MAKİNA ÖĞRENMESİ ALGORİTMALARI

KÜMELEME TEKİNİKLERİ

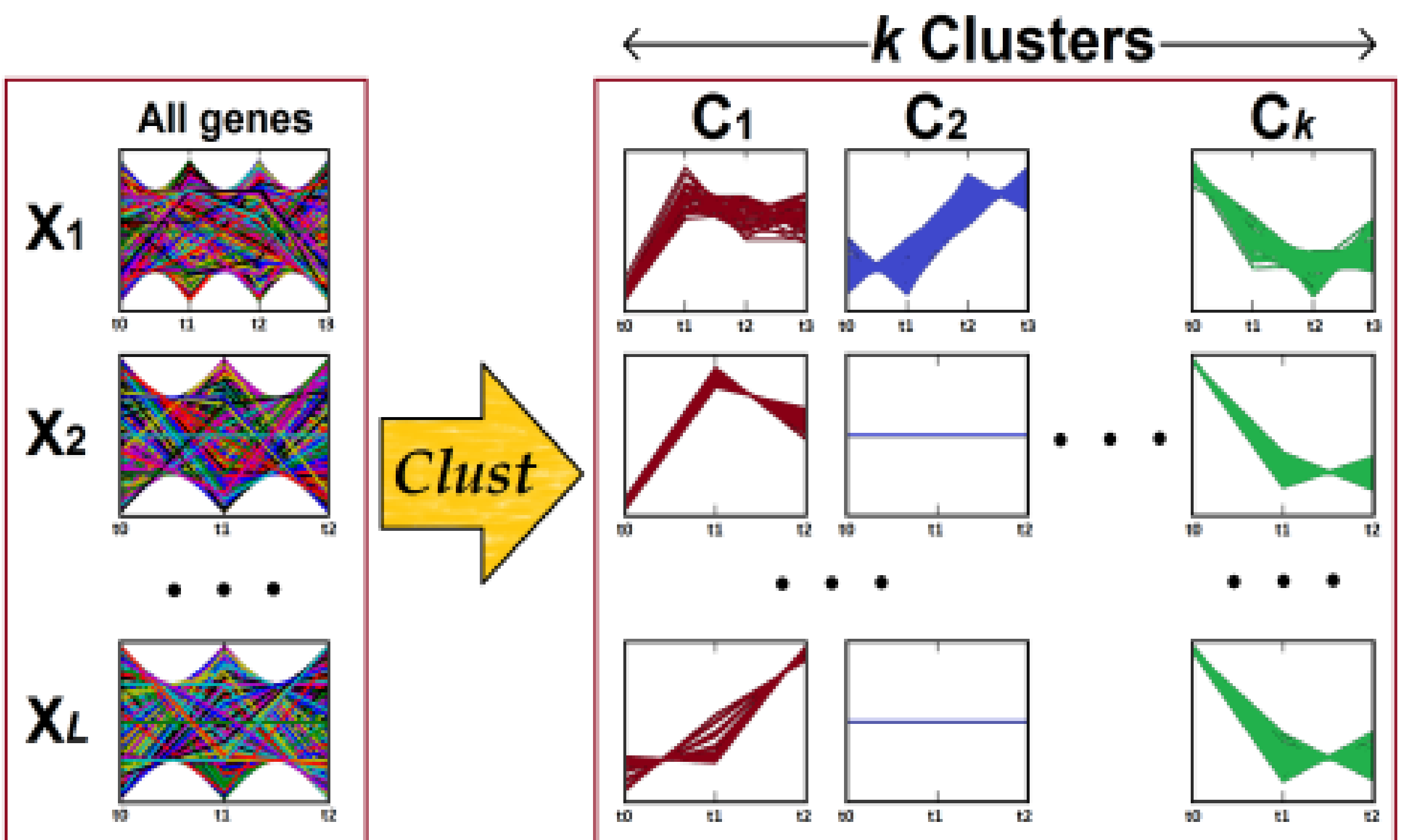
Kümeleme Teknikleri ve Python’da K-Means

•Makine öğrenmesinde elimizdeki verilerden yeni anlamlar çıkarma ve bunları sınıflandırma en ilgi çekici alanlardan biri olmuştur.

- Bir resmin içinde bir kedi mi yoksa köpek mi olduğunu anlamak için elimizdeki görsel veri setinden şekil, renk gibi özellikleri analiz edip anlamlar çıkararak en nihayetinde onları köpek veya kedi olduğuna dair sınıflandırırız. Fakat bu sınıflandırma işlemi, bizim eğitimimizde kullandığımız resimlerin kediye mi yoksa köpeğe mi ait olduğunu bilmemiz sayesinde yapılır.

•Elimizde her zaman anlamını, neye karşılık geldiğini bildiğimiz veriler olmayabilir. Kümeleme algoritmaları bu tarz verilerin sınıflandırılmasını amaçlar. Sınıflandırmadan tek farkı neyi sınıflandırdığımızı bilmememizdir.

- Gerçek hayatta bu tarz kümeleme işlemleri çok önemli alanlarda kullanılmakta. Örneğin elimizde bir canlının genetik özellikleri olduğunu varsayalım; bazı türlerin veya genlerin normal şartlarda fark edilemeyen özellikleri veya bağlantıları olabilir. Bu durumda, kümeleme teknikleri bilim insanlarına fayda sağlayacak ortaklık, benzerlik ve kalıpları gösterebilir. Tabii ki bu benzerlik bilgisayara bir şey ifade etmeyeceği için yorumu bilim insanlarına kalır.

