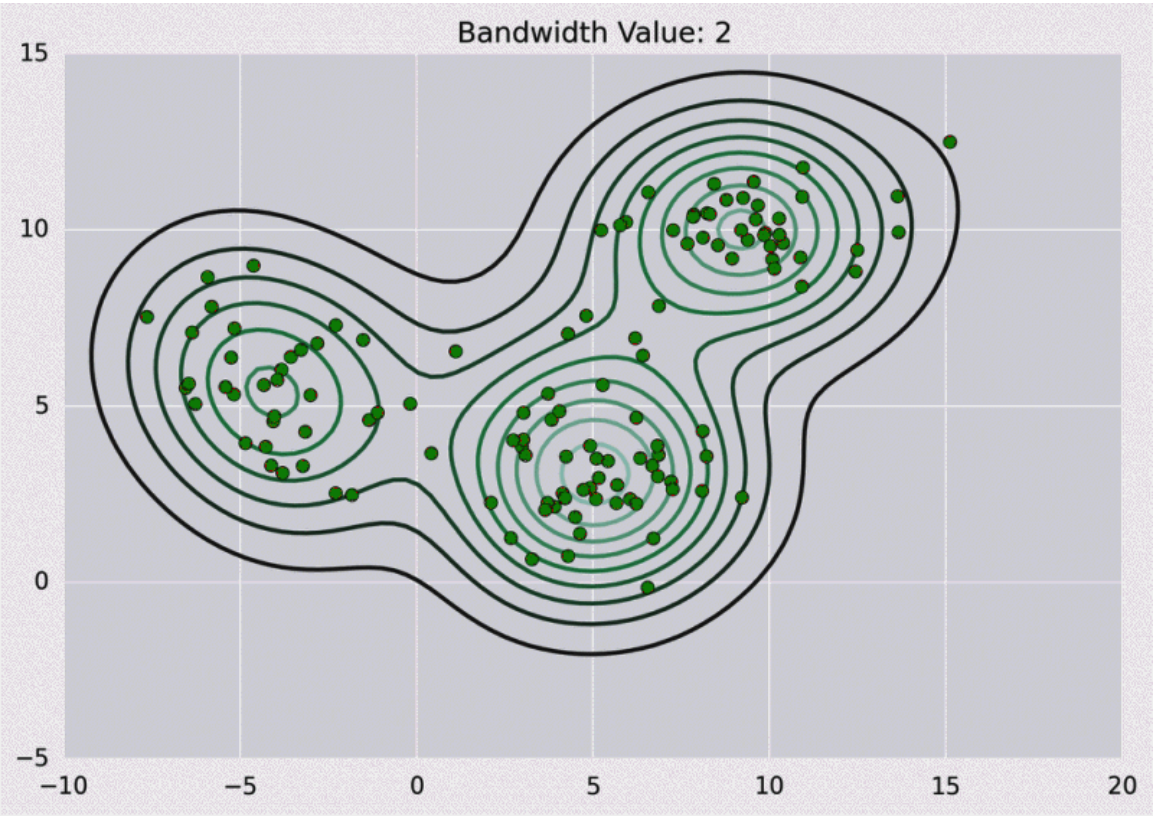
1.3.3 Mean-shift

Veri noktalarını moda doğru kaydırarak yinelemeli olarak kümelere veri noktalarını atamayı sağlar. Algoritmada mod diye tabir edilen, bölgedeki en yüksek veri noktaları yoğunluğudur.

Küme sayısının önceden belirtilmesi gerekmez.

Bir dizi veri noktası verildiğinde, algoritma her bir veri noktasını yinelemeli olarak en yakın küme merkezine doğru atar. En yakın küme merkezi, yakındaki noktaların çoğunun nerede olduğu ile belirlenir. Böylece her bir yinelemede, veri noktaları en çok noktanın bulunduğu yere yaklaşacak, bu da küme merkezini ortaya çıkaracaktır. Algoritma durduğunda, her nokta bir kümeye atanır.

En büyük dezavantajı, hesaplama açısından maliyetli olmasıdır. ($O(n^2)$)



Şekil 1.3 Mean-shift algoritmasının uygulanışı

1.3.4 Spectral Clustering

Spectral kümelemede, veri noktaları bir grafiğin düğümleri olarak ele alınır. Bu nedenle, kümeleme bir grafik bölümleme problemi olarak ele alınır. Düğümler daha sonra kümeler oluşturmak için kolayca ayrılabilen düşük boyutlu bir alana eşlenir. En önemli ayrıntı da, kümelerin şekli/formu hakkında herhangi bir varsayımda bulunulmamasıdır.

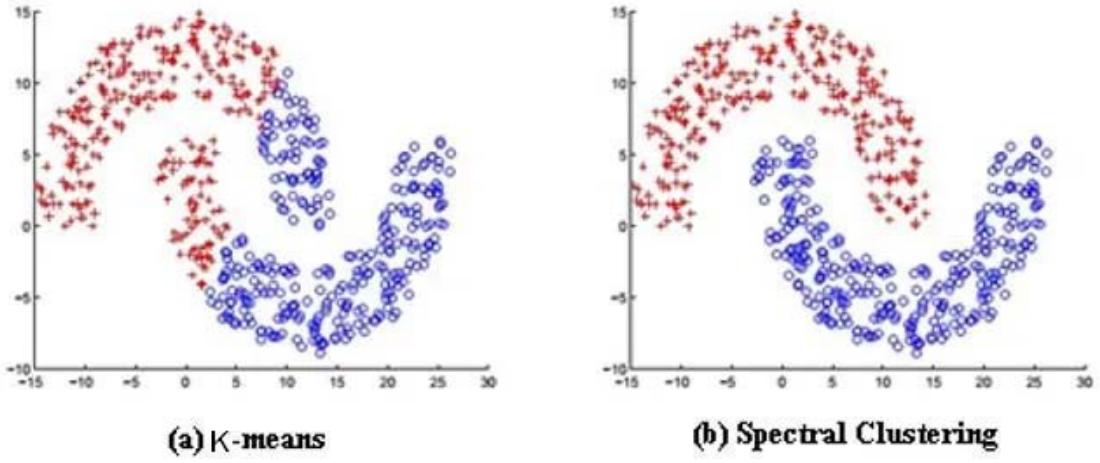
Çalışma mantığı şu şekildedir:

1- Benzerlik grafiği hesaplanır. Bu aşamada ε-komşuluk grafiği, K-NN grafiği ya da tam bağlantılı grafik kullanılabilir.

2- Veriler düşük boyutlu bir alana yansıtılır. Bu aşamada bir grafiğin ilginç özelliklerini bulmakta faydalı olabilecek Graph Laplacian hesaplanır. Bu şekilde veri noktaları düşük boyutlu bir uzaya gömülür ve özdeğer-özvektörler bulunur.

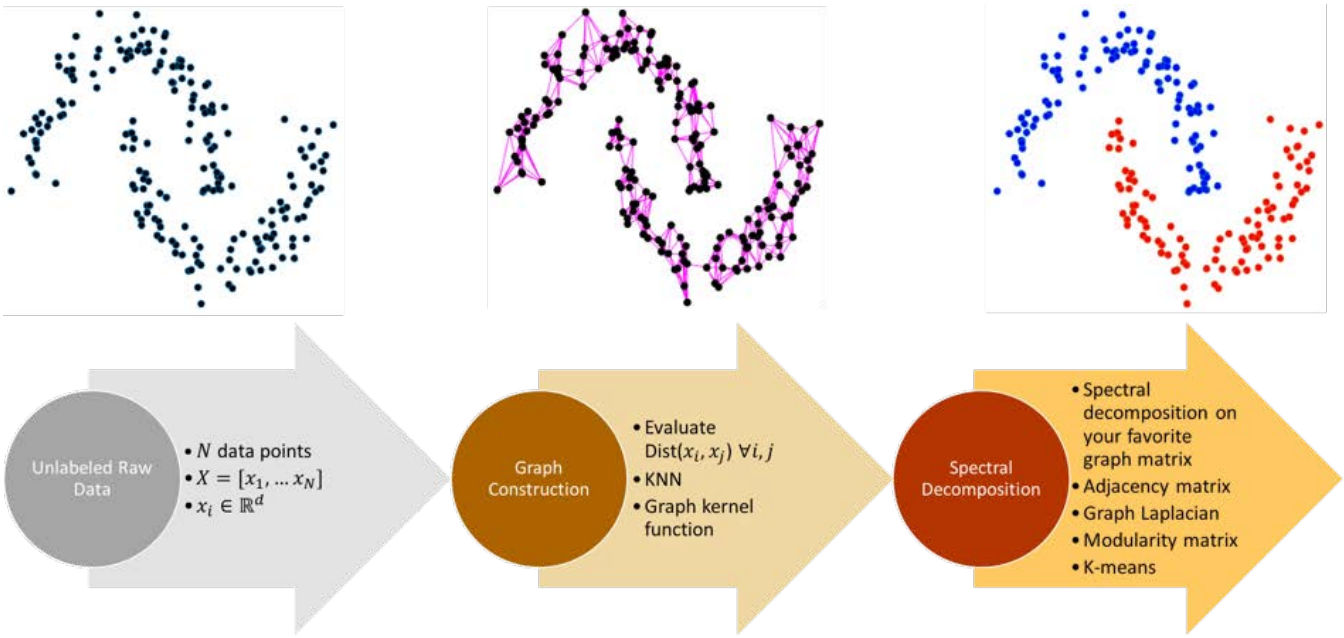
3- Özdeğerler ve özvektörler kullanılarak kümeleme yapılır. Bu aşamada K-Means kümelemesi kullanılır.

