

Dagilim Karisimleri ve Takip Edilmeden Kumeleme (Unsupervised Clustering)

Gaussian (normal) dagilimi tek tepesi olan (unimodal) bir dagilimdir. Bu demektir ki eger birden fazla tepe noktası olan bir veriyi modellemek istiyorsak, degisik yaklasimlar kullanmamiz gerekecektir. Birden fazla Gaussian'i "karistirmek (mixing)" bu tur bir yaklasim olabilir. Karistirmek, karisim icindeki her Gaussian'dan gelen sonuclari toplamaktır, yani kelimenin tam anlamıyla her veri noktasini teker teker karisimdaki tum dagilimlara gecip sonuclari toplamaktır. Eger cok boyutlu normal dagilimleri topluyorsak, formül:

$$p(x) = \sum_z \pi_z N(x|\mu_z, \Sigma_z)$$

π_z karistirma oranlaridir (mixing proportions). Iki Gaussian oldugunu dusunelim, π_1, π_2 oranlari 0.2, 0.8 olabilir mesela (toplam her zaman 1 olmalidir), her nokta her Gaussian'a verildikten sonra tekabül eden agirlikla mesela sirayla 0.2, 0.8 ile carpilip toplanir.

Ornek olarak alttaki veriye bakalim.

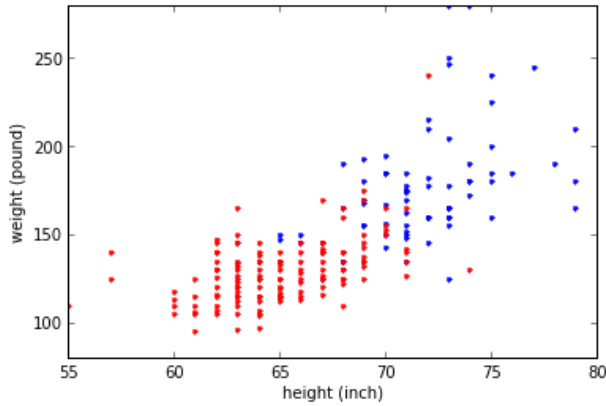
```
def pnorm(x, m, s):
    """
    Compute the multivariate normal distribution with values
    vector x, mean vector m, sigma (variances/covariances) matrix
    s
    """
    xmt = np.matrix(x-m).transpose()
    for i in xrange(len(s)):
        if s[i,i] <= sys.float_info[3]: # min float
            s[i,i] = sys.float_info[3]
    sinv = np.linalg.inv(s)
    xm = np.matrix(x-m)
    return (2.0*math.pi)**(-len(x)/2.0)*\
        (1.0/math.sqrt(np.linalg.det(s)))\
        *math.exp(-0.5*(xm*sinv*xmt))

import scipy.stats
data = np.loadtxt('biometric_data_simple.txt', delimiter=',')

women = data[data[:,0] == 1]
men = data[data[:,0] == 2]

plt.xlim(55,80)
plt.ylim(80,280)
plt.plot (women[:,1],women[:,2], 'b.')
plt.hold(True)
plt.plot (men[:,1],men[:,2], 'r.')
```

```
plt.xlabel('height (inch)')
plt.ylabel('weight (pound)')
plt.savefig('mixbern_1.png')
```



Bu grafik kadınlar ve erkeklerin boy (height) ve kilolarini (weight) iceren bir veri setinden geliyor, veri setinde erkekler ve kadınlara ait olan olcumler onceden isaretlenmis / etiketlenmis (labeled), biz de bu isaretleri kullanarak kadınlari kirmizi erkekleri mavi ile grafikledik. Ama bu isaretler / etiketler verilmiş olsun ya da olmasın, kavramsal olarak düşünürsek eğer bu veriye bir dağılım uydurmak (fit) istersek bir karışım kullanılması gerekli, çünkü iki tepe noktasiyle daha rahat temsil edileceğini düşündüğümüz bir durum var ortada.

```
# Multivariate gaussian, contours
#
import scipy.stats
data = np.loadtxt('biometric_data_simple.txt',delimiter=',')

def norm_pdf(b,mean,cov):
    k = b.shape[0]
    part1 = np.exp(-0.5*k*np.log(2*np.pi))
    part2 = np.power(np.linalg.det(cov),-0.5)
    dev = b-mean
    part3 = np.exp(-0.5*np.dot(np.dot(dev.transpose(),np.linalg.inv(cov)),dev))
    dmwnorm = part1*part2*part3
    return dmwnorm

women = data[data[:,0] == 1]
men = data[data[:,0] == 2]

plt.xlim(55,80)
plt.ylim(80,280)
plt.plot (women[:,1],women[:,2], 'b.')
plt.hold(True)
```

```

plt.plot (men[:,1],men[:,2], 'r.')
plt.xlabel('height (inch)')
plt.ylabel('weight (pound)')
plt.hold(True)

x = np.arange(55., 80., 1)
y = np.arange(80., 280., 1)
X, Y = np.meshgrid(x, y)

Z = np.zeros(X.shape)
nx, ny = X.shape
mu1 = np.array([ 72.89350086, 193.21741426])
sigma1 = np.matrix([[ 7.84711283, 25.03111826],
                    [ 25.03111826, 1339.70289046]])
for i in xrange(nx):
    for j in xrange(ny):
        Z[i,j] = norm_pdf(np.array([X[i,j], Y[i,j]]),mu1,sigma1)

levels = np.linspace(Z.min(), Z.max(), 4)

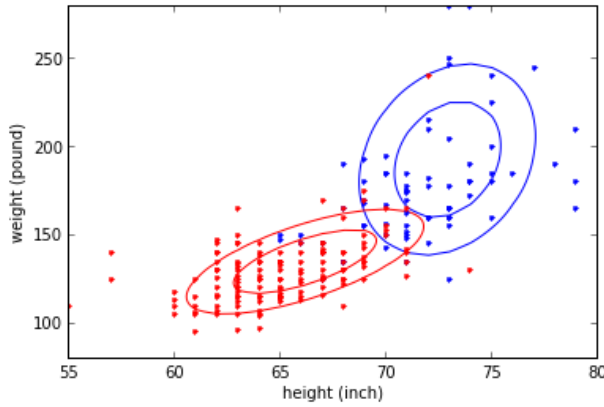
plt.contour(X, Y, Z, colors='b', levels=levels)
plt.hold(True)

Z = np.zeros(X.shape)
nx, ny = X.shape
mu2 = np.array([ 66.15903841, 135.308125 ])
sigma2 = np.matrix([[ 14.28189396, 51.48931033],
                    [ 51.48931033, 403.09566456]])
for i in xrange(nx):
    for j in xrange(ny):
        Z[i,j] = norm_pdf(np.array([X[i,j], Y[i,j]]),mu2,sigma2)

levels = np.linspace(Z.min(), Z.max(), 4)

plt.contour(X, Y, Z, colors='r', levels=levels)
plt.savefig('mixbern_2.png')

