

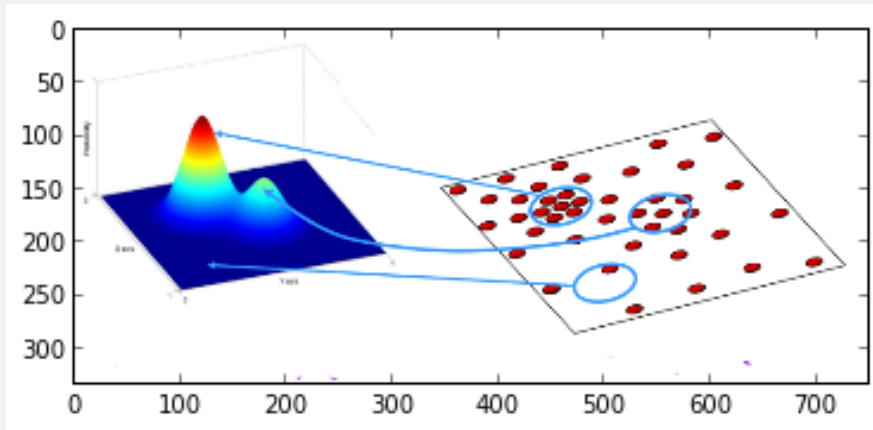
Ortalama Kaydırma ile Kumeleme (Mean Shift Clustering)

Kumeleme yapmak için bir metod daha: Ortalama Kaydırma metodu. Bu metodun mesela K-Means'den farkı kume sayısının önceden belirtilmeye ihtiyacı olmamasıdır, kume sayısı otomatik olarak metod tarafından saptanır.

“Kume” olarak saptanan aslında veri içindeki tüm yoğunluk bölgelerinin merkezleridir, yani alttaki resmin sağ kısmındaki bölgeler.

```
im=imread("dist.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0xa3f3c4c>



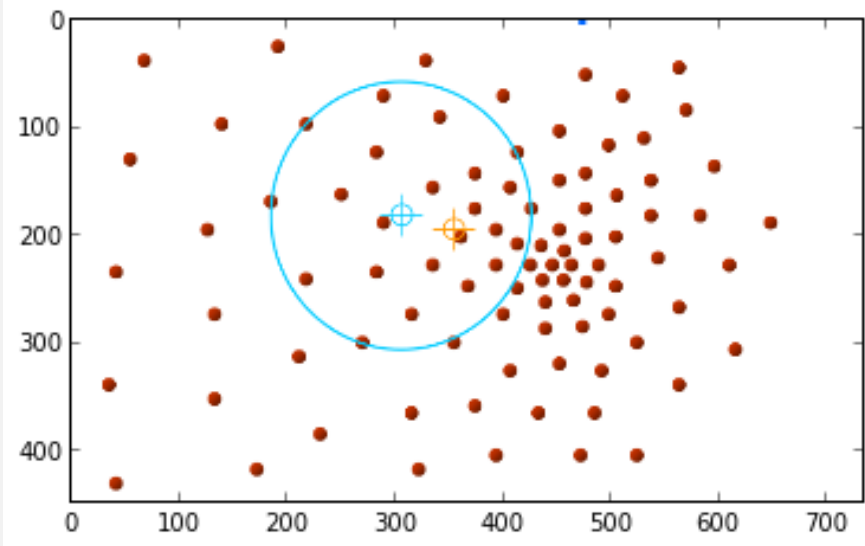
Baslangic neresidir? Baslangic tüm noktaldır, yani her noktadan baslanarak

1. O nokta etrafında (yeterince büyük) bir pencere tanımla
2. Bu pencere içine düşen tüm noktaları hesaba katarak bir ortalama yer hesapla
3. Pencereyi yeni ortalama noktayı merkezine alacak şekilde kaydır

Metodun ismi buradan geliyor, çünkü pencere yeni ortalama doğru “kaydırılıyor”. Altta bir noktadan baslanarak yapılan hareketi görüyoruz. Kaymanın sağa doğru olması mantıklı çünkü tek pencere içinden bakınca bile yoğunluğun “sağ tarafa doğru” olduğu görülmekte. Yöntemin puf noktası burada.

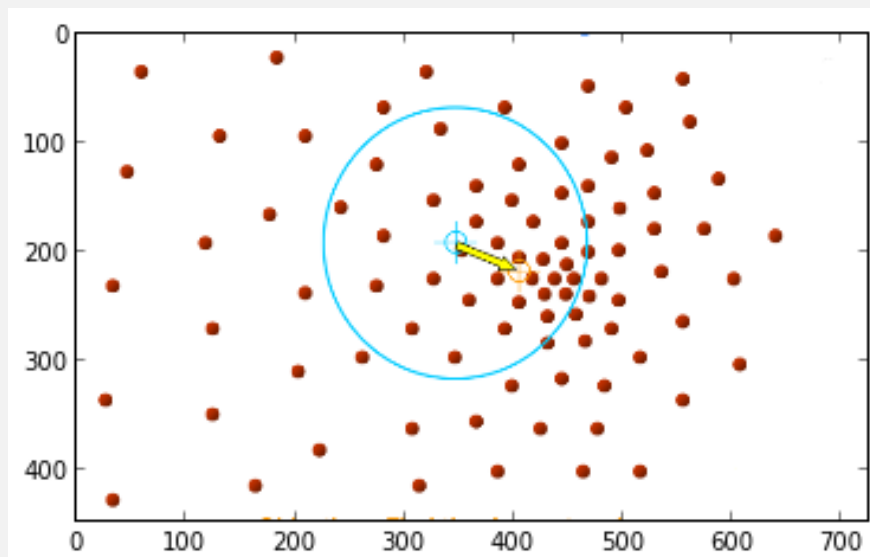
```
im=imread("mean_2.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9b966ac>