Uygulanması kolaydır ve iyi kümeleme sonuçları verir. Seyrek veri setleri için oldukça hızlıdır. Büyük veri kümeleri için ise hesaplama açısından oldukça maliyetlidir.



Şekil 1.4 K-Means ve Spectral Clustering karşılaştırması

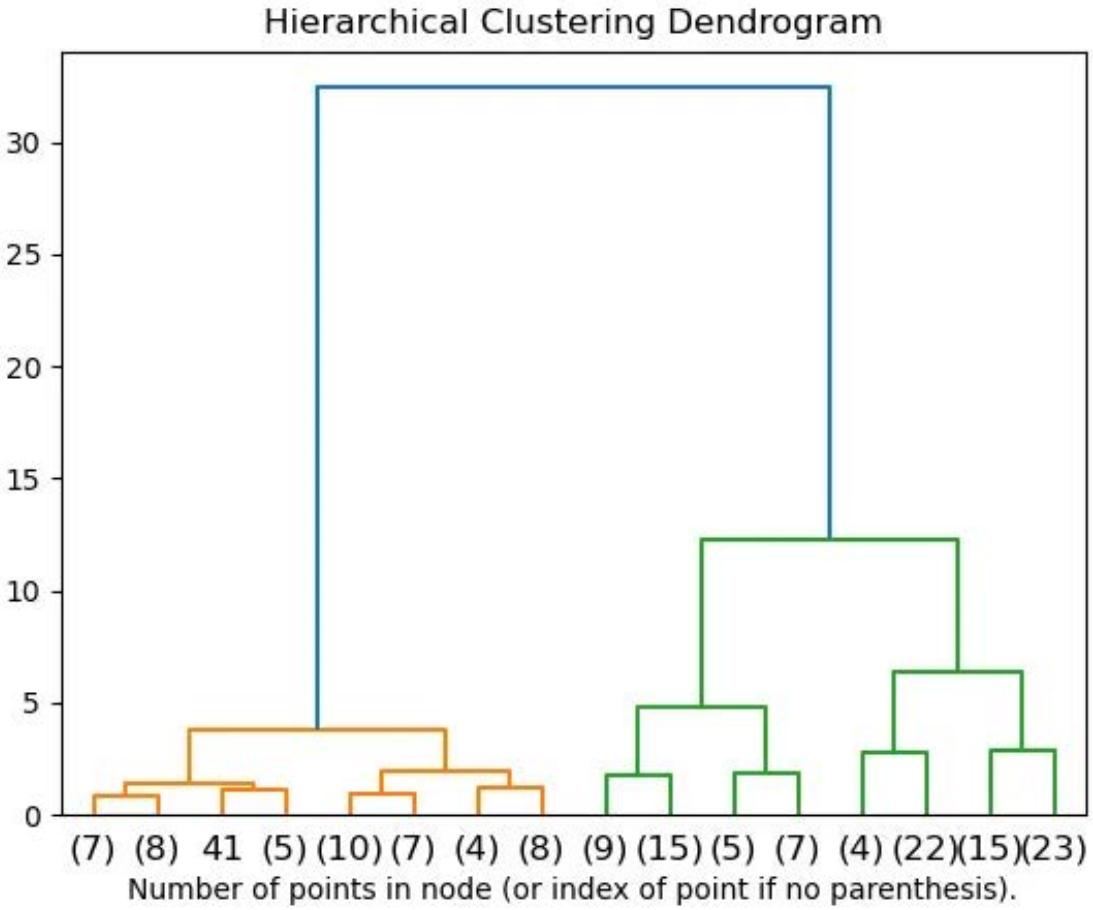
1.3.4 Spectral Clustering



Şekil 1.5 Spectral Clustering uygulama aşamaları

1.3.5 Hierarchical Clustering

Hiyerarşik kümeleme, iç içe kümeleri art arda birleştirerek veya bölerek oluşturan genel bir kümeleme algoritmaları ailesidir. Bu küme hiyerarşisi bir ağaç (veya dendrogram) olarak temsil edilir. Ağacın kökü, tüm örnekleri toplayan benzersiz kümedir, yapraklar yalnızca bir örnek içeren kümelerdir.



Şekil 1.6 Hiyerarşik Kümeleme dendogramı.

Hierarchical Clustering

Hiyerarşik kümeleme stratejileri genellikle iki türe ayrılır.

Aglomeratif : Bu bir “ aşağıdan yukarıya “ yaklaşımdır. Her gözlem kendi kümesinde başlar ve hiyerarşide yukarı doğru çıkıldıkça küme çiftleri birleştirilir.

Bölücü : Bu bir “ yukarıdan aşağıya “ yaklaşımdır. Tüm gözlemler tek bir kümede başlar ve hiyerarşide aşağı doğru hareket ettikçe bölmeler yinelemeli olarak gerçekleştirilir.

Bu yazıda hiyerarşik kümeleme stratejilerinden aglomeratif kümelemeden bahsedeceğiz.

Aglomeratif kümeleme, aşağıdan yukarıya bir yaklaşıma sahiptir. Küme bağlantılarını oluşturmak için aşağıdaki metriklerden biri kullanılır:

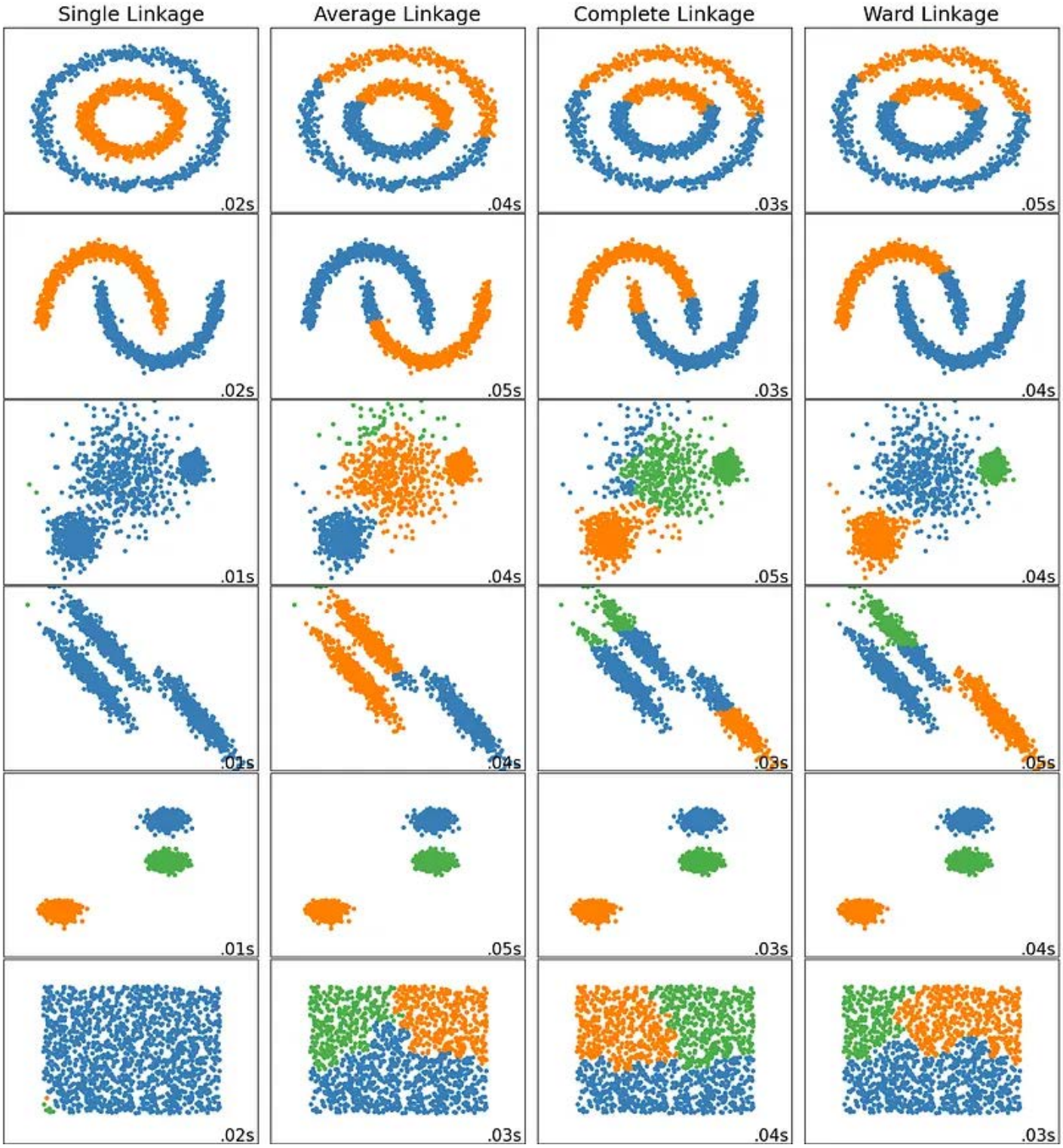
Ward , tüm kümelerdeki kare farklarının toplamını en aza indirir. Bu, varyansı en aza indiren bir yaklaşımdır ve bu anlamda K-Means fonksiyonuna benzer, ancak kümeleyici hiyerarşik bir yaklaşımla ele alınır.

Maksimum veya tam bağlantı (Complete Linkage) , küme çiftlerinin gözlemleri arasındaki maksimum mesafeyi en aza indirir.

Ortalama Bağlantı (Avarage Linkage), küme çiftlerinin tüm gözlemleri arasındaki mesafelerin ortalamasını en aza indirir.

Tek bağlantı (Single Linkage), küme çiftlerinin en yakın gözlemleri arasındaki mesafeyi en aza indirir.

Hierarchical Clustering



Şekil 1.7 Aglomeratif kümelemede farklı metriklerin karşılaştırılması

1.3.6 DBSCAN

Yoğunluğa dayalı bir algoritmadır. Birbirine çok yakın olan noktaları (birçok yakın komşuya sahip noktalar) birlikte gruplandırır. Düşük yoğunluklu (en yakın komşuları çok uzakta olan) bölgelerde bulunan noktaları ise tek başına bulunan aykırı noktalar olarak işaretler. DBSCAN, bilimsel literatürde en çok alıntı yapılan algoritmadır.

Algoritma, kümeleri belirlerken ε (eps) ve minimum points (minPts) parametrelerini kullanır. ε (eps), noktanın komşularını arayacağı uzaklığı belirtir. Minimum points (minPts) ise bir kümenin oluşabilmesi için, noktalar arasında kaç tane komşuluk olması gerektiğini belirtir.

Algoritmanın çalışma mantığı şu şekilde özetlenebilir:

Her noktanın ε (eps) komşuluğundaki noktaları bulun ve minPts sayısından daha fazla komşu bulunursa, o noktayı çekirdek nokta olarak belirleyin.

Bağlı bileşenler arasında çekirdeği olmayan tüm noktaları göz ardı ederek, grafik üzerindeki noktalar.

Çekirdeği olmayan noktalar, çekirdeği olan kümelerdeki noktalardan herhangi birine ε (eps) uzaklıktaysa, o kümeye atanır.

Küme sayısının önceden belirlenmesinin gerekmemesi, aykırı değerlere karşı dayanıklı olması avantajlarıdır.

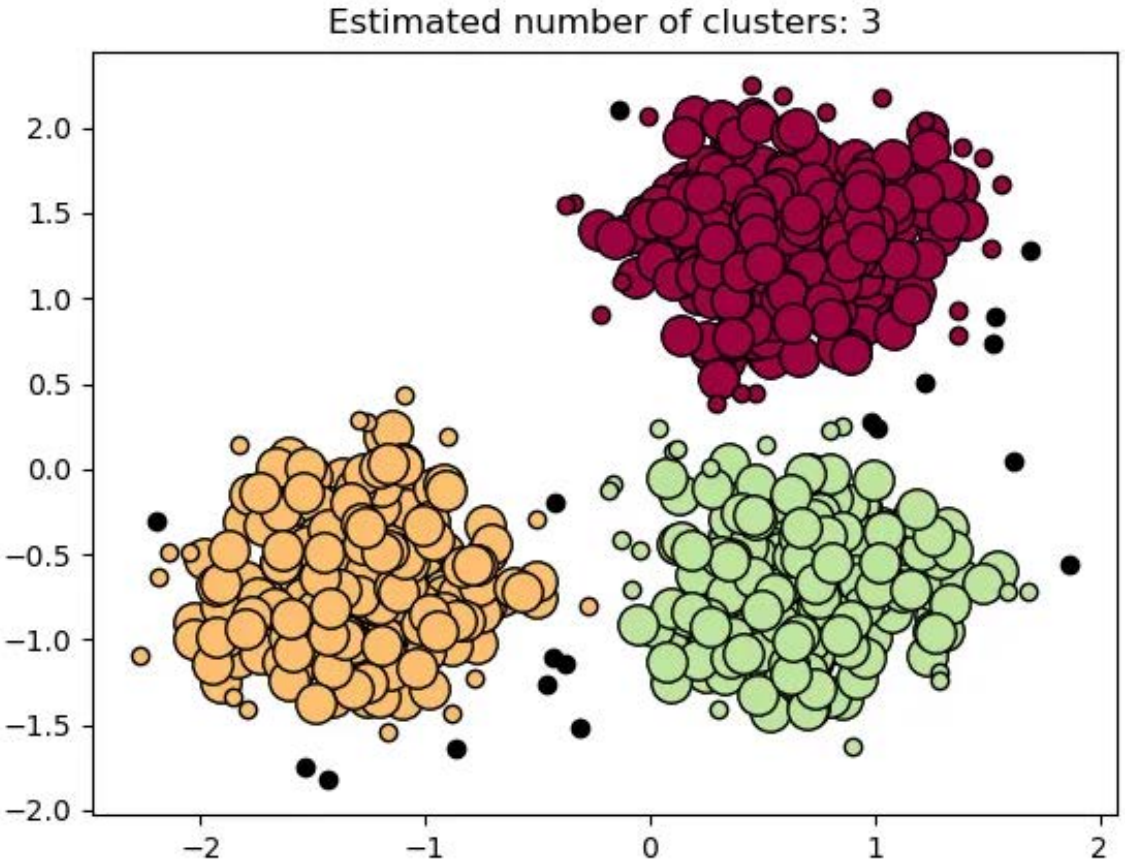
DBSCAN

Dezavantajları ise:

MinPts-ε kombinasyonu tüm kümeler için uygun şekilde seçilemediğinden, yoğunluklardaki büyük farklılıklarla veri kümelerini iyi bir şekilde kümeleyemez.

Veri ve ölçek iyi anlaşılmadıysa, anlamlı bir mesafe eşiği ε seçmek zor olabilir.

Tamamen belirleyici değildir: Birden fazla kümeden erişilebilen sınır noktaları, verilerin işlenme sırasına bağlı olarak her iki kümenin de parçası olabilir.