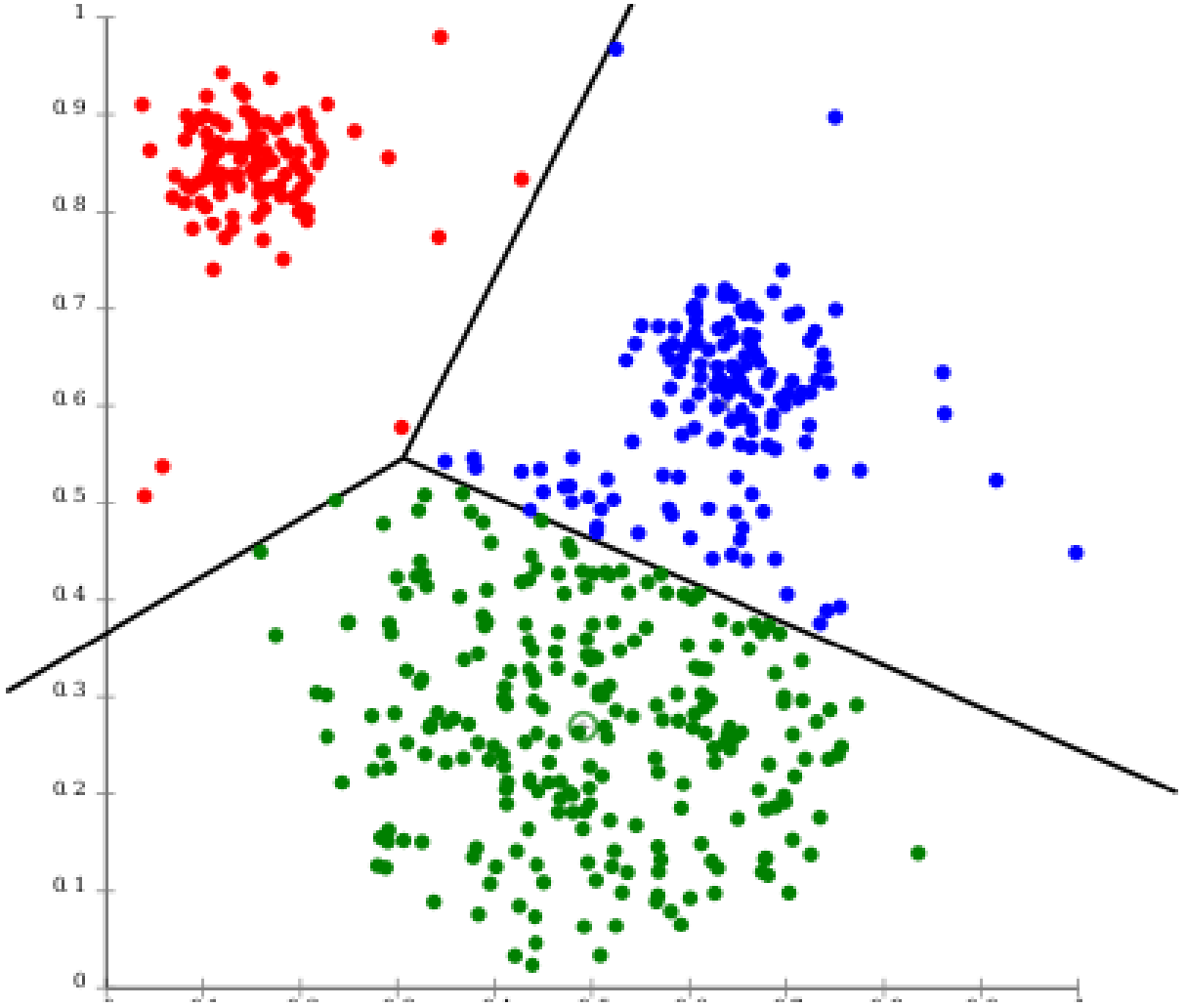

k Clusters
All genes
C1
C2
Ck
X1
X2
XL
Clust

•Aynı şekilde veriler için bir boyut küçültme tekniği olmasının yanı sıra resimlerde kalıp tespiti, deprem analizi ve pazarlama gibi alanlarda da kümeleme teknikleri kullanılmakta.

- Elimizdeki verilerin anlamı/etiketi olmadığı için kümeleme, gözetimsiz öğrenme (unsupervised learning) türüne girer. Temel amacımız bu anlamsız verilerden birbirine benzeyenleri belirli bir sayıda kümeye bölmektir.
- Bu yazımızda ilk başta kümelemenin en temel algoritması olan K-Means'i anlayıp makine öğrenme kütüphanesi kullanmadan Python'da kodladıktan sonra yumuşak (Fuzzy C-Means, EM (Expectation Maximization) algoritmasının GMM'lerde (Gaussian Mixture Models kullanımı) ve hiyerarşik kümeleme gibi daha kompleks tekniklere, matematiksel detaylarına çok fazla girmeden nasıl işlediğini ve nerelerde nasıl kullanıldığını anlayacağız.

K-Means Algoritması

Bu algoritmada ‘K’ parametresi elimizdeki verinin kaç tane kümeye ayrılacağını belirtiyor. Bu parametrenin seçimi için birkaç analiz yöntemi olsa da en iyisi algoritmayı farklı k değerlerinde yürütüp işimize en çok yarayanı almaktır. Çünkü farklı sayıda gruplar, farklı özellikleri yüz üstüne çıkarabilir.



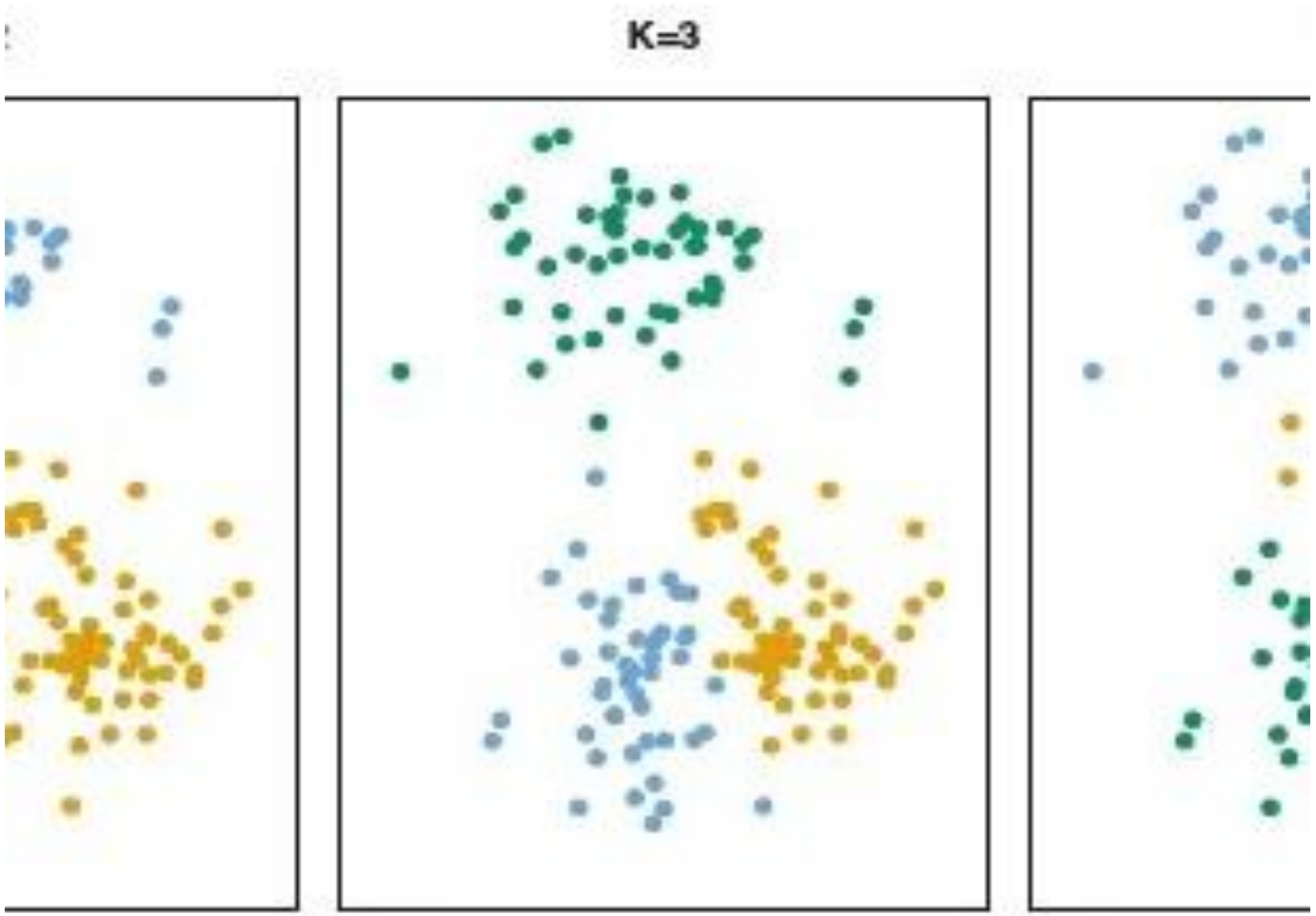
Algoritmaya bakarsak;

K adet rastgele merkez nokta (centroid) oluştur.

