

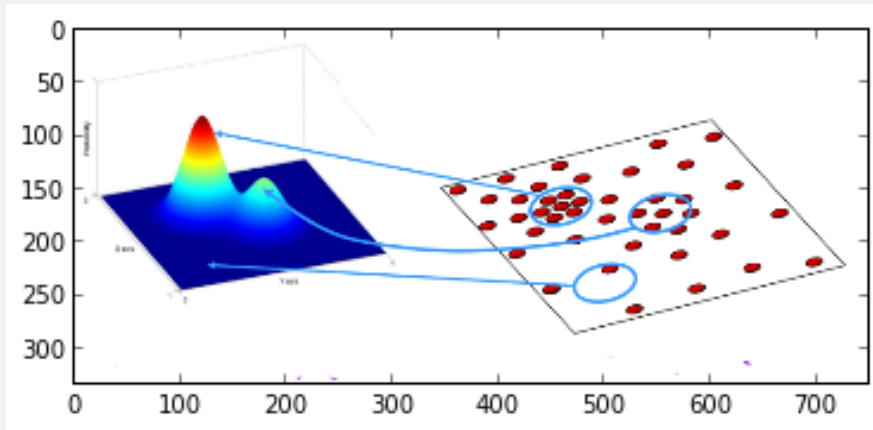
Ortalama Kaydırma ile Kumeleme (Mean Shift Clustering)

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>



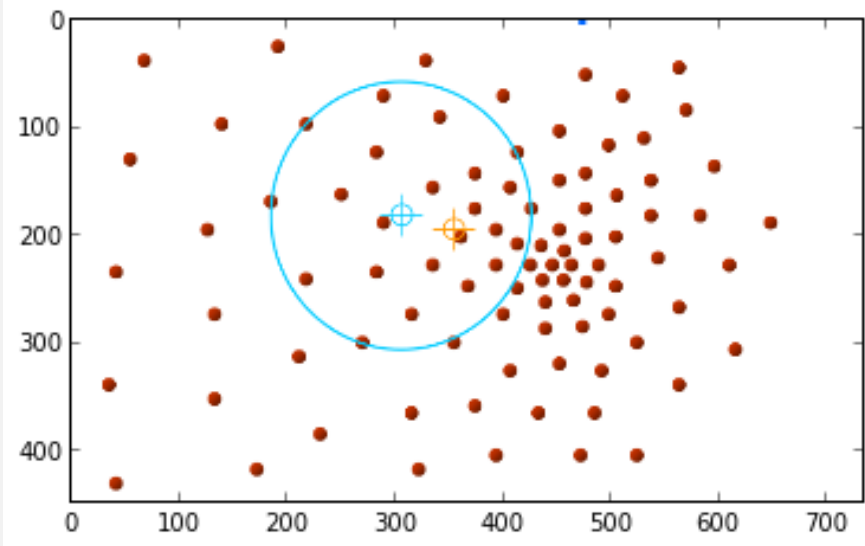
Baslangic neresidir? Baslangic tüm noktaldır, yani her noktadan baslanarak

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 ortalamaya 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.