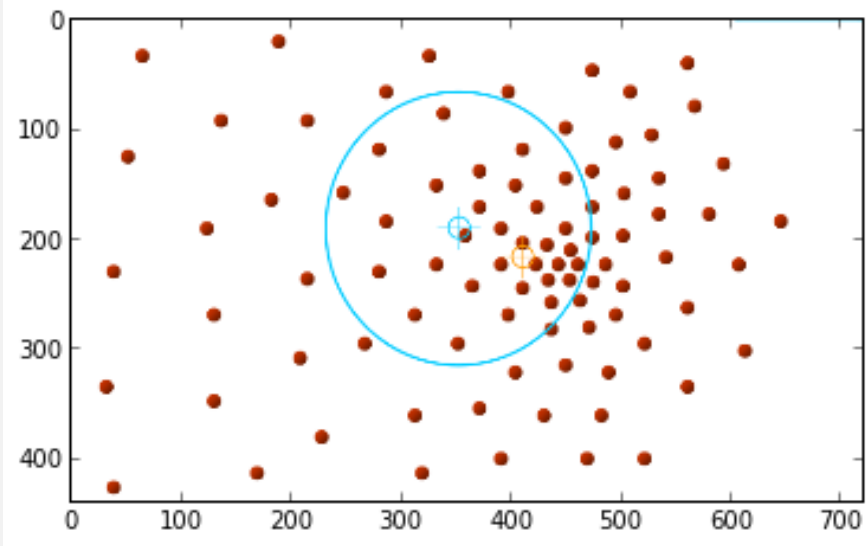
```
im=imread("mean_3.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9cd99ec>



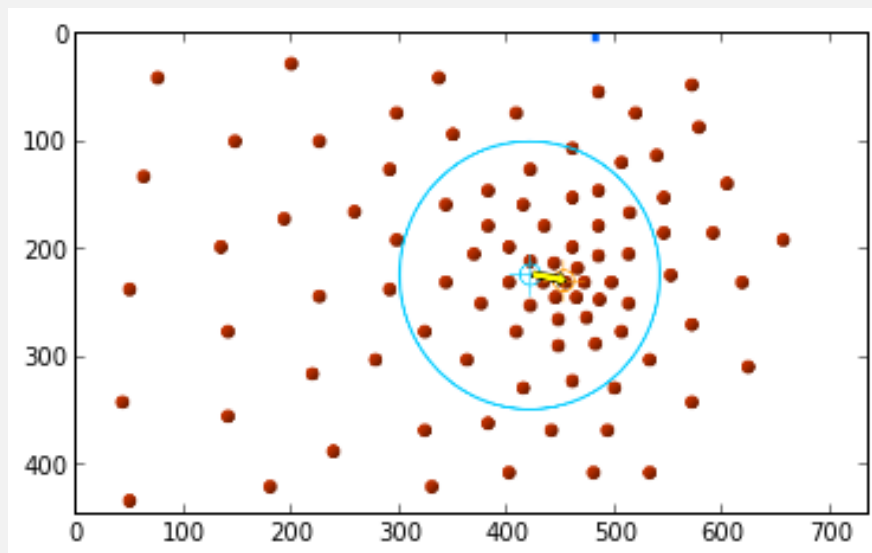
```
im=imread("mean_4.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9e3cfac>