— Örn. 2 boyutlu düzlemde k=3 ise merkez noktalarımız (0,0)-(10,10)-(20,8) olabilir...

— Bu merkez noktalarımız en sonunda k adet kümemizin merkezlerini temsil edecek. (Dolaylı olarak kümemizi temsil ediyor)

— Amacımız ise bu rastgele tanımlanmış noktaları yavaş yavaş değiştirerek ideal kümelerimizin merkezine oturtmak



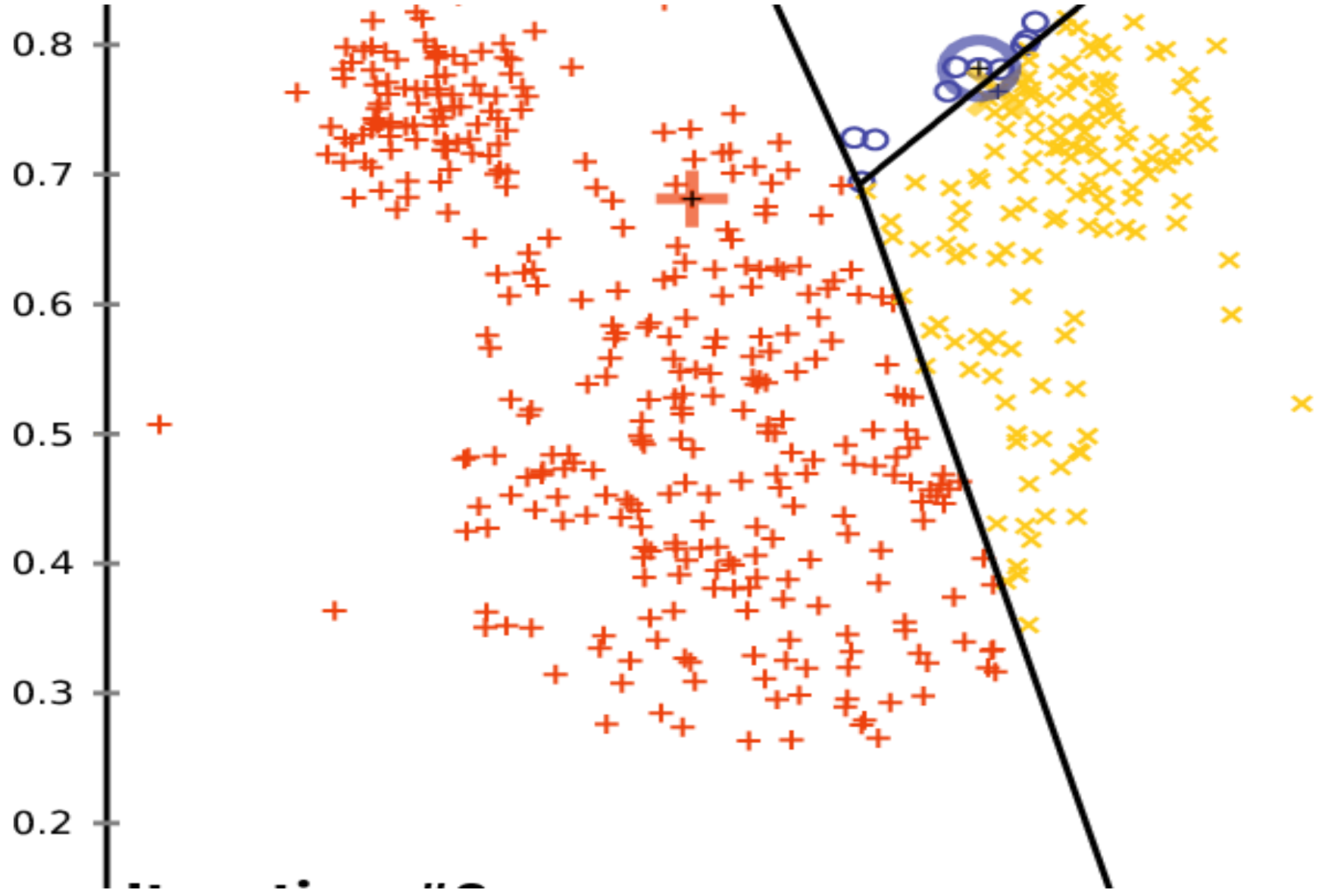
Kümelerin içindeki noktaların değişmesi azalınca veya artık ilerleme anlamsız olana kadar: (Anlamsızlığın neye göre olacağını tartışacağız)

1-) Her bir örneği en yakınındaki merkez noktasının kümesine ata

— Bunun için bir mesafe fonksiyonu seçmemiz lazım. 2 noktanın arasını öklid uzunluğuyla hesaplayabiliriz

2-) Merkez noktaların her birini, bu atanan noktaların ortalaması olarak güncelle

— Yani merkez noktalarımızı oluşan kümelerin ortasına yerleştirip tekrar tekrar hesaplama yapıyoruz


0.8
0.7
0.6
0.5
0.4
0.3
0.2

•Fark edebileceğiniz gibi bu algoritmada başlangıç merkez noktalarının seçimi kümelemeyi direkt olarak etkileyen bir faktördür. Farklı başlangıç noktaları farklı şekilde kümelemelere sebep oluyor. Bunları rastgele seçmek yerine biraz daha mantıklı seçimler yapabiliriz. Örneğin birbirinden uzak noktaları seçmek gibi. Başka birkaç tane yöntem olsa da her senaryoya uyum sağlamadığı için hala bu noktaların seçimi ucu açık bir araştırma konusudur. hat

- Peki algoritmayı ne zaman durduracağız? Bunun için en sağlıklı yöntem bir optimizasyon kriteri oluşturmaktır.

- Amacımız J fonksiyonunu olabildiğince düşük tutmak. Yani küme içi uzaklıkların (intra-cluster distance) toplamını en düşük hale getirmektir. Bunu daha açıklayıcı anlatmak gerekirse: Her küme için, içindeki her noktanın merkez noktasına olan uzaklığını topluyoruz. Başka bir optimizasyon örneği olarak merkez noktaların birbirine olan uzaklığını (inter-cluster distance) maksimize edebilme şansımız var. Ya küme içi noktalar birbirine çok yakın olsun ya da kümeler birbirinden olduğunca uzak olsun.