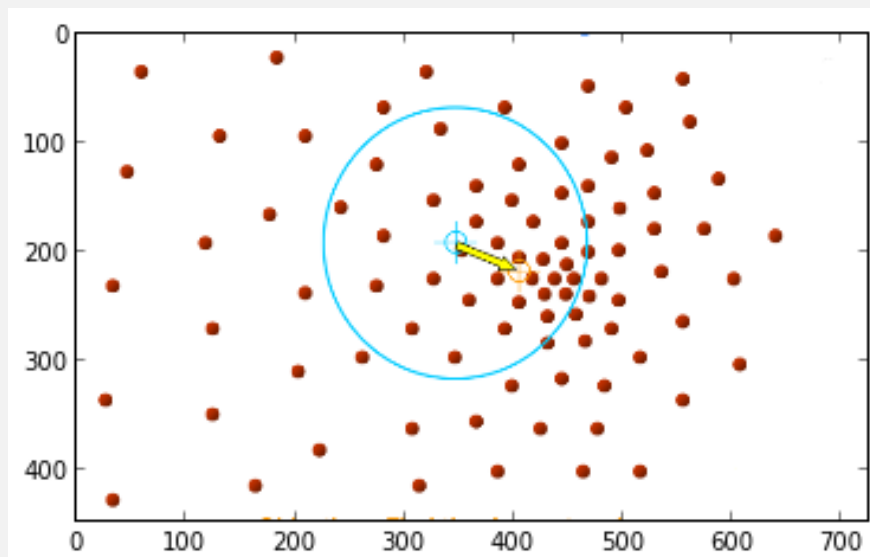
```
im=imread("mean_2.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9b966ac>



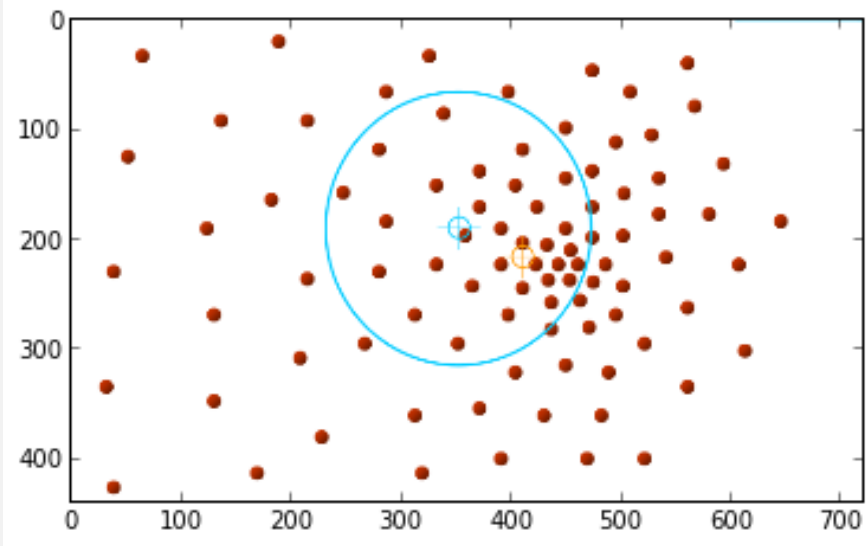
```
im=imread("mean_3.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9cd99ec>