Şekil 1.8 DBSCAN uygulanarak kümelenmiş bir veri seti

K- Means Uygulama

In [7]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random as rd
from collections import defaultdict
import matplotlib.cm as cm
```

In [8]:

```
dataset=pd.read_csv('Mall_Customers.csv')
X=dataset.iloc[:,[3,4]].values
```

In [9]:

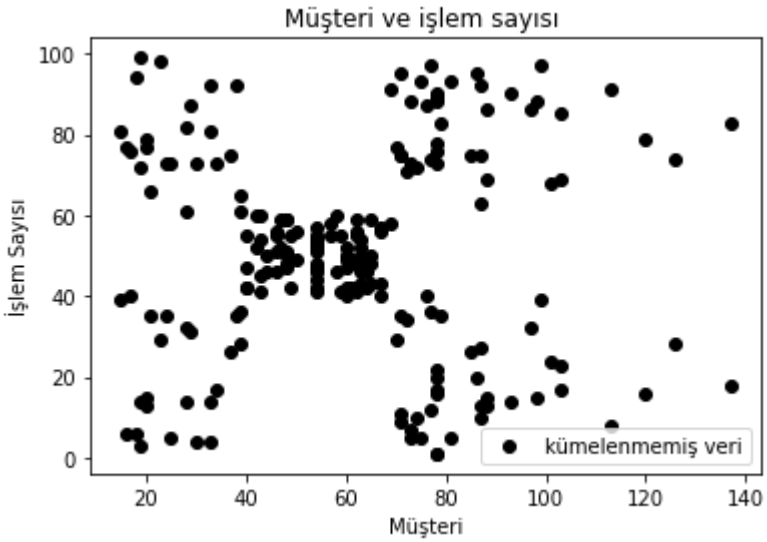
```
K=6 # küme sayısı
m=200  # gösterilecek iterasyon sayısı
Centroids=np.array([]).reshape(2,0) #verileri rastgele noktalara yerleştiriyoruz
```

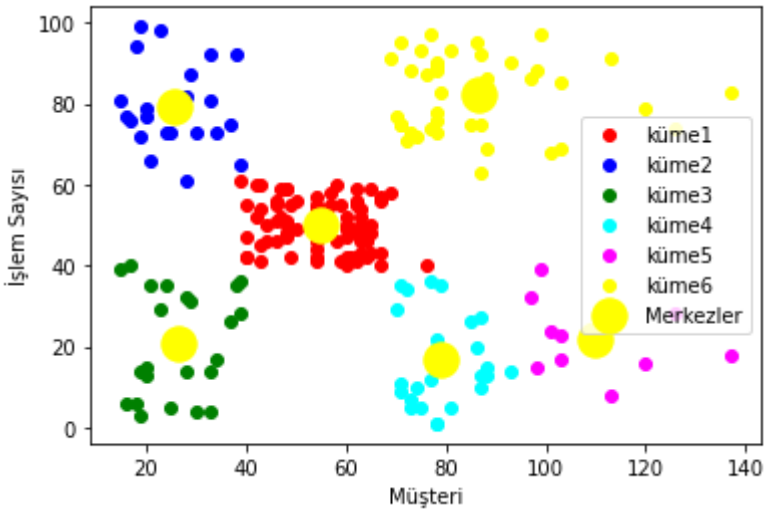
In [10]:

```
for i in range(K):
    rand=rd.randint(0,m-1)
    Centroids=np.c_[Centroids,X[rand]]
```

In [11]:

```
num_iter=100
Output=defaultdict()
Output={}
for n in range(num_iter):

    Mesafe=np.array([]).reshape(m,0)
    for k in range(K):
        tempDist=np.sum((X-Centroids[:,k])**2,axis=1)
        Mesafe=np.c_[Mesafe,tempDist]

    C=np.argmin(Mesafe,axis=1)+1

    Y={}
    for k in range(K):
        Y[k+1]=np.array([]).reshape(2,0)
    for i in range(m):
        Y[C[i]]=np.c_[Y[C[i]],X[i]]

    for k in range(K):
        Y[k+1]=Y[k+1].T


    for k in range(K):
        Centroids[:,k]=np.mean(Y[k+1],axis=0)

    Output=Y

plt.scatter(X[:,0],X[:,1],c='black',label='kümelenmemiş veri')
plt.xlabel('Müşteri')
plt.ylabel('İşlem Sayısı')
plt.legend()
plt.title('Müşteri ve işlem sayısı ')
plt.show()