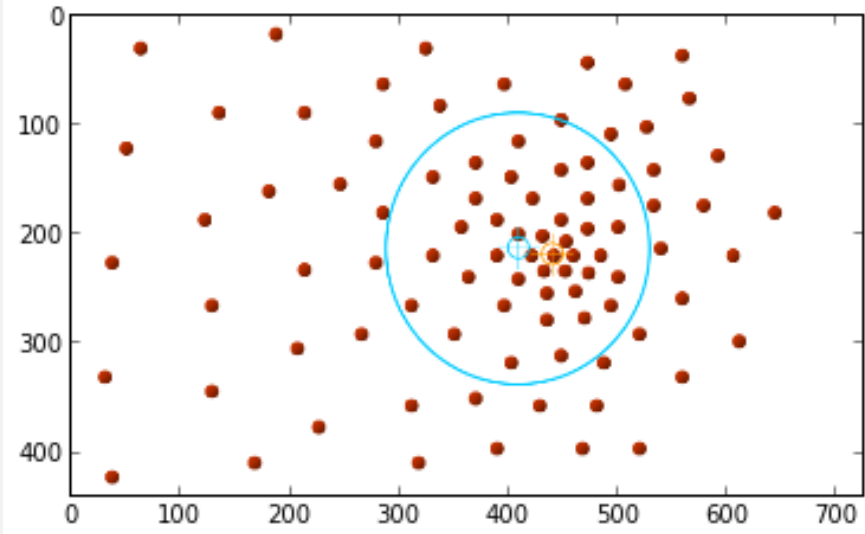
```
im=imread("mean_5.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9f9b5ec>



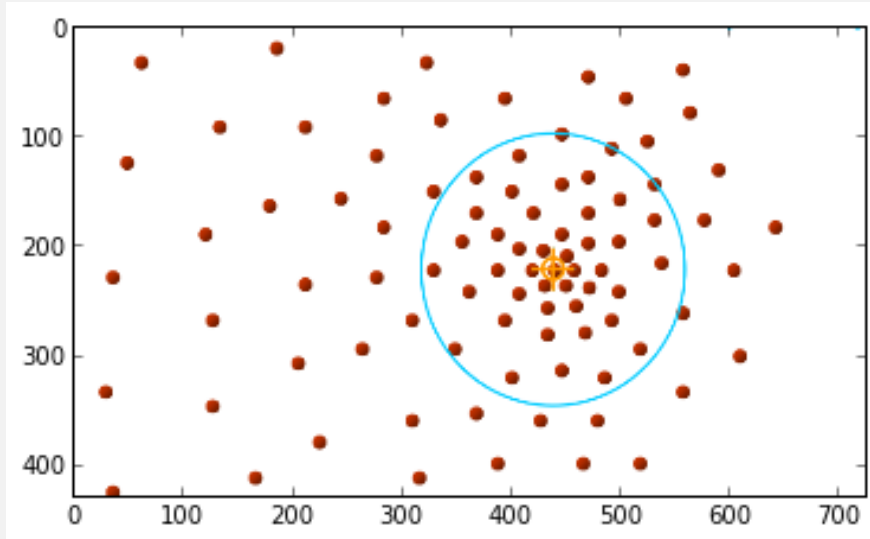
```
im=imread("mean_6.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0xa13cd0c>



```
im=imread("mean_7.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0xa2a132c>



Fakat metodu anlatan kaynaklarda pek bahsedilmeyen bir konu var, o da pencere icindeki diger noktalarin kume aidiyeti ve pencere ilerlerken o pencerenin altinda olan ve olmus tum noktalarin “bakilmis” yani “islenmis” olarak addedilmesi ve bir daha ele alinmaması. Mesela alttaki resimde sol ust kisimdaki herhangi bir noktadan (cogul) baslamissak, o bolgedeki diger noktalar da herhalde benzer pencerelerin altina dusecekler, ve yavas yavas, yeni ortalamalar sayesinde o pencereyi asagi dogru itecekler, ve soldaki yogunluk merkezini bulacagiz. Ayni sey sag tarafa da olacak. O sebeple pencere altina dusen nokta artik ayri bir sekilde baslangic olarak islenmiyor.