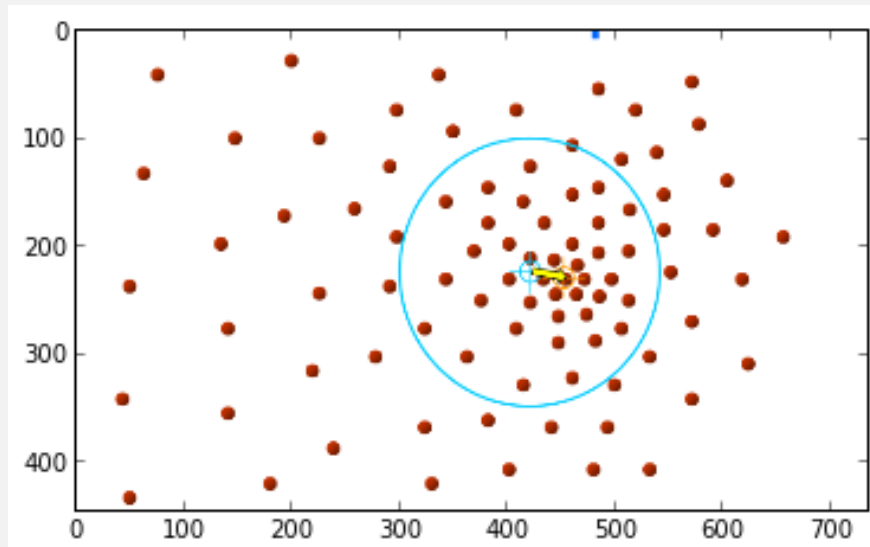
```
im=imread("mean_4.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9e3cfac>



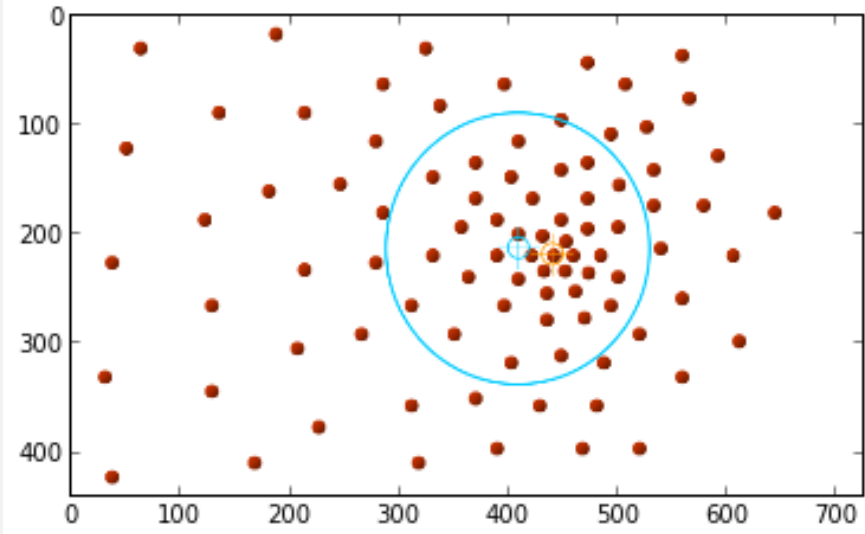
```
im=imread("mean_5.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0x9f9b5ec>



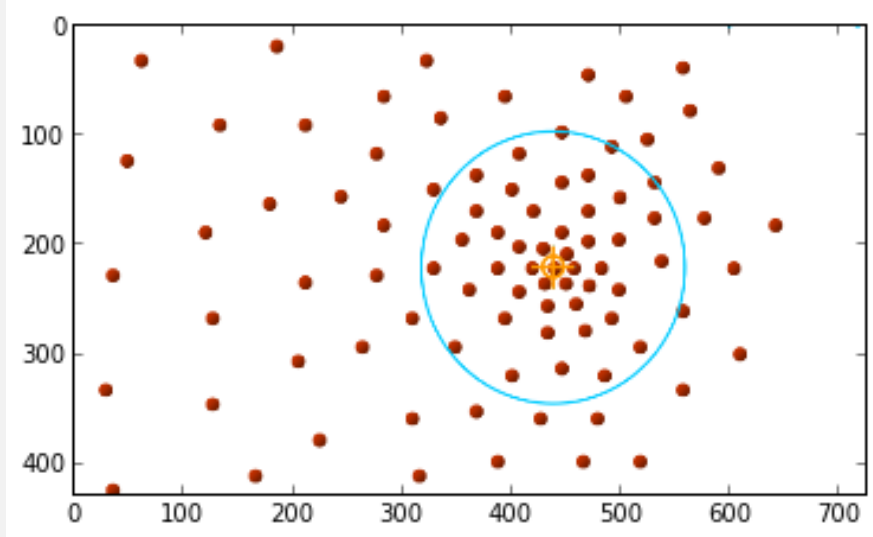
```
im=imread("mean_6.png"); imshow(im)
```

<matplotlib.image.AxesImage at 0xa13cd0c>



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

<matplotlib.image.AxesImage at 0xa2a132c>



Eger yogunluk merkezine çok yakın bir noktadan / noktalardan baslamissak ne olur?