- J fonksiyonumuz kümeler daha kompakt hale geldikçe azalacak ve en nihayetinde bu değer çok az değişmeye başladığı zaman algoritmayı durduracağız. (threshold)

- Ayrıca algoritmanın kaç defa tekrar edeceğini önceden belirleyebiliriz. (max_iter)

- Gelin şimdi K-Means'i Görsel 5'deki optimizasyon kriterine göre makine öğrenmesi kütüphanelerini kullanmadan Python'da kodlayalım. Önceki perceptron yazımda kullandığım veri setini kullanacağım. (Sette ilk üç A değerini (0,5), (0,4), (0,3) olarak değiştirdim) Tabii ki noktaların A veya B olup olmadığını bilemeyeceğimiz için kodda sileceğiz.

```
import numpy as np
import pandas as pd
from math import sqrt
import csv
```

```
data_file = "Example.tsv"
k = 3
max_iter = 100 #Maksimum yürütme sayısı
threshold = 0.5 #Hata fonksiyonun değişimi bu sayının a

#Merkez noktalarımızı (centroid) dict şeklinde kaydetmer
#Hem küme etiketini hem nokta değerlerini tutabiliriz
centroids = {}
```

Ardından verimizi düzenleyelim:

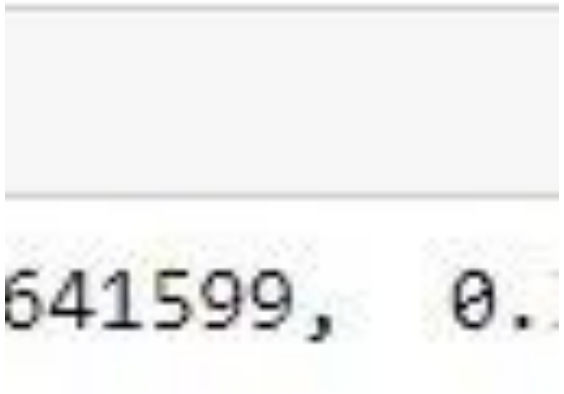
```
df = pd.read_csv(data_file, sep='\t', header=None)

#Kümelemede etiketleri kullanmadığımız için sadece 1. ve 2. satırı alıyoruz
dropped = df[[1,2]]

#Panda dataframe'indeki değerleri listeye aktardık
X = dropped.values.tolist()

print(len(X))
```

```
400
```

- Öklid mesafe fonksiyonumuzu yazalım

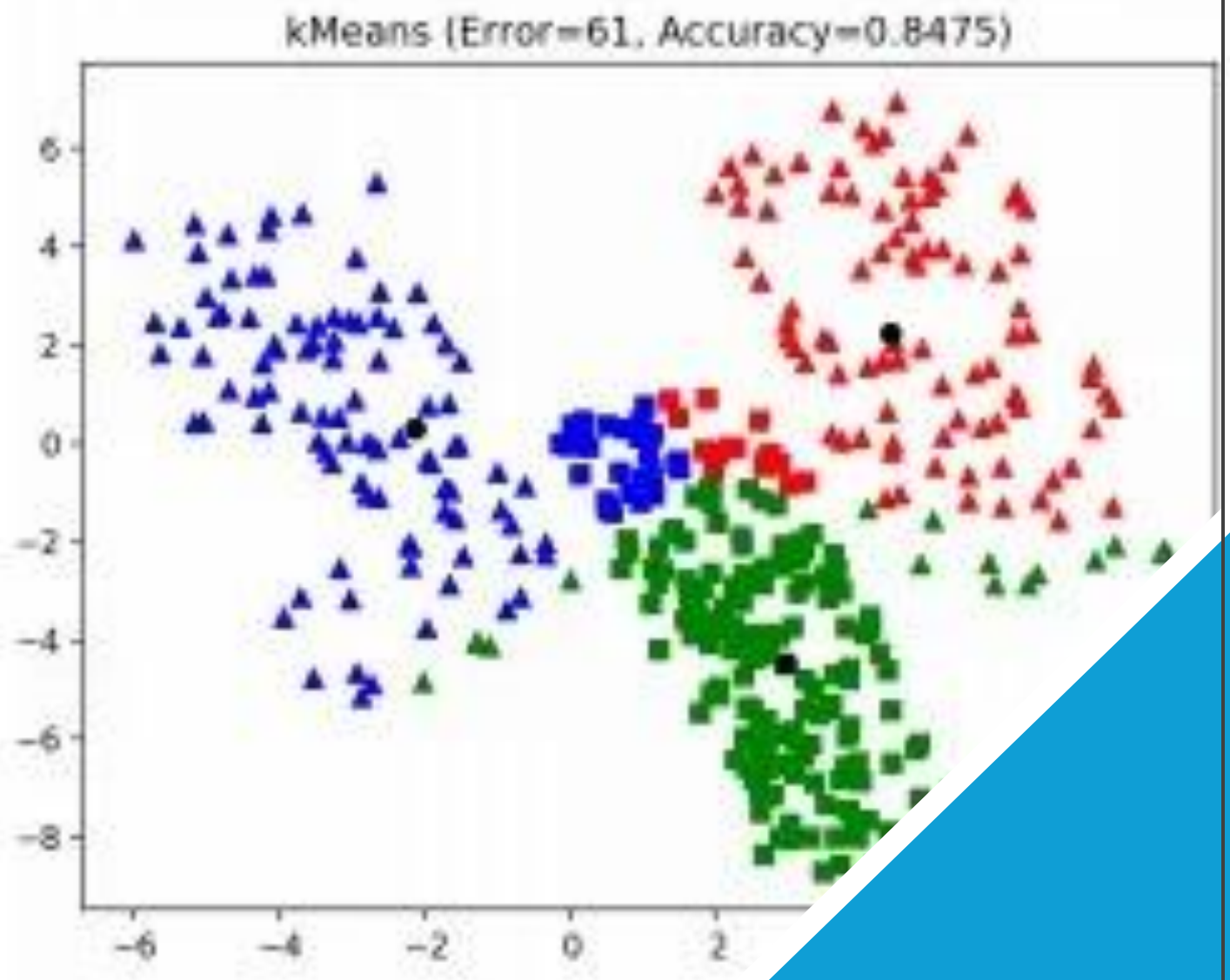
```
#Mesafe fonksiyonumuz: Öklid uzaklığı
def distance(feat_one, feat_two):
    squared_distance = 0

    for i in range(len(feat_one)):
        squared_distance += (feat_one[i] - feat_two[i])*

    return sqrt(squared_distance);
```