color=['red','blue','green','cyan','magenta','yellow','orange','gray','black','purple']
labels=['küme1','küme2','küme3','küme4','küme5','küme6','küme7','küme8','küme9','küme10'
for k in range(K):
    plt.scatter(Output[k+1][:,0],Output[k+1][:,1],c=color[k],label=labels[k])
plt.scatter(Centroids[0,:],Centroids[1,:],s=300,c='yellow',label='Merkezler')
plt.xlabel('Müşteri')
plt.ylabel('İşlem Sayısı')
plt.legend()
plt.show()
```


Müşteri ve işlem sayısı
İşlem Sayısı
Müşteri
kümelenmemiş veri


İşlem Sayısı
Müşteri
küme1
küme2
küme3
küme4
küme5
küme6
Merkezler

Affinty Propagation Uygulama

In [5]:

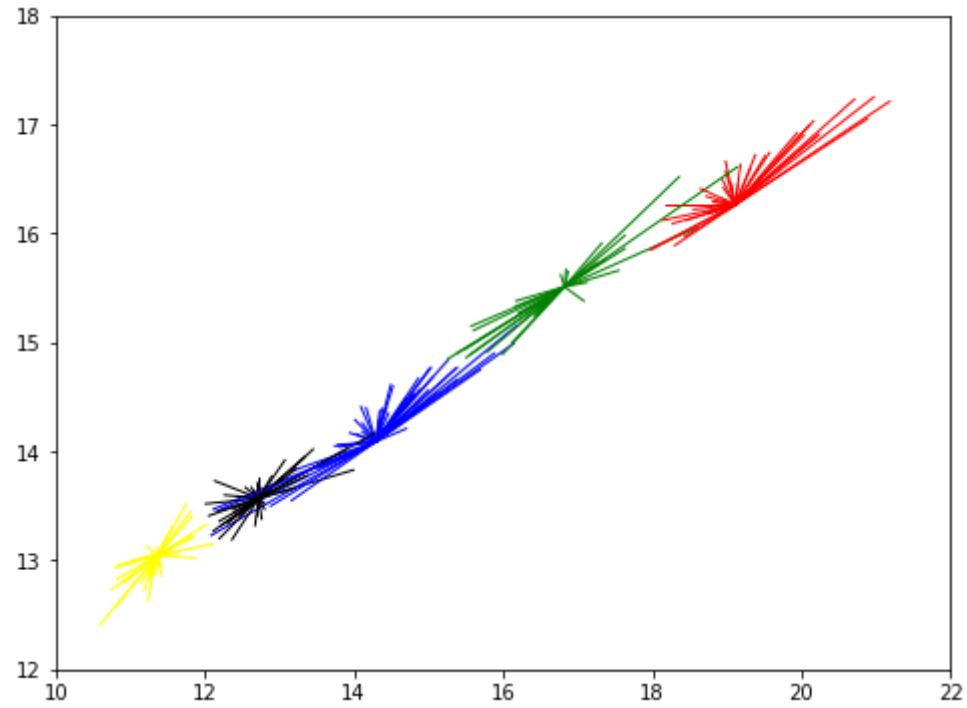
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

'''
Test datamızı okutuyoruz.
'''

def init_sample():
    data = pd.io.parsers.read_csv('SEEDS.csv')
    Xn=data.to_numpy()
    print(Xn)
    dataLen = len(Xn)
    print(type(Xn))
    return Xn,dataLen

'''
Similarity matrisini hesaplıyoruz
'''
def cal_simi(Xn):
    ##Bu veri kümesinin benzerlik matrisi iki boyutlu bir dizidir
    simi = []
    for m in Xn:
        ##Her sayının tüm sayılara, yani matristeki bir satıra benzerliğinin listesi
        temp = []
        for n in Xn:
            ##Benzerliği hesaplamak için negatif Öklid mesafesini kullanıyoruz
            #print(len(n));
            sumAll=sum_of_squares(m, n)
            s = -np.sqrt(sumAll)
            # s =-np.sqrt((m[0]-n[0])**2 + (m[1]-n[1])**2)
            temp.append(s)
        simi.append(temp)

    ##Referans derecesini, yani köşegenin değerini, genellikle minimum veya medyan değer
    #p = np.min(simi)
    #p = np.max(simi)
    p = np.median(simi)
    for i in range(dataLen):
        simi[i][i] = p
    return simi
'''
Hesaplama
'''
def sum_of_squares(m,n):
    sumAll=0;
    for i in range(len(n)):
        sumAll=sumAll+(m[i]-n[i])**2
    return sumAll;
'''
Responsibility matrisini, yani R'yi hesaplayın
       1: r(n+1) =s(n)-(s(n)+a(n))
       2: r(n+1)=(1-λ)*r(n+1)+λ*r(n)
'''

##R matrisini ve A matrisini başlatıyoruz
def init_R(dataLen):
    R = [[0]*dataLen for j in range(dataLen)]
    return R
```

```
def init_A(dataLen):
    A = [[0]*dataLen for j in range(dataLen)]
    return A

##R matrisini yinelemeli olarak güncelliyoruz
def iter_update_R(dataLen,R,A,simi):
    old_r = 0 ##Güncellemeden önceki r değeri
    lam = 0.5 ##Algoritma yakınsaması için kullanılan katsayı
    ##Bu döngüde R matrisini güncelliyoruz
    for i in range(dataLen):
        for k in range(dataLen):
            old_r = R[i][k]
            if i != k:
                max1 = A[i][0] + R[i][0]
                for j in range(dataLen):
                    if j != k:
                        if A[i][j] + R[i][j] > max1 :
                            max1 = A[i][j] + R[i][j]
                ##Güncellenmiş R [i] [k] değeri
                R[i][k] = simi[i][k] - max1
                ##Tekrar güncellemek için lamda değerini kullanıyoruz
                R[i][k] = (1-lam)*R[i][k] +lam*old_r
            else:
                max2 = simi[i][0]
                for j in range(dataLen):
                    if j != k:
                        if simi[i][j] > max2:
                            max2 = simi[i][j]
                ##Güncellenmiş R [i] [k] değeri
                R[i][k] = simi[i][k] - max2
                ##Tekrar güncellemek için lamda değerini kullanıyoruz
                R[i][k] = (1-lam)*R[i][k] +lam*old_r
    print("max_r:"+str(np.max(R)))
    #print(np.min(R))
    return R
'''
   Availability matrisini, yani A'yı hesaplıyoruz
'''
##A matrisini yinelemeli olarak güncelle
def iter_update_A(dataLen,R,A):
    old_a = 0 ##Güncellemeden önceki a değeri
    lam = 0.5 ##Algoritma yakınsaması için kullanılan katsayı
    ##Bu döngüde A matrisini güncelliyoruz
    for i in range(dataLen):
        for k in range(dataLen):
            old_a = A[i][k]
            if i ==k :
                max3 = R[0][k]
                for j in range(dataLen):
                    if j != k:
                        if R[j][k] > 0:
                            max3 += R[j][k]
                        else :
                            max3 += 0
                A[i][k] = max3
                ##Tekrar güncellemek için lamda değerini kullanıyoruz
                A[i][k] = (1-lam)*A[i][k] +lam*old_a
            else :
                max4 = R[0][k]
                for j in range(dataLen):
```

```
                    if j != k and j != i:
                        if R[j][k] > 0:
                            max4 += R[j][k]
                        else :
                            max4 += 0


                if R[k][k] + max4 > 0:
                    A[i][k] = 0
                else :
                    A[i][k] = R[k][k] + max4

                ##Tekrar güncellemek için lamda değerini kullanıyoruz
                A[i][k] = (1-lam)*A[i][k] +lam*old_a
    print("max_a:"+str(np.max(A)))
    #print(np.min(A))
    return A

'''
   Küme merkezlerini hesaplıyoruz
'''

def cal_cls_center(dataLen,simi,R,A):
    ## Küme belirlendikten sonra önceden ayarlanmış yineleme sayısına ulaşılıncaya kadar
    ## veya küme merkezi artık değişmeyene kadar yinelemeye devam edin
    max_iter = 100    ##Maksimum yineleme sayısı
    curr_iter = 0     ##Geçerli yineleme sayısı
    max_comp = 30     ##Maksimum karşılaştırma sayısı
    curr_comp = 0     ##Mevcut karşılaştırma sayısı
    class_cen = []    ##Veri noktalarının dizinini Xn'de depolayan küme merkezlerinin li
    while True:
        ## R matrisini hesapla
        R = iter_update_R(dataLen,R,A,simi)
        ## A matrisini hesapla
        A = iter_update_A(dataLen,R,A)
        ## Küme merkezini hesaplamaya başlayın
        for k in range(dataLen):
            if R[k][k] +A[k][k] > 0:
                if k not in class_cen:
                    class_cen.append(k)
                else:
                    curr_comp += 1
        curr_iter += 1
        print(curr_iter)
        if curr_iter >= max_iter or curr_comp > max_comp :
            break
    return class_cen


if __name__=='__main__':
    Xn,dataLen = init_sample()
    ##R ve A matrislerini başlatma
    R = init_R(dataLen)
    A = init_A(dataLen)
    ##Similarityi hesapla
    simi = cal_simi(Xn)
    ##Küme merkezlerinin çıktısı
    class_cen = cal_cls_center(dataLen,simi,R,A)
    #for i in class_cen:
    #    print(str(i)+":"+str(Xn[i]))
```

```
    #print(class_cen)

    ##Verileri küme merkezine göre bölme
    c_list = []
    for m in Xn:
        temp = []
        for j in class_cen:
            n = Xn[j]
            #d = -np.sqrt((m[0]-n[0])**2 + (m[1]-n[1])**2)
            sumAll = sum_of_squares(m, n)
            d = -np.sqrt(sumAll)
            temp.append(d)

        c = class_cen[temp.index(np.max(temp))]
        c_list.append(c)
    ##Görüntüleme
    colors = ['red','blue','black','green','yellow','gray', 'cyan', 'magenta','purple']
    plt.figure(figsize=(8,6))
    plt.xlim([10,22])
    plt.ylim([12,18])
    for i in range(dataLen):
        d1 = Xn[i]
        d2 = Xn[c_list[i]]
        c = class_cen.index(c_list[i])
        plt.plot([d2[0],d1[0]],[d2[1],d1[1]],color=colors[c],linewidth=1)
        #if i == c_list[i] :
        #    plt.scatter(d1[0],d1[1],color=colors[c],linewidth=3)
        #else :
        #    plt.scatter(d1[0],d1[1],color=colors[c],linewidth=1)
    plt.show()
```

```
[[15.26   14.84    0.871  ...  3.312   2.221   5.22  ]
 [14.88   14.57    0.8811 ...  3.333   1.018   4.956 ]
 [14.29   14.09    0.905  ...  3.337   2.699   4.825 ]
 ...
 [13.2    13.66    0.8883 ...  3.232   8.315   5.056 ]
 [11.84   13.21    0.8521 ...  2.836   3.598   5.044 ]
 [12.3    13.34    0.8684 ...  2.974   5.637   5.063 ]]
<class 'numpy.ndarray'>
max_r:-0.03898288602915295
max_a:-0.07518896611205676
1
max_r:1.0581476643304253
max_a:2.9402002193898604
2
max_r:1.0782886533286467
max_a:5.226233910973186
3
max_r:1.5552509894426538
max_a:13.993971383630857
4
max_r:2.6206950121533104
max_a:34.32574877955807
5
max_r:3.4294294331467015
max_a:56.15368763962509
6
max_r:4.2056380222552425
max_a:78.14197868204704
7
max_r:4.989540937664804
max_a:100.3877444709997
8
max_r:5.785067957882787
max_a:122.98774258074434
9
max_r:6.583435280402575
max_a:145.8919319524751
10
```