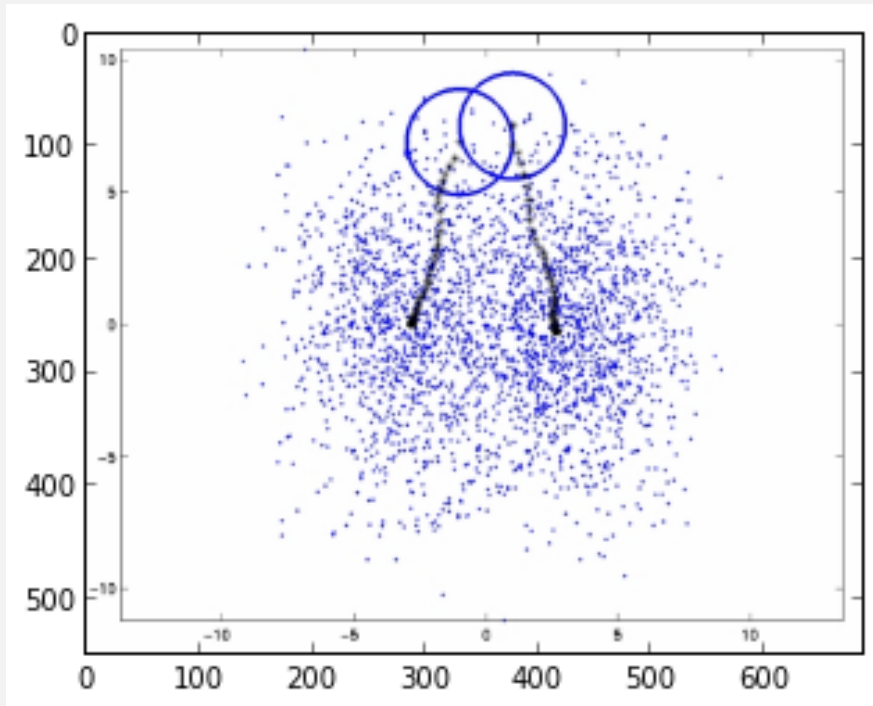
O zaman ilerleme o baslangic noktasi için anında bitecek, çünkü hemen yogunluk merkezine gelmiş olacağız. Diğer yonlerden gelen pencereler de aynı yere gelecekler tabii, o zaman aynı / yakın yogunluk merkezlerini aynı kume olarak kabul etmemiz gerekir. Bu “aynı kume irdelemesi” sayısal hesaplama açısından ufak farklar gösterebilir tabii, ve bu ufak farkı gozonune alarak “kume birlestirme” mantigini da eklemek gerekiyor.

Ortalama Kaydırma sisteminde pencere büyüklüğü kullanıcı tarafından tanımlanır. Optimal

pencere buyuklugunu nasil buluruz? Deneme yanilma yontemi, verinin tarifsel istatistiklerine ke-
stirme bir hesap (estimate) etmek, ya da kullanicinin ayni istatistiklere bakarak tahminde bu-
lunmasi. Birkac farkli pencere buyuklugu de denenebilir. Bu konu literaturde (Ing. bandwidth
selection) adi atlinda uzun uzadiya tartisilmaktadir. Fakat evet, kullanıcı tarafından tanimli bu
parametrenin bir anlamda bu metotun bir zayifligi oldugu soylenebilir. KMeans kume sayisini istiy-
ordu, bu metot ta pencere buyuklugunu istiyor. Hangi metotun ne zaman uygun oldugunu anlamak
tecrube gerektiriyor.

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

```
<matplotlib.image.AxesImage at 0x9af0f2c>
```



Altta ornek veri ve kodu bulabilirsiniz (kod scikit-learn adli kutuphaneden alinmistir). Metot kume
sayisi 17'yi otomatik olarak buluyor.

Alternatif bir kod meanshift_alternative.py dosyasinda bulunabilir, bu kod pencereler kaydirirken
onlari uzerinden gectigi noktalar “sahiplenen” turden bir kod. Yani [encere hareketini durdur-
dugunda hem kume merkezini hem de o kumenin altindaki noktalar bulmus oluyoruz. Tabii son-
raki pencereler bazi noktalar onceki kumelerden calabilirler. Neyse, islemin normal isleyisine gore
bir sonraki pencere secilecektir ve bu pencere “geriye kalan noktalar” uzerinden islem yapacaktır.
Beklenir ki, islem ilerledikce islenmesi gereken noktalar azalacaktir ve yontemin bu sebeple klasik
yonteme gore daha hizli isleyecegi tahmin edilebilir. Hakikaten de boyledir.