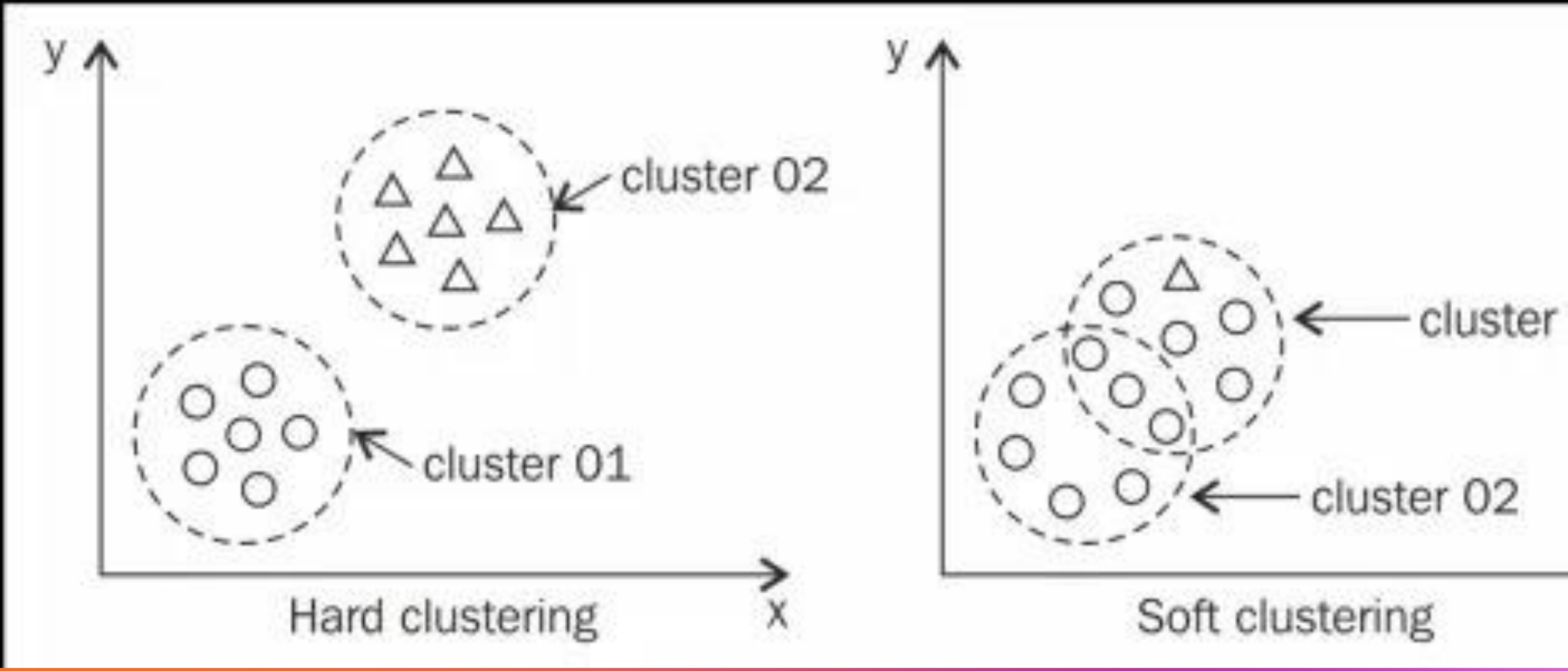
```
    tgele k merkez noktasını verimizin ilk k tane verisi olarak tanımlıyoruz
    i in range(k):
    centroids[i] = X[i]

 ıstgele centroidlerimizi listeye ekledik
 ntroid_points = []
 ntroid_points.append([centroids[i] for i in range(k)])

 Hatamızı takip edebileceğimiz bir liste
 rrors = []

#Algoritma başlar
for epoch in range(max_iter):
    classes = {}
    for i in range(k):
        classes[i] = [] #K tane kümemizi boş liste olarak tanımlıyoruz

    #Her noktanın her centroide olan uzaklığını hesapladıktan sonra en yakın olan kümeye ekliyoru
    for feature in X:
        distances = [distance(feature, centroids[centroid]) for centroid in centroids]
        classification = distances.index(min(distances))
        classes[classification].append(feature)

    error = 0

    #Hata hesaplaması: Küme içi uzaklıkların toplamı
    for j in classes.keys():
        for point in classes[j]:
            diff = distance(point, centroids[j])
            error += diff * diff


    errors.append(error)

    #Eğer hatadaki değişim sınır değerinden az ise durdur
    if epoch > 0:
        if (errors[epoch - 1] - error) < threshold:
            break

    #Merkez noktaları yeniden hesaplıyoruz
    for classification in classes:
        centroids[classification] = np.average(classes[classification], axis = 0)

    centroid_points.append([centroids[i] for i in range(3)])
```



En son merkez noktalarımız şu şekilde çıkıyor:

- Yumuşak Kümeleme Yöntemleri K-Means algoritmasında her nokta sadece bir kümeye atanıyor ve o kümenin elemanı oluyor. Bu tarz kümeleme tekniklerine “Hard Clustering (Sert Kümeleme)” deniyor. Buna karşın “Soft Clustering (Yumuşak Kümeleme)” tekniklerinde ise bir nokta birden fazla kümenin belirli bir derecede üyesi olabiliyor




y
cluster 02
cluster 01
x
Hard clustering
y
cluster
cluster 02
Soft clustering

Örneğin bir yumuşak kümeleme yöntemi olan Fuzzy (Bulanık) C-Means (FCM) algoritmasında her nokta/eleman/örnek her kümenin belirli bir olasılıkta üyesi oluyor.

Temel yapısı K-Means ile aynı olan bu algoritma üyelik (membership) fonksiyonun değişimine bağlı. Her adımda bu üyelik fonksiyonunu kullanarak merkez noktalarımızı hesaplayıp tekrar bu fonksiyonu güncelliyoruz.

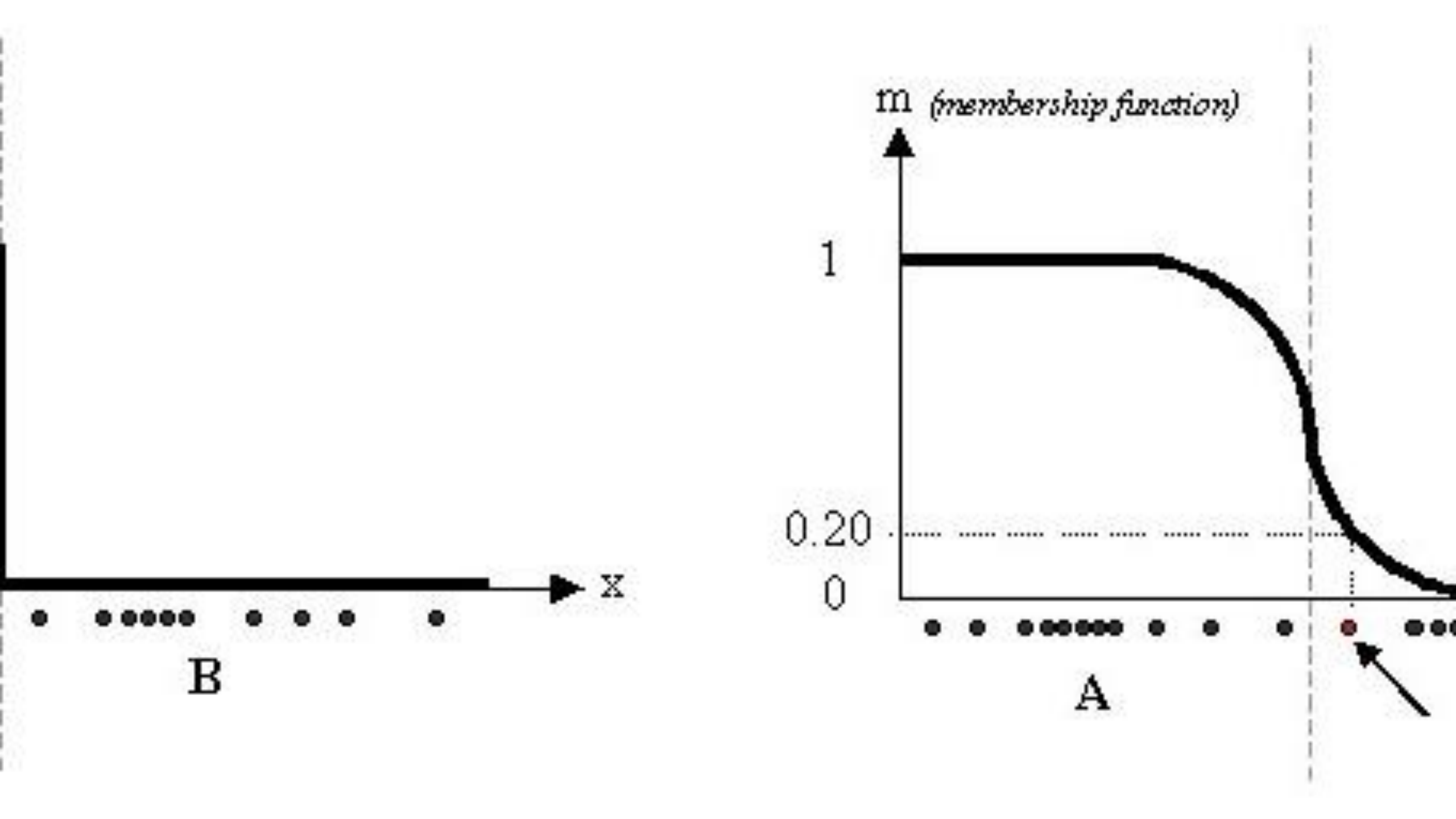
$$= \frac{\sum_{i=1}^{N} u_{\alpha i}^{m} \cdot x_i}{\sum_{i=1}^{N} u_{\alpha i}^{m}} \quad \textbf{ve} \quad u_{\alpha i} = \frac{1}{\sum_{\beta=1}^{c} \left(\frac{d_{i\alpha}^{2}}{d_{i\beta}^{2}} \right)^{1}}$$