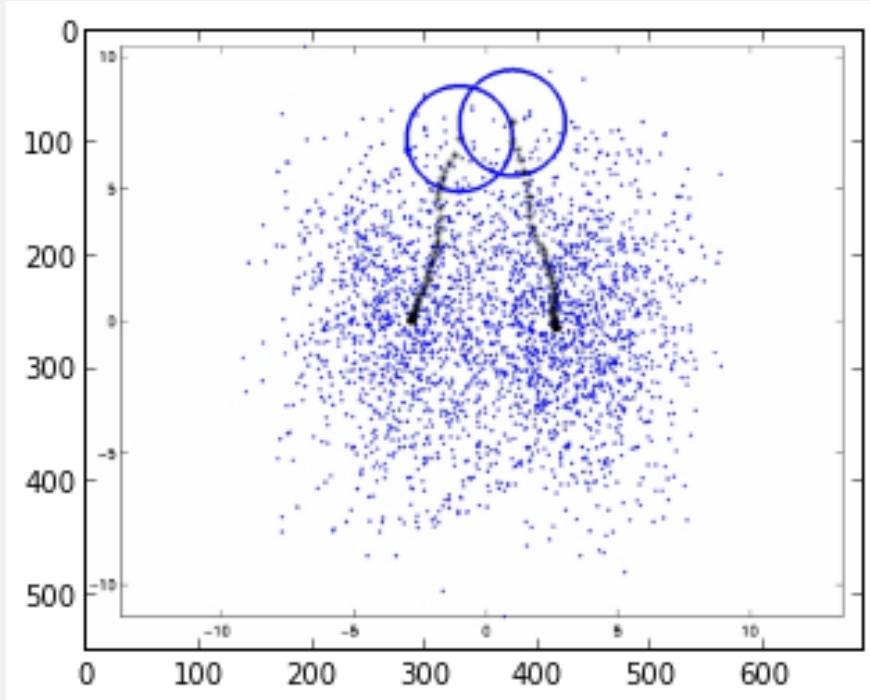
Bir diger konu: ya yogunluk merkezine cok yakin bir noktadan / noktalardan baslamissak?

O zaman ilerleme o baslangic noktasi icin aninda bitecek, cunku hemen yogunluk merkezine gelmis olacagiz. Diger yonlerden gelen pencereler de ayni yere gelecekler tabii, o zaman ayni / yakin yogunluk merkezlerini ayni kume olarak kabul etmemiz gerekir. Bu “ayni kume irdelemesi” sayisal hesaplama acisindan ufak farklar gosterebilir tabii, ve bu ufak farki gozonune alarak “kume birlestirme” mantigini da eklemek gerekiyor.

Ortalama Kaydırma sisteminde pencere buyuklugu kullanıcı tarafından tanımlanır. Fakat bu seçimin çok “hassas” bir ayar olmadigini gorduk (ki bu iyi bir sey), yani pencere buyuklugunde yapılan ufak degisimlerin bulunan kume sayisi, ve kalitesinde buyuk degisimler yaratmıyor.