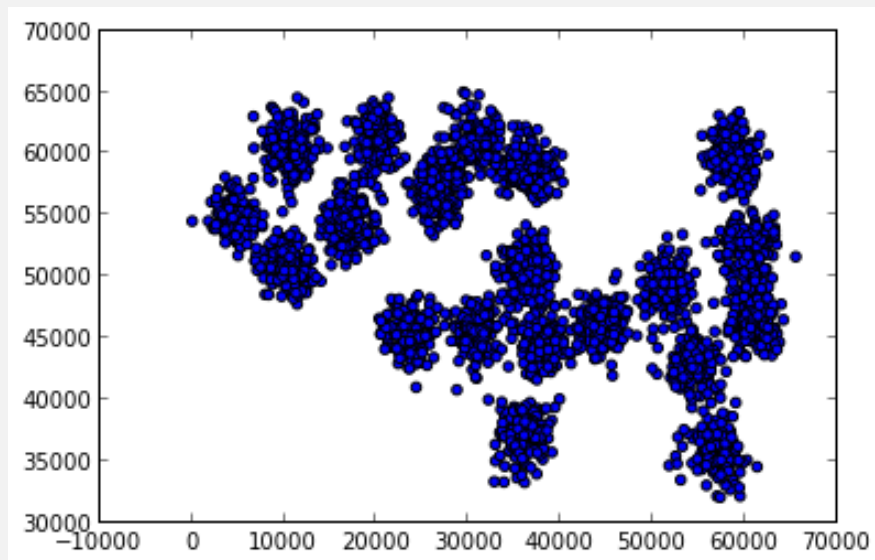
```
from pandas import *  
data = read_csv("synthetic.txt",header=None,sep=" ")
```

```
print data.shape
data = np.array(data)
```

(3000, 2)

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

<matplotlib.collections.PathCollection at 0xa0b7e0c>



```
import numpy as np
from sklearn.neighbors import NearestNeighbors
from sklearn.utils import extmath

def mean_shift(X, bandwidth=None, max_iterations=300):

    seeds = X
    n_samples, n_features = X.shape
    stop_thresh = 1e-3 * bandwidth # when mean has converged
    center_intensity_dict = {}
    nbrs = NearestNeighbors(radius=bandwidth).fit(X)

    # For each seed, climb gradient until convergence or max_iterations
    for my_mean in seeds:
        completed_iterations = 0
        while True:
            # Find mean of points within bandwidth
            i_nbrs = nbrs.radius_neighbors([my_mean], bandwidth,
                                           return_distance=False)[0]
            points_within = X[i_nbrs]
            if len(points_within) == 0:
```

```

        break # Depending on seeding strategy this condition may occur
    my_old_mean = my_mean # save the old mean
    my_mean = np.mean(points_within, axis=0)
    # If converged or at max_iterations, addS the cluster
    if (extmath.norm(my_mean - my_old_mean) < stop_thresh or
        completed_iterations == max_iterations):
        center_intensity_dict[tuple(my_mean)] = len(points_within)
        break
    completed_iterations += 1

# POST PROCESSING: remove near duplicate points
# If the distance between two kernels is less than the bandwidth,
# then we have to remove one because it is a duplicate. Remove the
# one with fewer points.
sorted_by_intensity = sorted(center_intensity_dict.items(),
                             key=lambda tup: tup[1], reverse=True)
sorted_centers = np.array([tup[0] for tup in sorted_by_intensity])
unique = np.ones(len(sorted_centers), dtype=np.bool)
nbrs = NearestNeighbors(radius=bandwidth).fit(sorted_centers)
for i, center in enumerate(sorted_centers):
    if unique[i]:
        neighbor_idx = nbrs.radius_neighbors([center],
                                              return_distance=False)[0]
        unique[neighbor_idx] = 0
        unique[i] = 1 # leave the current point as unique
cluster_centers = sorted_centers[unique]

# ASSIGN LABELS: a point belongs to the cluster that it is closest to
nbrs = NearestNeighbors(n_neighbors=1).fit(cluster_centers)
labels = np.zeros(n_samples, dtype=np.int)
distances, idxs = nbrs.kneighbors(X)
labels = idxs.flatten()

return cluster_centers, labels