•Bu sayede her nokta için bir üyelik (membership) vektörü oluşuyor. Küme sayısı kadar olasılık bulunduran ve toplamı 1'e eşit olan bu vektör elemanları, o noktanın ne kadar büyük bir olasılıkta kümenin üyesi olduğunu gösteriyor.

- Örneğin 3 adet küme olan bir sistemde bir noktanın üyelik vektörü şu şekilde olabilir:

- [(k1 -> 0.1) , (k2 -> 0.5), (k3-> 0.4)]

- Aşağıdaki resimde sol tarafta K-Means'in sağ tarafta ise Fuzzy C-Means'in üyelik fonksiyonlarının noktaları nasıl görüyorsunuz.


m (membership function)
1
0.20
0
x
B
A

•Yumuşak kümeleme yöntemleri doğal olarak çok fazla işlem yaptığı için sert kümeleme yöntemlerinden daha uzun sürüyor.

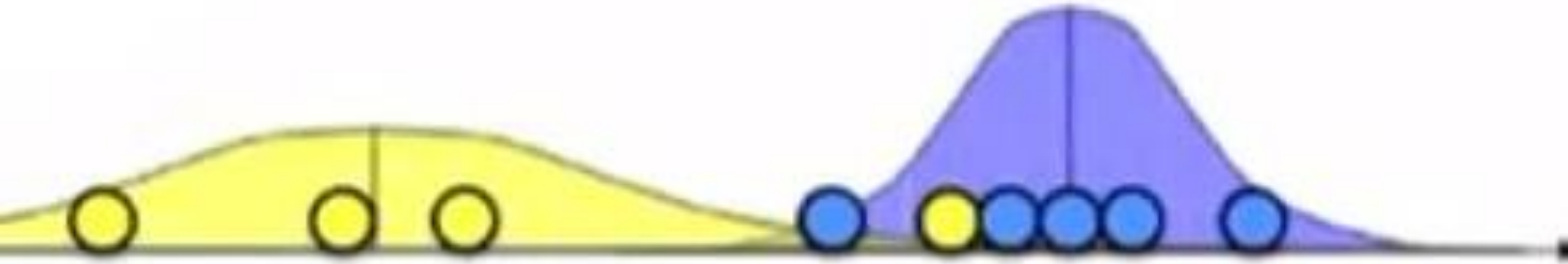
- Diğer önemli ve etkili bir yumuşak kümeleme yöntemi Gauss (Normal) dağılımlarına Expectation Maximization (Beklenti Maksimizasyonu) yaparak elde ediliyor.

- Gauss dağılımının oluşmasını sağlayan iki temel parametre olan ortalama ve varyans değerlerini değiştirerek en iyi kümelemeyi yapmaya çalışıyoruz.

- Daha iyi anlamak için isterseniz gelin hangi noktanın hangi kümeye ait olduğunu bildiğimizi varsayalım. Yani şuan verimiz etiketli olsa nasıl bir görüntü ortaya çıkacağına bakalım

$$\mu_b = \frac{x_1 + x_2 + \ldots + x_{n_b}}{n_b}$$

$$\sigma_b^2 = \frac{(x_1 - \mu_1)^2 + \ldots + (x_n - \mu_n)^2}{n_b}$$

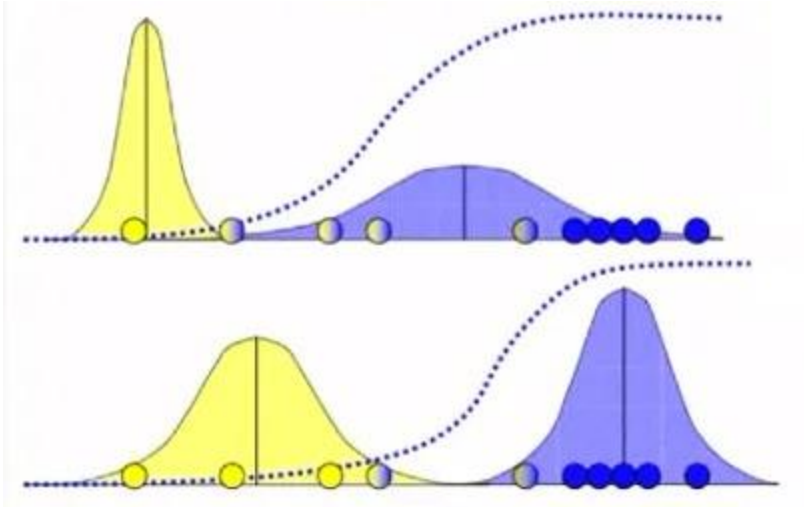


•Sarı ve mavi noktaların ortalamasını ve varyans değerini formüldeki gibi hesaplarsak bu noktaları kapsayan iki tane Gauss dağılımından oluşan kümemizi üstteki resimdeki gibi oluşturabiliriz.

