

K-Means Kumeleme Metodu

Yapay Ogrenim (Machine Learning) alanında önemli algoritmalarından biri k-means metodu. K-means kumelemesi için kaç tane klemenin olması gerektiği baştan tanımlanır (yani k parametresi), algoritma bunu kendisi bulmaz.

Metodun geri kalanı basittir - bir döngü (iteration) içinde her basamakta:

- 1) Her nokta için, eldeki kume merkezleri teker teker kontrol edilir ve o nokta en yakın olan kumeye atanır
- 2) Atamalar tamamlandıktan sonra her kume içinde hangi noktaların olduğu bilindiği için her kumedeki noktaların ortalaması alınarak yeni kume merkezi hesaplanır. Eski merkez hesapları atılır.
- 3) Basa donulur

Döngü tekrar ilk adıma döndüğünde, bu sefer yeni kume merkezlerini kullanarak, aynı adımlar tekrar yapılacaktır.

Fakat bir problem yok mu? Daha birinci döngü başlamadan kume merkezlerinin nerede olduğunu nereden bileceğiz? Burada bir tavuk-yumurta problemi var, kume merkezleri olmadan noktaları atayamayız, atama olmadan kume merkezlerini hesaplayamayız.

Bu probleme pratik bir çözüm ilk başta kume merkezlerini tahmin etmektir. Yani merkezleri rasgele bir şekilde hesaplamak. Pratikte bu yöntem çok iyi işliyor. Tabii bu rasgelelik yüzünden K-means'ın doğru sonuca yaklaşıksallaşı (convergence) garantisi değildir, ama gerçek dünya uygulamalarında çoğunlukla kullanışlı kümeler bulunur. Bu potansiyel problemlerden kaçınmak için k-means pek çok kez işletilebilir (her seferinde yeni rasgele başlangıçlarla yani) ve aynı sonuca ulaşıp ulaşılmadığı kontrol edilebilir.

Pek en iyi k nasıl bulunur? Burada da yapay öğrenim literatüründe pek çok yaklaşım vardır [1], veriyi pek çok parçaya bölüp, farklı k kume sayısı için kumeleme yapmak ve çapraz sağlama (cross-validation) kullanmak, vs.

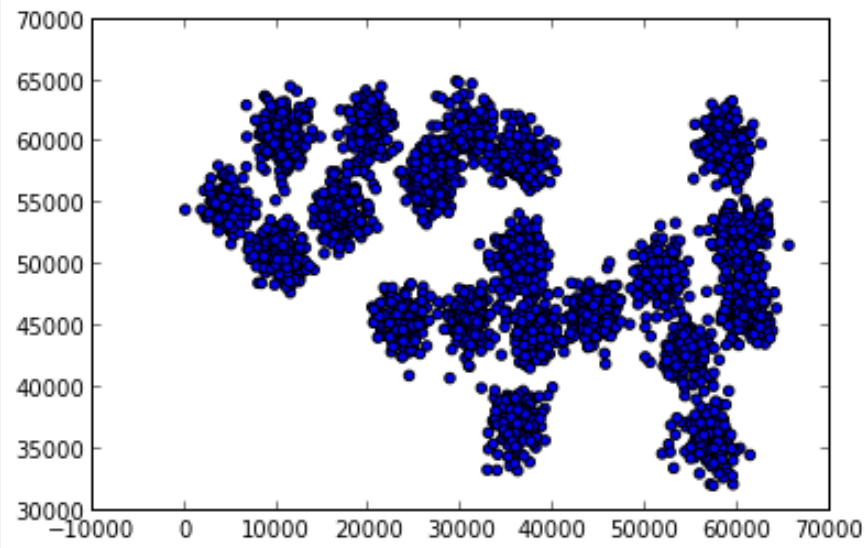
Örnek test verisi aşağıda

```
from pandas import *  
data = read_csv("synthetic.txt",names=['a','b'],sep=" ")  
print data.shape
```

```
(3000, 2)
```

```
data = np.array(data)  
scatter(data[:,0],data[:,1])
```

```
<matplotlib.collections.PathCollection at 0xa371b8c>
```



```
def find_nearest(X,mu):
    nexamples = X.shape[0]
    k = mu.shape[0]

    Xmu = dot(X,mu.T)
    xx = sum(X*X,axis=1)
    mm = sum(mu*mu, axis=1)

    cl = empty(nexamples,dtype=int)
    dists = inf * ones(nexamples)
    for i in range(k):
        new_dist = - 2*Xmu[:,i] + mm[i]
        change = new_dist < dists
        if sum(change)>0:
            cl[change] = [i for j in range(sum(change))]
            dists[change] = new_dist[change]
    return cl

def kmeans(X, k=5, niter=100):
    nexamples,nfeatures = X.shape

    # initialize with a random clustering
    random.seed(1) # seed random number generator for reproducibility
    cl = random_integers(0,k,nexamples)
    cl_old = cl

    # allocate means
    mu = np.empty((k,nfeatures))

    it = 0
    while True:
        # compute means
```

```

for i in range(k):
    mu[i,:] = np.mean(X[cl==i,:],axis=0)

# assign examples to clusters
cl = find_nearest(X, mu)

# compute difference in assignments
nchanges = sum(cl != cl_old)
print 'it: %d, num. changes: %d' % (it, nchanges)
if nchanges == 0 or it == niter:
    break

cl_old = cl
it += 1
return cl, mu