```
im=imread("start.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9af0f2c>



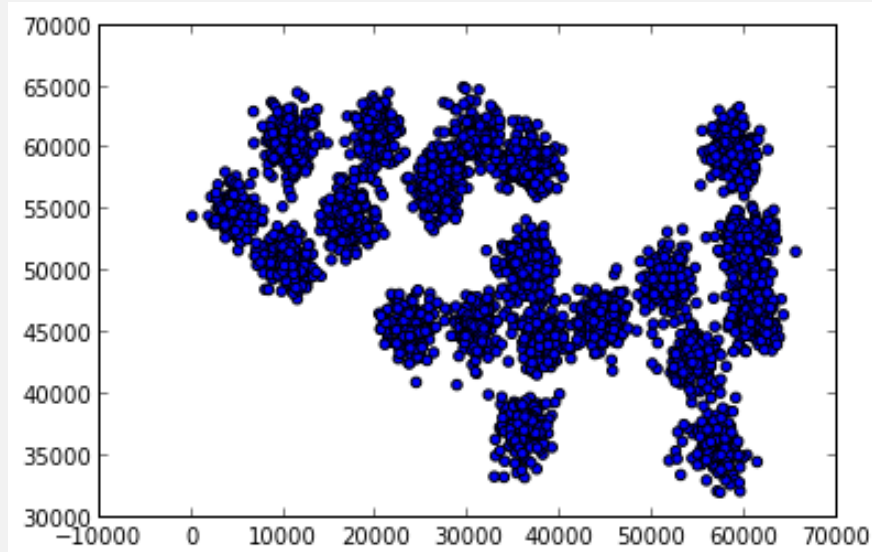
Altta ornek veri ve kodu bulabilirsiniz. Metot kume sayisi 17'yi otomatik olarak buluyor.

```
from pandas import *
data = read_csv("synthetic.txt",header=None,sep=" ")
print data.shape
data = np.array(data)
```