```

```
cluster_centers, labels = mean_shift(np.array(data), 4000)
```

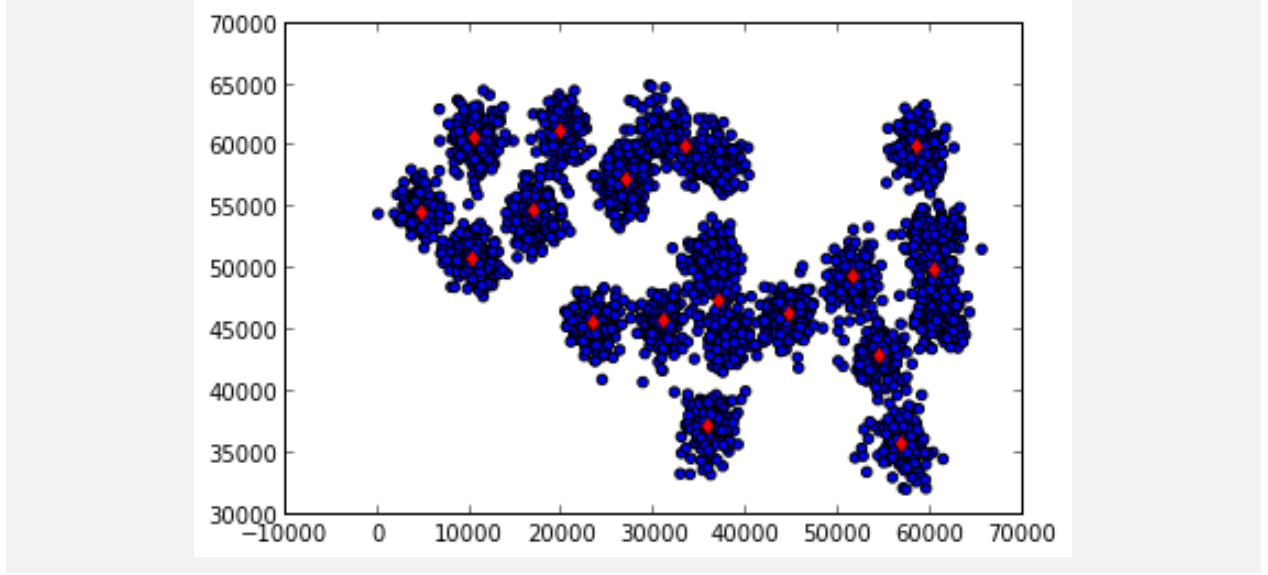
```
print len(cluster_centers)
```

```
17
```

```

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

```



Teorik Konular

Bu metodu 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 başka 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 (elimizde bir sürekli fonksiyon olduğu için türev rahatlıkla alabiliyoruz), ve gradyan yerel tepe noktasını gösterdiği için o yöne yapılan hareket bizi yavaş yavaş tepeye götürecektir. Bu hareketin yerel tepeleri bulacağı, ve tüm yöntemin nihai olarak sonuca yaklaşıncaya (convergence) matematiksel olarak ispat edilebilir.

KDE ile elde edilen teorik dağılım fonksiyonunun icbukey olup olmadığı önemli değil (ki mesela lojistik regresyonda bu önemliydi), çünkü nihai tepe noktasını değil, birkaç yerel tepe noktasından birini (hatta hepsini) bulmakla ilgileniyoruz. Gradyan bizi bu noktaya taşıyacaktır.

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

Scikit-Learn Kodlari

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