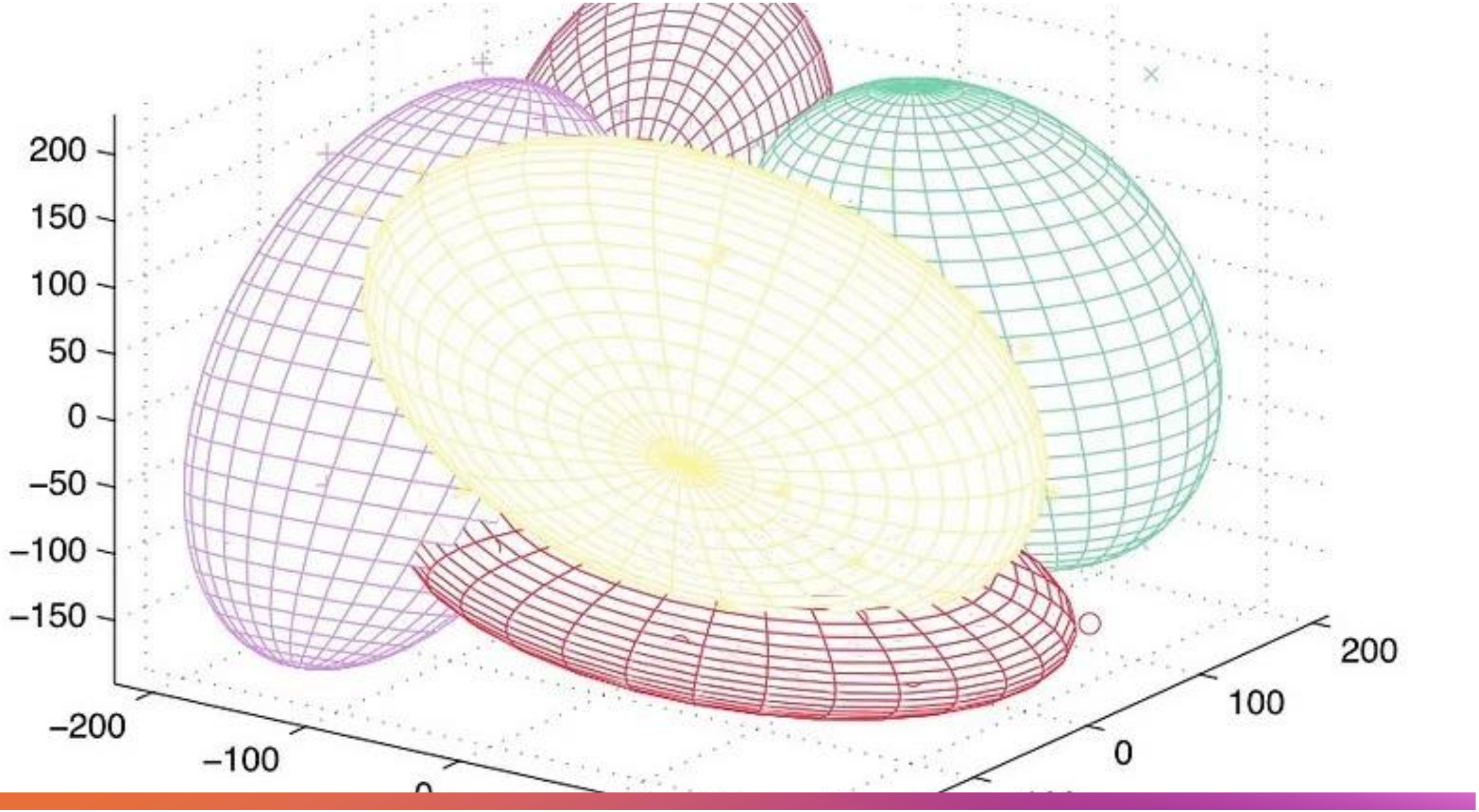
- Diğer taraftan baktığımızda da noktaların hangi kümeye ait olduğunu bilmesek ama ortalama ve varyans değerleri bize doğru bir şekilde verilse yine aynı şekilde kümeleme işlemini başarılı bir şekilde gerçekleştirebiliriz. Normalde ise bu ikisini de bilmiyoruz. Bu nedenle beklenti maksimizasyonu (Expectation Maximization) algoritmasını uygulamamız gerekiyor.

- Aynı K-Means'te olduğu gibi bu sefer merkez noktaları yerine ortalama ve varyans değerlerimizi rastgele bir şekilde tanımlıyoruz. Ardından elimizde n tane Gauss dağılımı ve bunların belirli bir olasılıkta kendilerine atanmış noktaları olacaktır.

- Örneğin rastgele iki Gauss dağılımının Görsel 11'deki gibi oluştuğunu düşünelim. Ardından yapılacak adım ise K-Means ile benzer bir şekilde bu modelleri noktaların ortasına oturtmaya çalışmak. Fakat bu sefer noktaların değerlerini toplayıp küme eleman sayısına bölmek yerine olasılık tabanlı bir şekilde yani aslında bir nevi noktalara ağırlıklar vererek yapılıyor.



Özellikle Görsel 11 ve 12'deki gibi tek düzlemdeki yani 1-boyutlu kümelendirmede K-Means'e göre çok bir avantajı yokmuş gibi gözükebilir ama boyut arttıkça ve dolayısıyla karmaşıklaştıkça çok daha iyi performans gösteriyor.


200
150
100
50
0
−50
−100
−150
−200
−100
0
100
200

•Hiyerarşik Kümeleme Yöntemleri

- Küme sayısını her zaman önceden belirlemek zorunda değiliz. Hiyerarşik kümeleme yöntemi ile veri setimiz içinde belirli sınırlara, algoritmamız bittikten sonra o sınıra göre küme sayısını seçebileceğimiz bir sistem oluşturabiliriz.

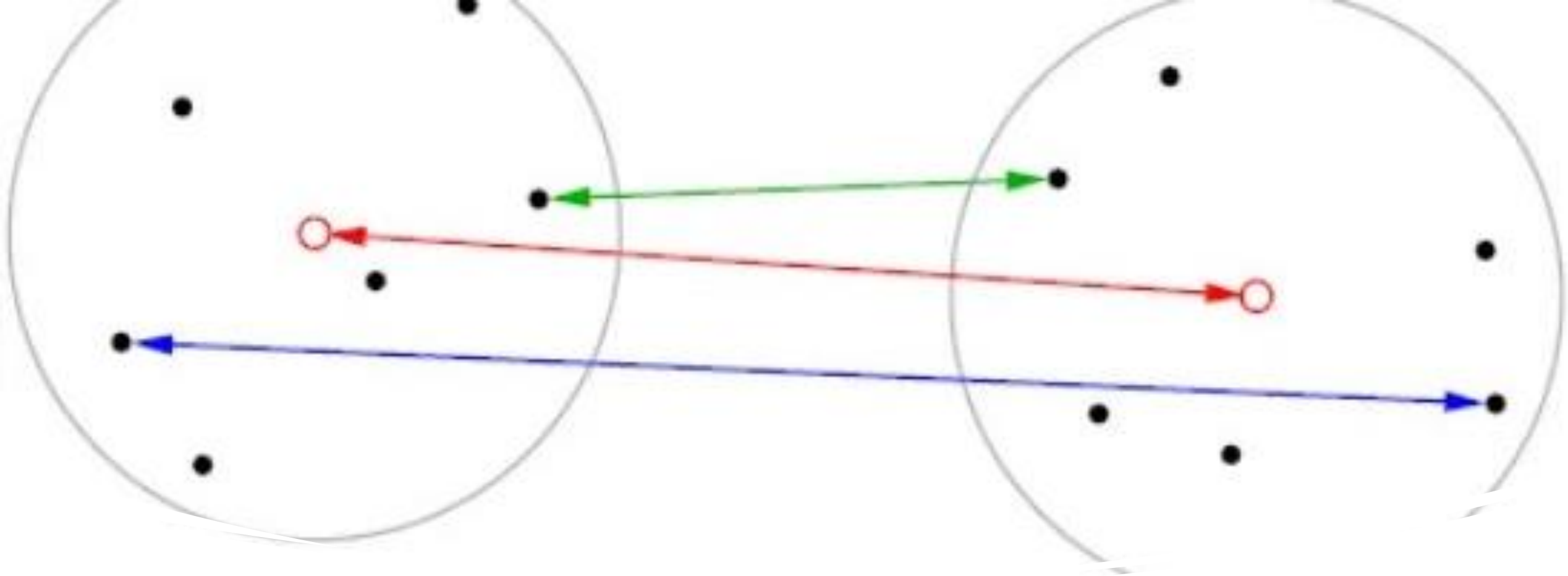
- İlk başta iki elemanlı kümelerden başlayıp bunları gittikçe büyüterek gittiğimiz yöntem "bottom-up" olarak isimlendiriliyor. Bu yöntemde;