(3000, 2)

```
scatter(data[:,0],data[:,1])
```

<matplotlib.collections.PathCollection at 0x9fb632c>



```
from numpy import *
from numpy import linalg as la

def mean_shift(dataPts, bandwidth):
    dataPts = asarray( dataPts )
    bandwidth = float( bandwidth )
    plotFlag = False

    numDim, numPts = dataPts.shape
    numClust = 0
    bandSq = bandwidth**2
    initPtInds = arange( numPts )
    #biggest size in each dimension
    maxPos = dataPts.max(0)
    #smallest size in each dimension
    minPos = dataPts.min(0)
    #bounding box size
    boundBox = maxPos-minPos
    #indicator of size of data space
    sizeSpace = la.norm(boundBox)
    #when mean has converged
    stopThresh = 1e-3*bandwidth
    #center of clust
    clustCent = []
    #track if a points been seen already
    beenVisitedFlag = zeros( numPts, dtype = uint8 )
    #number of points to possibly use as initialization points
```

```

numInitPts = numPts
#used to resolve conflicts on cluster membership
clusterVotes = []

while numInitPts:

    rand = random.rand()
    #pick a random seed point
    tempInd = int(floor( (numInitPts-1e-6)*rand ))
    #use this point as start of mean
    stInd = initPtInds[ tempInd ]
    # intilize mean to this points location
    myMean = dataPts[ :, stInd ]
    # points that will get added to this cluster
    myMembers = []
    #used to resolve conflicts on cluster membership
    thisClusterVotes = zeros( numPts, dtype = uint16 )

    while True:
        #dist squared from mean to all points still active
        sqDistToAll = (( myMean[:,newaxis] - dataPts )**2).sum(0)
        #points within bandwidth
        inInds = where(sqDistToAll < bandSq)
        #add a vote for all the in points belonging to this cluster
        thisClusterVotes[ inInds ] = thisClusterVotes[ inInds ]+1

        #save the old mean
        myOldMean = myMean
        #compute the new mean
        myMean = mean( dataPts[ :, inInds[0] ], 1 )
        #add any point within bandwidth to the cluster
        myMembers.extend( inInds[0] )
        #mark that these points have been visited
        beenVisitedFlag[myMembers] = 1

    if la.norm(myMean-myOldMean) < stopThresh:
        #check for merge possibilities
        mergeWith = None
        for cN in xrange( numClust ):
            #distance from possible new clust max to old clust max
            distToOther = la.norm( myMean - clustCent[ cN ] )
            #if its within bandwidth/2 merge new and old
            if distToOther < bandwidth/2:
                mergeWith = cN
                break

        # something to merge
        if mergeWith is not None:
            #record the max as the mean of the two merged (I know biased towards
            new ones)
            clustCent[ mergeWith ] = 0.5*( myMean + clustCent[ mergeWith ] )

```

```

        #add these votes to the merged cluster
        clusterVotes[ mergeWith ] += thisClusterVotes
    else:
        #increment clusters
        numClust = numClust+1
        #record the mean
        clustCent.append( myMean )
        clusterVotes.append( thisClusterVotes )

    break

    initPtInds = where(beenVisitedFlag == 0)[0]
    numInitPts = len(initPtInds)

    data2cluster = asarray( clusterVotes ).argmax(0)

    return clustCent, data2cluster

```

```

dataPts = asarray([[1,1],[2,2],[3,3],[9,9],[9,9],[9,9],[10,10]]).T
print dataPts
print dataPts.shape
bandwidth = 2
print 'data points:', dataPts
print 'bandwidth:', bandwidth
clustCent, data2cluster = mean_shift(dataPts, 2)
print 'cluster centers:', sorted( asarray( clustCent ).squeeze().tolist() )
print 'data2cluster:', data2cluster
print len( clustCent )
print sorted( asarray( clustCent ).squeeze().tolist() )

```

```

[[ 1  2  3  9  9  9 10]
 [ 1  2  3  9  9  9 10]]
(2, 7)
data points: [[ 1  2  3  9  9  9 10]
 [ 1  2  3  9  9  9 10]]
bandwidth: 2
cluster centers: [[2.0, 2.0], [9.25, 9.25]]
data2cluster: [1 1 1 0 0 0 0]
2
[[2.0, 2.0], [9.25, 9.25]]

```

```

print asarray(data.T)[:30]
clustCent, data2cluster = mean_shift(asarray(data.T), 5000)