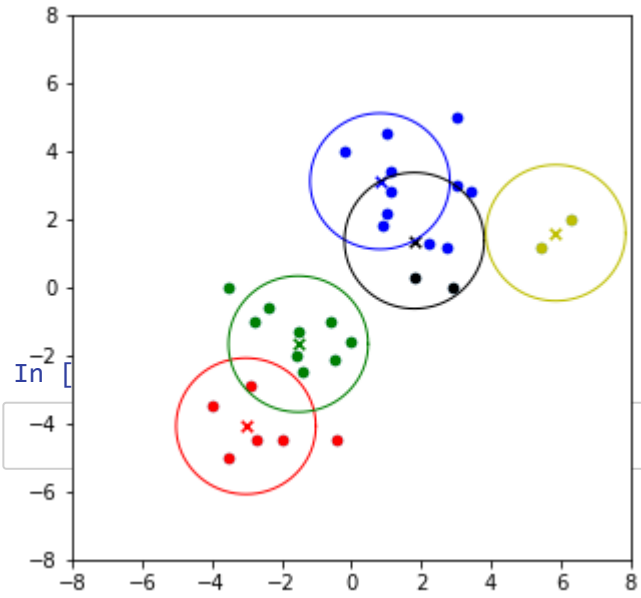
Mean-Shift Uygulama

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([
    [-4, -3.5], [-3.5, -5], [-2.7, -4.5],
    [-2, -4.5], [-2.9, -2.9], [-0.4, -4.5],
    [-1.4, -2.5], [-1.6, -2], [-1.5, -1.3],
    [-0.5, -2.1], [-0.6, -1], [0, -1.6],
    [-2.8, -1], [-2.4, -0.6], [-3.5, 0],
    [-0.2, 4], [0.9, 1.8], [1, 2.2],
    [1.1, 2.8], [1.1, 3.4], [1, 4.5],
    [1.8, 0.3], [2.2, 1.3], [2.9, 0],
    [2.7, 1.2], [3, 3], [3.4, 2.8],
    [3, 5], [5.4, 1.2], [6.3, 2]
])


def mean_shift(data, radius=2.0):
    clusters = []
    for i in range(len(data)):
        cluster_centroid = data[i]
        cluster_frequency = np.zeros(len(data))

        # Daire içindeki noktaları arıyoruz
        while True:
            temp_data = []
            for j in range(len(data)):
                v = data[j]

                if np.linalg.norm(v - cluster_centroid) <= radius:
                    temp_data.append(v)
                    cluster_frequency[i] += 1

            # Merkezleri güncelliyoruz
            old_centroid = cluster_centroid
            new_centroid = np.average(temp_data, axis=0)
            cluster_centroid = new_centroid

            if np.array_equal(new_centroid, old_centroid):
                break


        has_same_cluster = False
        for cluster in clusters:
            if np.linalg.norm(cluster['centroid'] - cluster_centroid) <= radius:
                has_same_cluster = True
                cluster['frequency'] = cluster['frequency'] + cluster_frequency
                break

        if not has_same_cluster:
            clusters.append({
                'centroid': cluster_centroid,
                'frequency': cluster_frequency
            })

    print('clusters (', len(clusters), '): ', clusters)
    clustering(data, clusters)
    show_clusters(clusters, radius)
```

```
def clustering(data, clusters):
    t = []
    for cluster in clusters:
        cluster['data'] = []
        t.append(cluster['frequency'])
    t = np.array(t)

    for i in range(len(data)):
        column_frequency = t[:, i]
        cluster_index = np.where(column_frequency == np.max(column_frequency))[0][0]
        clusters[cluster_index]['data'].append(data[i])


def show_clusters(clusters, radius):
    colors = 10 * ['r', 'g', 'b', 'k', 'y']
    plt.figure(figsize=(5, 5))
    plt.xlim((-8, 8))
    plt.ylim((-8, 8))
    plt.scatter(X[:, 0], X[:, 1], s=20)
    theta = np.linspace(0, 2 * np.pi, 800)
    for i in range(len(clusters)):
        cluster = clusters[i]
        data = np.array(cluster['data'])
        plt.scatter(data[:, 0], data[:, 1], color=colors[i], s=20)
        centroid = cluster['centroid']
        plt.scatter(centroid[0], centroid[1], color=colors[i], marker='x', s=30)
        x, y = np.cos(theta) * radius + centroid[0], np.sin(theta) * radius + centroid[1
        plt.plot(x, y, linewidth=1, color=colors[i])
    plt.show()


mean_shift(X, 2)
```

```
clusters (5): [{'centroid': array([-3.02, -4.08]), 'frequency': array
([14., 14., 10., 24., 21., 29.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.])}, {'centroid': array([-1.52222222, -1.6666666
7]), 'frequency': array([ 0.,  0.,  0.,  0.,  0.,  0., 25., 18., 16., 29.,
23., 29., 41.,
       32., 43.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.])}, {'centroid': array([0.81666667, 3.1166666
7]), 'frequency': array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
0.,  0.,  0.,
        0.,  0., 16., 33., 32., 16., 14., 16.,  0., 37.,  0., 45., 23.,
       29., 60.,  0.,  0.])}, {'centroid': array([1.8       , 1.3714285
7]), 'frequency': array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0., 25.,  0., 29.,  0.,  0.,
        0.,  0.,  0.,  0.])}, {'centroid': array([5.85, 1.6 ]), 'frequenc
y': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 4., 4.])}]
```



In [

Spectral Clustering Uygulama

In [1]:

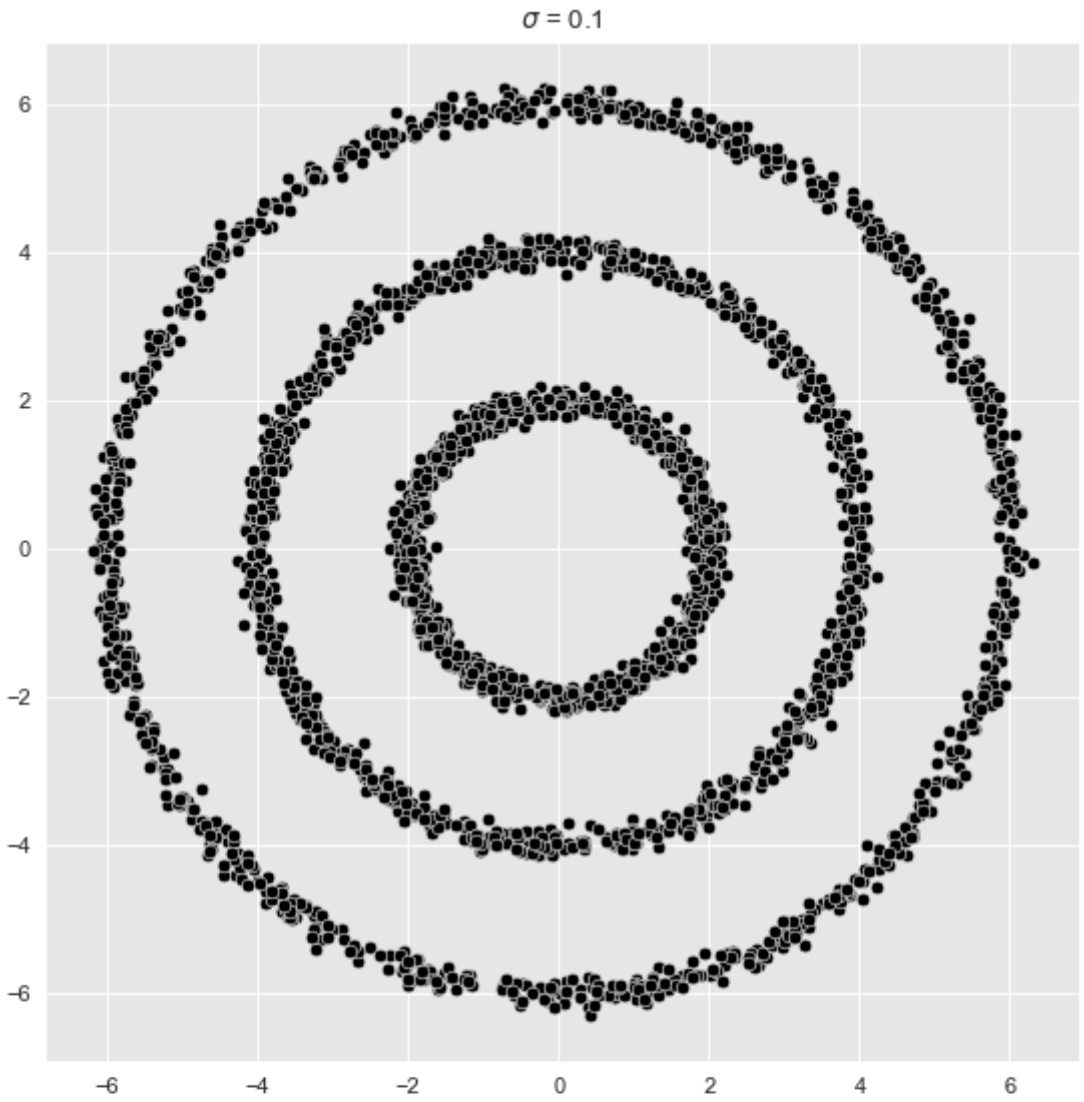
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid', {'axes.facecolor': '.9'})
sns.set_palette(palette='deep')
sns_c = sns.color_palette(palette='deep')
%matplotlib inline

from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

In [2]:

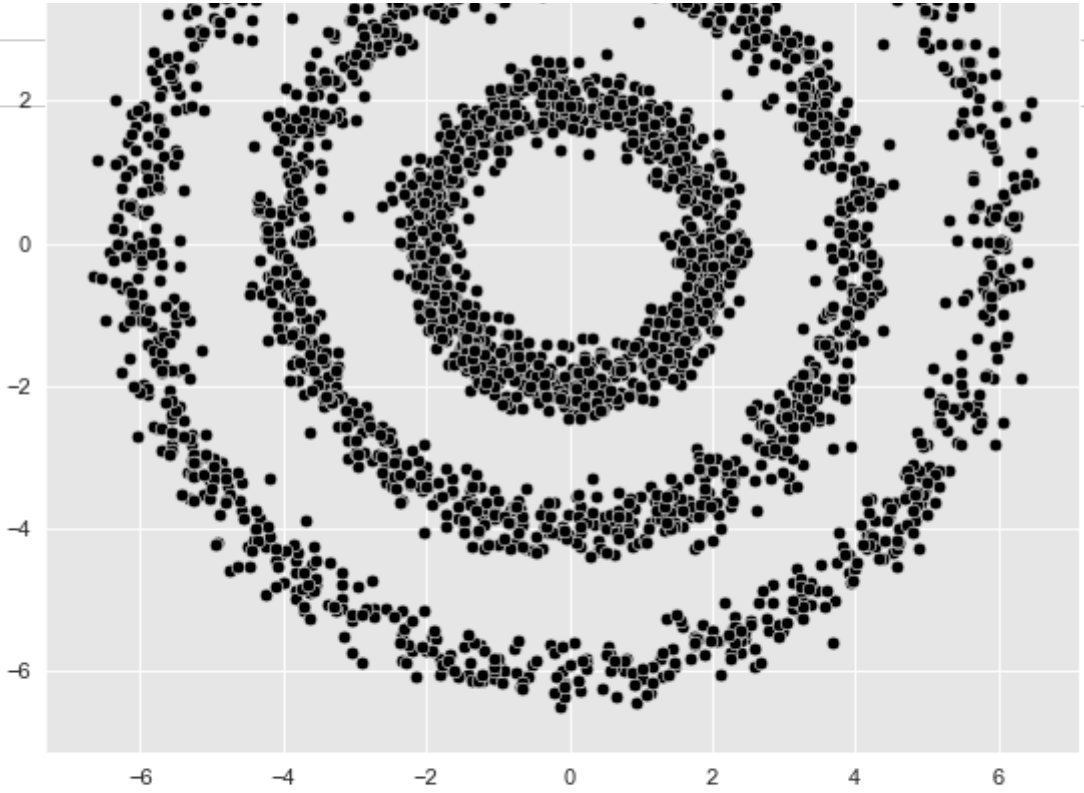
```
# Rastgele durumu ayarlayın.
rs = np.random.seed(25)