- Her noktayı ilk başta tek başına bir küme olarak düşünüyoruz.
- Ardından,

•En son sadece tek bir küme kalana kadar:

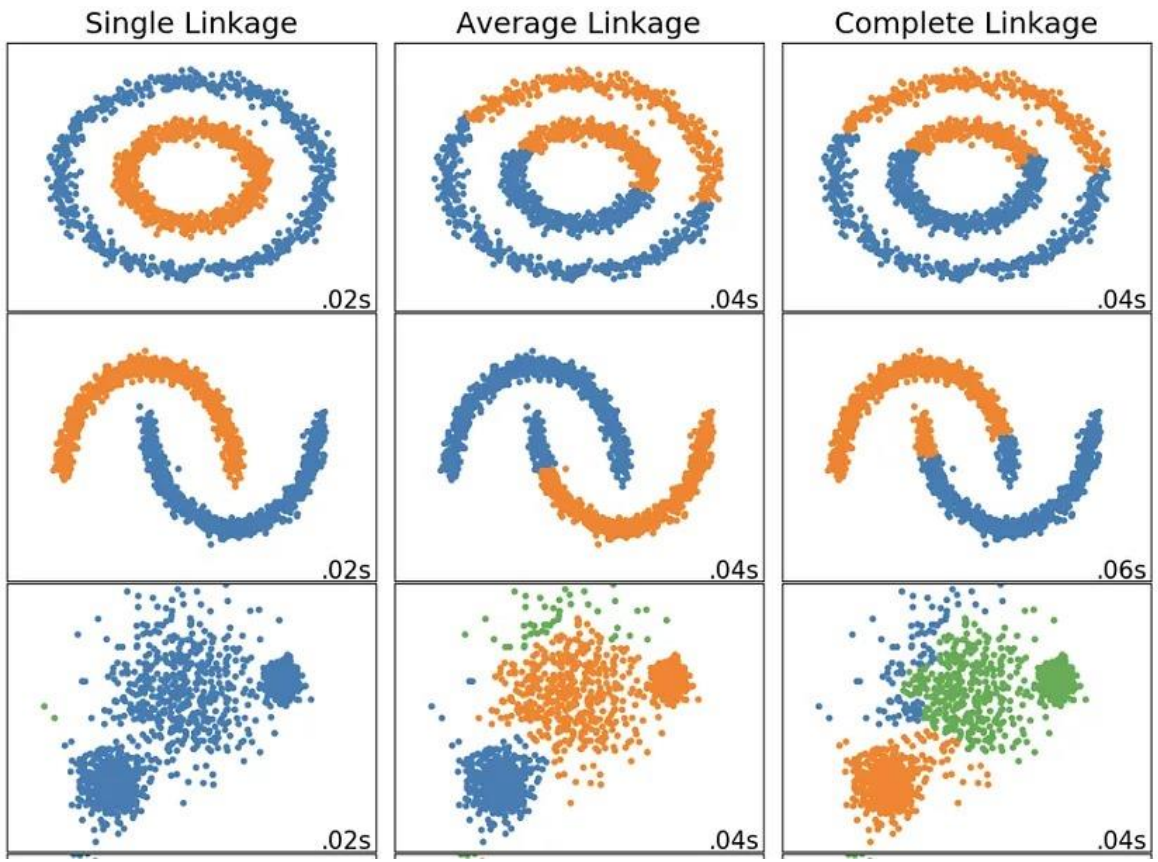
- Birbirine en benzer iki kümeyi bul ve bunları birleştir.

- Burada benzer kelimesi önemli çünkü iki kümenin neye göre benzer olacağı değişiyor. K-Means'i düşünürsek biz zaten iki noktanın benzerliğini öklid uzaklığıyla hesaplayabiliyorduk. Kümelerin benzerliğini de aynı şekilde öklid mesafesiyle hesaplayacağız ama örneğin altı elemanlı iki kümenin hangi noktalarına göre hesap yapacağımız değişiyor.

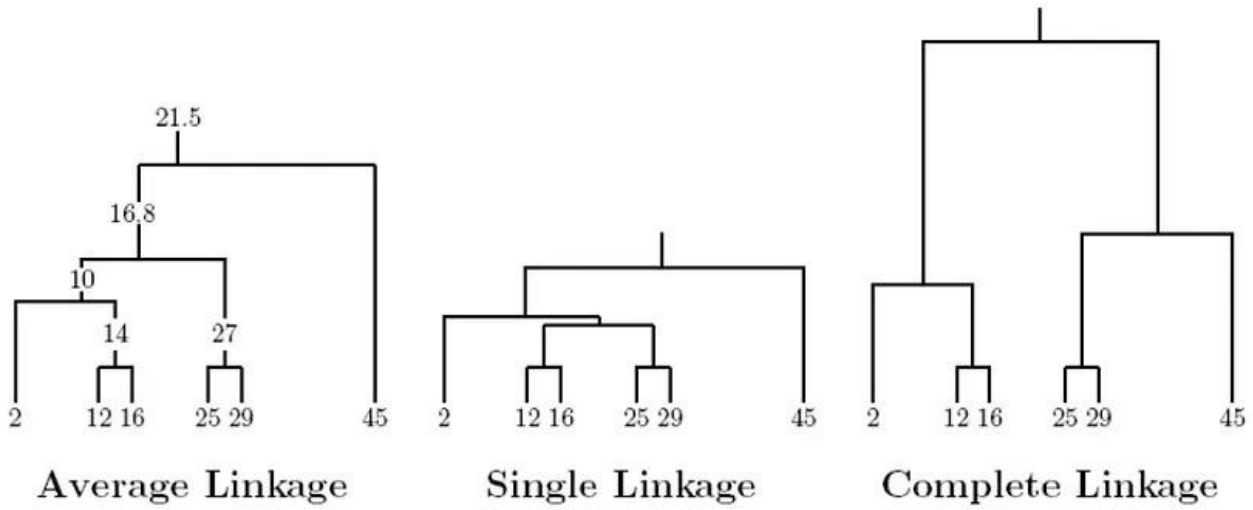


- Yukarıda gördüğünüz üç farklı metriğin her biri veri setimizde farklı şekillerde hiyerarşinin oluşmasına sebebiyet veriyor. İki kümenin benzerliğinde ya en yakın (single linkage) ya en uzak (complete linkage) ya da noktaların ortalamasının (centroid linkage) öklid mesafesini baz alıyoruz

Dolayısıyla da farklı şekilde kümelenmelere sebep oluyor:



Ayrıca dendrogram adını verdiğimiz bu hiyerarşiyi görselleştirebileceğimiz bir tablo ortaya çıkıyor:



- Görsel 16'daki dendrogramlarda her dikey çizginin uzunluğu iki küme arasındaki öklid mesafesine göre değişiyor yani aslında her yatay çizgi bir sınır değerini belirttiğini söyleyebiliriz. Buna göre de farklı sınır değerlerine göre kaç tane küme oluşacağını algoritma kendi belirleyebilir.