```

```

[[54620 52694 53253 ..., 8828 8879 10002]
 [43523 42750 43024 ..., 59102 59244 61399]]

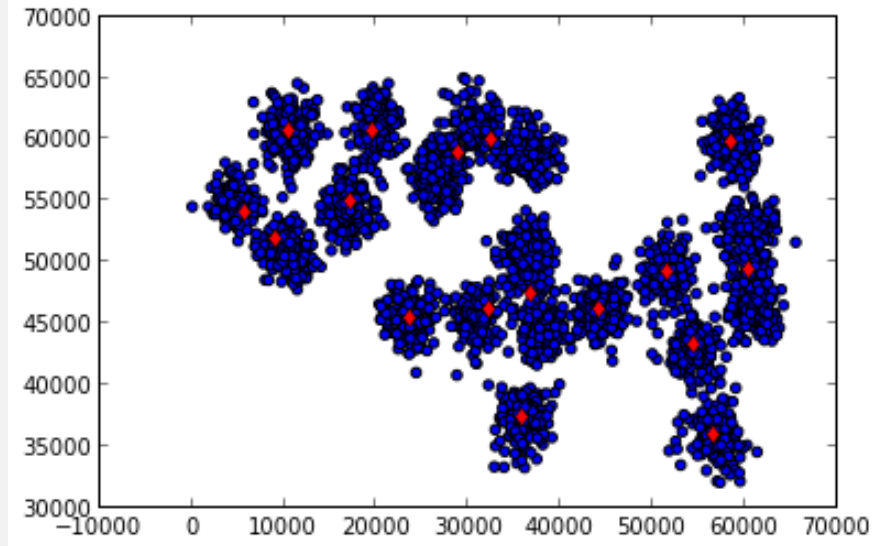
```



```
print asarray(clustCent)[:6]
print asarray(clustCent).shape
```

```
[[ 23699.95483871  45459.8      ]
 [ 51577.72222222  49152.9691358 ]
 [ 36875.68540055  47307.43779413]
 [ 10569.30463576  60602.27152318]
 [ 19721.95906433  60651.1871345 ]
 [ 32502.45283019  59820.74339623]]
(17, 2)
```

```
scatter(data[:,0],data[:,1])
plt.hold(True)
for x in asarray(clustCent): plot(x[0],x[1], 'rd')
```



Teorik Konular

Bu metota teorik bir yapıya oturtmak için onu yazının ilk basındaki resimde olduğu gibi görmek gerekiyor, yani mesela o ilk resmin sağındaki 2 boyuttaki veri dağılımı (ki ayrışal, sayısal), 3 boyuttaki sürekli (continuous) bir baska dağılımın yansıması sanki, ki o zaman 2 boyuttaki yoğunluk bölgeleri sürekli dağılımdaki tepe noktalarını temsil ediyorlar, ve biz o sürekli versiyondaki tepe noktalarını bulmalıyız. Fakat kümeleme işleminin elinde sadece 2 boyuttaki veriler var, o zaman sürekli dağılımı bir şekilde yaratmak lazım.

Bunu yapmak için problem / veri önce bir Çekirdek Yoğunluk Kestirimi (Kernel Density Estimation -KDE-) problemi gibi görülüyor, ki her nokta üzerine bir çekirdek fonksiyonu koyularak ve onların toplamı alınarak sayısal dağılım pürüzsüz bir hale getiriliyor. Ortalama Kaydırma için gerekli kayma “yonu” ise iste bu yeni sürekli fonksiyonun gradyanıdır deniyor, ve gradyan yerel tepe noktasını gösterdiği için o yöne yapılan hareket bizi yavaş yavaş tepeye götürecektir. Bu hareketin yerel

tepeleri bulacagi, ve tum yontemin nihai olarak sonuca yaklasacagi (convergence) matematiksel olarak ispat edilebilir.

Tum dagilim fonksiyonunun icbukey olup olmadigi onemli degil (ki mesela lojistik regresyonda bu onemliydi), cunku nihai tepe noktasini degil, birkac yerel tepe noktasindan birini (hatta hepsini) bulmakla ilgileniyoruz. Gradyan bizi bu noktaya tasiyacaktir.

Kaynaklar

<http://www.serc.iisc.ernet.in/~venky/SE263/slides/Mean-Shift-Theory.pdf>

<http://saravananthirumuruganathan.wordpress.com/2010/04/01/introduction-to-mean-shift-algorithm/>

http://www.cse.yorku.ca/~kosta/CompVis_Notes/mean_shift.pdf

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TUZEL1/MeanShift.pdf

<http://yotamgingold.com/code/MeanShiftCluster.py>