def generate_circle_sample_data(r, n, sigma):
    """Rastgele Gauss gürültüsüyle daire verileri oluşturun"""
    angles = np.random.uniform(low=0, high=2*np.pi, size=n)

    x_epsilon = np.random.normal(loc=0.0, scale=sigma, size=n)
    y_epsilon = np.random.normal(loc=0.0, scale=sigma, size=n)

    x = r*np.cos(angles) + x_epsilon
    y = r*np.sin(angles) + y_epsilon
    return x, y


def generate_concentric_circles_data(param_list):
    """Rastgele Gauss gürültüsüyle birçok daire verisi üretir."""
    coordinates = [
        generate_circle_sample_data(param[0], param[1], param[2])
     for param in param_list
    ]
    return coordinates
```

In [3]:

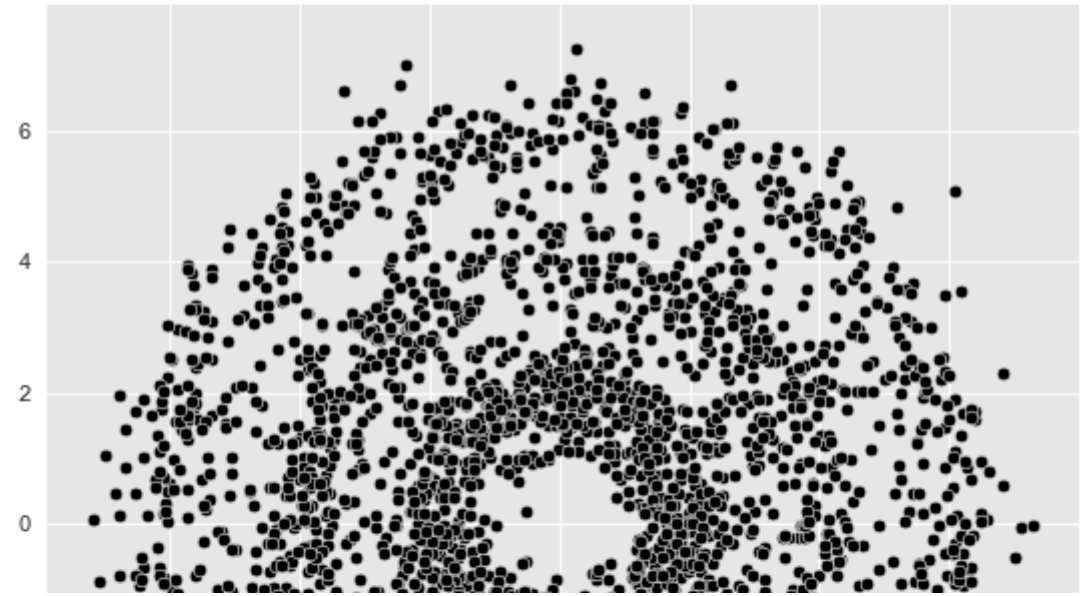
```
# Genel çizim parametrelerini ayarlayın.
plt.rcParams['figure.figsize'] = [8, 8]
plt.rcParams['figure.dpi'] = 80

# Daire başına nokta sayısı.
n = 1000
# Radius.
r_list =[2, 4, 6]
# Standart sapma .
sigmas = [0.1, 0.25, 0.5]

param_lists = [[(r, n, sigma) for r in r_list] for sigma in sigmas]
# Verileri bu listede saklıyoruz.
coordinates_list = []

fig, axes = plt.subplots(3, 1, figsize=(7, 21))

for i, param_list in enumerate(param_lists):

    coordinates = generate_concentric_circles_data(param_list)

    coordinates_list.append(coordinates)

    ax = axes[i]

    for j in range(0, len(coordinates)):

        x, y = coordinates[j]
        sns.scatterplot(x=x, y=y, color='black', ax=ax)
        ax.set(title=f'$\sigma$ = {param_list[0][2]}')