```

```
c = kmeans(data)
```

```

it: 0, num. changes: 2490
it: 1, num. changes: 561
it: 2, num. changes: 277
it: 3, num. changes: 142
it: 4, num. changes: 50
it: 5, num. changes: 32
it: 6, num. changes: 35
it: 7, num. changes: 33
it: 8, num. changes: 24
it: 9, num. changes: 13
it: 10, num. changes: 6
it: 11, num. changes: 2

it: 12, num. changes: 0

```

```
c
```

```

(array([4, 4, 4, ..., 1, 1, 1]),
 array([[ 31006.2193159 ,  58653.75251509],
        [ 12247.38927098,  56210.74828061],
        [ 38859.46193772,  44285.88408304],
        [ 27194.63636364,  45613.61279461],
        [ 57107.29855716,  47792.74694784]]))

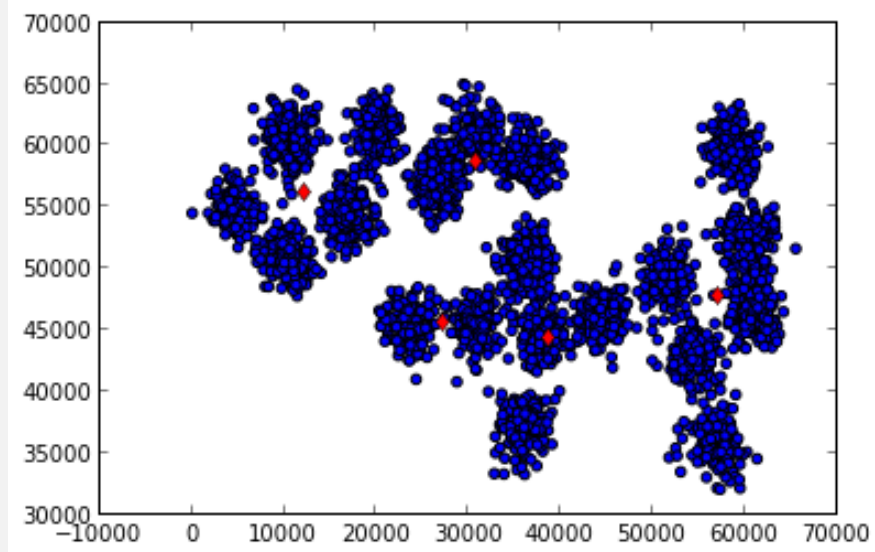
```

Ustteki sonucun icinde iki ana vektor var, bu vektorlerden birincisi icinde 4,1, gibi sayilar goruluyor, bu sayilar her noktaya tekabul eden kume atamaları. Ikinci vektor icinde iki boyutlu k tane vektor var, bu vektorler de her kumenin merkez noktası. Merkez noktalarını ham veri üzerinde grafiklersek (kirmizi noktalar)

```

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

```



Goruldugu gibi 5 tane kume icin ustteki merkezler bulundu. Fena degil. Eger 10 dersekf

```

c = kmeans(data,k=10)
scatter(data[:,0],data[:,1])
plt.hold(True)
for x in c[1]:
    plot(x[0],x[1], 'rd')

```

```

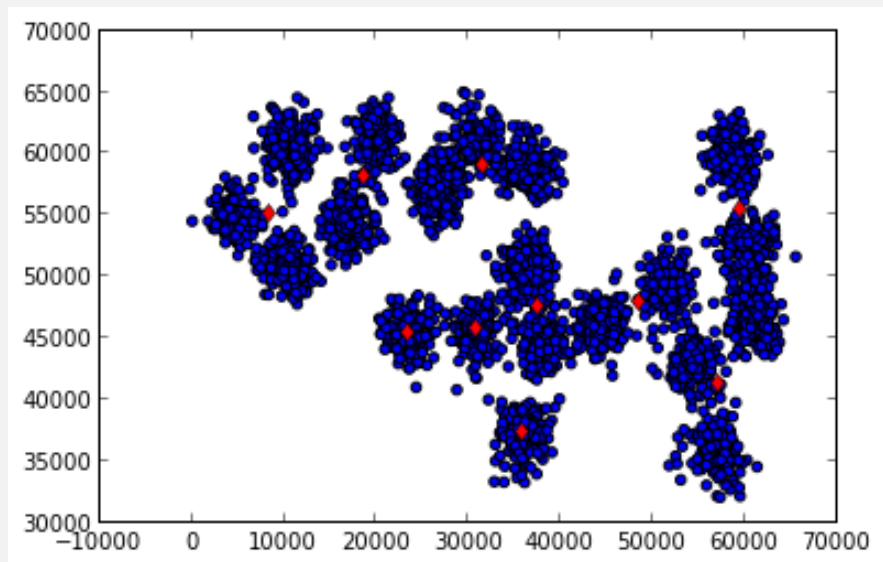
it: 0, num. changes: 2683
it: 1, num. changes: 706
it: 2, num. changes: 648
it: 3, num. changes: 262
it: 4, num. changes: 152
it: 5, num. changes: 133
it: 6, num. changes: 132
it: 7, num. changes: 118
it: 8, num. changes: 90

it: 9, num. changes: 71
it: 10, num. changes: 40
it: 11, num. changes: 20
it: 12, num. changes: 12
it: 13, num. changes: 15
it: 14, num. changes: 11
it: 15, num. changes: 10

```

```
it: 16, num. changes: 9
it: 17, num. changes: 7

it: 18, num. changes: 6
it: 19, num. changes: 4
it: 20, num. changes: 6
it: 21, num. changes: 4
it: 22, num. changes: 0
```



- [1] http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set
- [2] nbviewer.ipython.org/url/cbcb.umd.edu/~hcorrada/PML/src/kmeans.ipynb