plt.tight_layout()
```

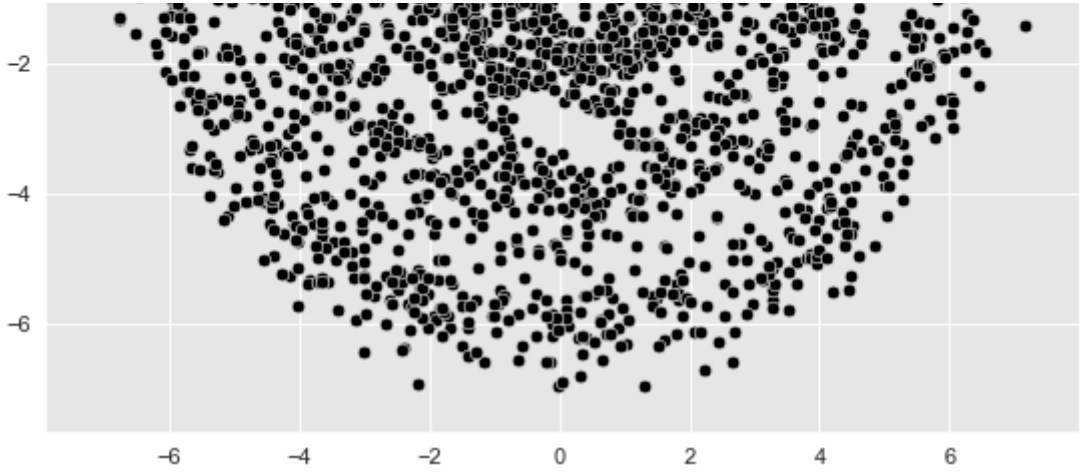

σ = 0.1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
σ = 0.25
6
4

In [ ]:



$\sigma = 0.5$




−2
−4
−6
−6
−4
−2
0
2
4
6

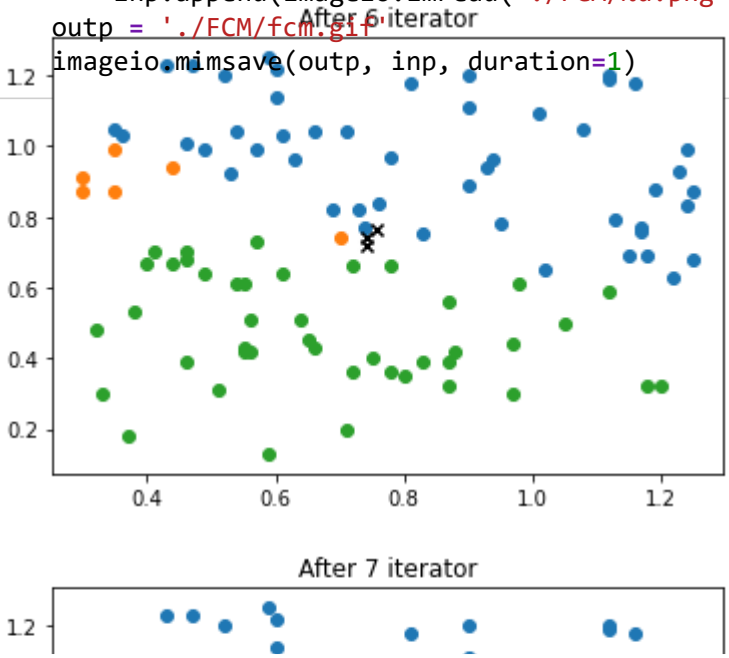
Fuzzy C Means Uygulama

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import imageio

class FCM:
    def __init__(self, data, m, c):
        self.data = data # Ham veri
        self.c = c  # Başlangıç Kümesi
        self.it = 0 # İterasyon sayısı
        self.m = m  # Fuzzy değeri

        self.N = len(self.data) # Ham veri sayısı
        self.L = len(self.c)    # Küme sayısı
        self.n = len(self.data[0])  # Veri boyutu
        self.U = np.zeros((self.N, self.L))

        self.clusterIni(0)

    def clusterIni(self, sig):
        self.cluster = {}
        for i in range(self.L):
            if i==0 and sig==0:
                self.cluster[i] = data
            else:
                self.cluster[i] = []


    def getU(self):
        # U matrisini hesaplıyoruz
        for _i,i in enumerate(self.data):
            for _k,k in enumerate(self.c):
                d = 0
                for n in range(self.n):
                    d = d + (i[n]-k[n])**2
                self.U[_i,_k] = np.power(1/d, 1/(self.m-1))

        # Veri noktasının ait olduğu sınıfı işartetliyoruz
        cluster = []
        for _u,u in enumerate(self.U):
            s = np.sum(u)
            for _l in range(self.L):
               self.U[_u,_l] = self.U[_u,_l]/s

            cluster.append(np.argmax(u))

        # Küme verilerini kaydediyoruz
        self.clusterIni(1)
        for ind,dat in enumerate(self.data):
            self.cluster[cluster[ind]].append(dat)


    def getC(self):
        # Küme merkezini yeniden hesaplıyoruz
        c = []
        for l in range(self.L):
            s1 = []
            for n in range(self.n):
                s1.append(0)
            s2 = 0
```

```
            for _i,i in enumerate(self.data):
                u = self.U[_i,l]
                for n in range(self.n):
                    s1[n] = s1[n] + np.power(u, self.m) * i[n]
                s2 = s2 + np.power(u, self.m)
            l = []
            for n in range(self.n):
                l.append(s1[n]/s2)
            c.append(l)

        if self.c == c:
            return 0
        else:
            self.c = c
            return 1

    # İterasyon
    def iter(self, it):
        for i in range(it):
            self.getU()
            b = self.getC()
            self.it = self.it + 1
            self.plot(1)
            if not b:
                print("Total iteration %d times"%(self.it-1))
                break

    def plot(self, isSave = 0):
        # Kümeleri gösteriyoruz
        for c in self.cluster:
            if(self.cluster[c] == []):
                continue
            x = np.array(self.cluster[c])[:,0]
            y = np.array(self.cluster[c])[:,1]
            plt.scatter(x, y)

        # Merkez noktalarını gösteriyoruz
        mx = np.array(self.c)[:,0]
        my = np.array(self.c)[:,1]
        plt.scatter(mx, my, marker='x', color='black')

        plt.title("After %d iterator"%self.it)
        if isSave:
            plt.savefig("./FCM/%d.png"%self.it)
        plt.show()


if __name__ == '__main__':

    f = open('./clusterData.txt', 'r')
    data = []
    for _d in f:
        dat = _d.rstrip().split(' ')
        data.append([float(dat[0]), float(dat[1])])
    f.close()
    c = [[3,3],[6,5],[10,1]]

    # FCM
    obj = FCM(data, 500 , c)
    obj.iter(10)
```

```
inp = []
for i in range(obj.it):
    inp.append(imageio.imread('./FCM/%d.png'%(i+1)))
outp = './FCM/fcm.gif'
imageio.mimsave(outp, inp, duration=1)
```



In [ ]:

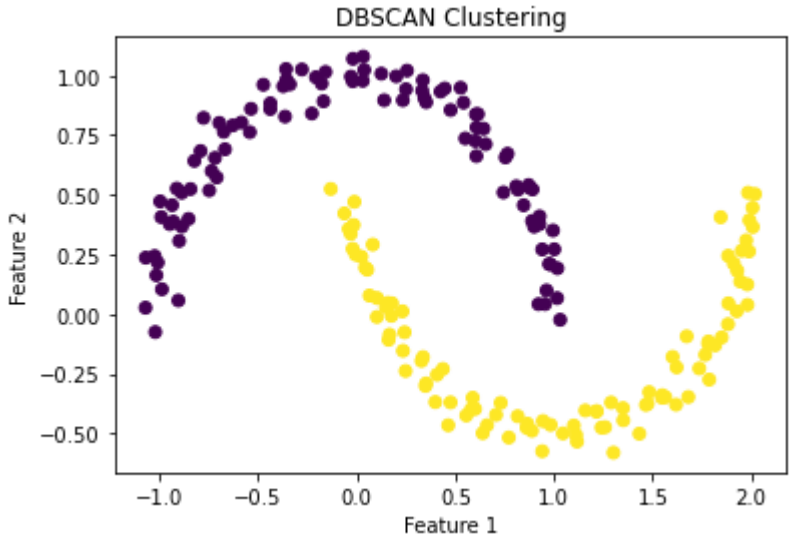
DBSCAN Uygulama

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
```

In [2]:

```
# Ay şeklinde bir veri kümesi oluşturun
n_samples = 200
X, _ = make_moons(n_samples=n_samples, noise=0.05, random_state=0)
# DBSCAN kümelemesi gerçekleştirin
dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(X)
# Veri noktalarını ve bunların küme atamalarını çizin
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title("DBSCAN Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



In [ ]:

```
```

https://medium.com/machine-learning-türkiye/kümeleme-clustering-algoritmaları-d9235bbd7946

https://www.geeksforgeeks.org/ml-classification-vs-clustering/

https://scikit-learn.org/stable/modules/clustering.html#k-means

https://www.geeksforgeeks.org/ml-mean-shift-clustering/

https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7

https://www.ibm.com/blogs/research/2018/08/spectral-clustering/

https://en.wikipedia.org/wiki/Hierarchical_clustering

https://en.wikipedia.org/wiki/DBSCAN

https://www.researchgate.net/publication/319284000_Adaptive_fuzzy_exponent_cluster_ensemble_system_based_feature_selection_and_spectral_clustering/figures?lo=1

https://stanford.edu/~cpiech/cs221/handouts/kmeans.html