```



Bu karisim icindeki Gaussian'leri ustteki gibi cizebilirdik (gerci ustteki aslinda ileride yapacagimiz net bir hesaptan bir geliyor, ona birazdan geliyoruz, ama ciplak gozle de bu sekil uydurulabilirdi). Modeli kontrol edelim, elimizde bir karisim var, nihai olasilik degeri $p(x)$ 'i nasil kullaniriz? Belli bir noktanin olasiligini hesaplamak icin bu noktayi her iki Gaussian'a teker teker geceriz (ornekte iki tane), ve gelen olasilik sonuclarini karisim oranlari ile carparak toplariz. Agirliklar sayesinde iki sey elde ediyoruz 1) karisim entegre edilince hala 1 degeri cikiyor zaten bir dagilimin uy-masi gereken sartlardan biri bu 2) kesisim olan bolgelerde her iki Gaussian buyuk bir deger verebilir, o zaman agirliklar devreye girer, ve nihai olasilik, agirliklara gore carpilip toplanan bir sonuc olur. Bu bolgelerde bir Gaussian'in agirliginin digerinden fazla olmasinin da ozel bir anlami var, demek ki o bolgede agirligi fazla olan Gaussian daha fazla noktaya sahip (verisel olarak), ki o zaman o bolgedeki bir noktanin olasiligi sorulunca, agirligi fazla olan Gaussian daha yuksek bir olasilik degeri geri dondurmeli.

Kesisme olmayan bolgeler zaten pek onemli degil, o noktalarin olasilik degeri zaten agirlikla tek bir Gaussian'dan geliyor olacak, cunku diger Gaussian o bolge icin sifira yakin bir deger verir, ve bu sifira yakin deger toplamda zaten bir fark yaratmayacak.

Etiketler Bilin**mi**yorsa

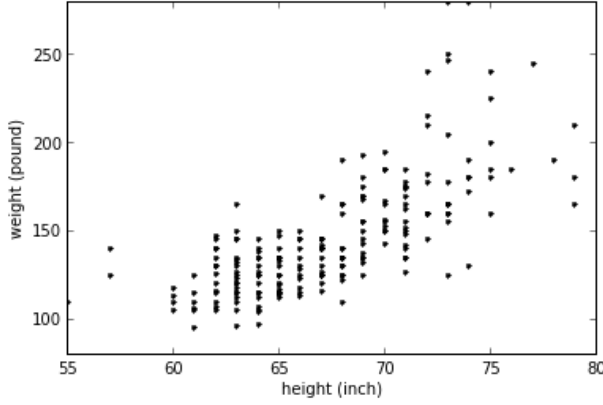
Simdi veriyi modellemenin otesinde, biraz daha analitik, daha makine ogrenimi ile alakali ihtiyaclara geelim. Eger etiketler bize onceden verilmemis olsaydi, hangi veri noktalarinin kadınlara, hangilerinin erkeklere ait oldugunu bilmeseydik o zaman ne yapardik? Bu veriyi grafiklerken etiketleri renkleyemezdik tabii ki, soyle bir resim cizebilirdik ancak,

```
import scipy.stats
data = np.loadtxt('biometric_data_simple.txt', delimiter=',')

women = data[data[:,0] == 1]
men = data[data[:,0] == 2]

plt.xlim(55,80)
plt.ylim(80,280)
```

```
plt.plot (data[:,1],data[:,2], 'k.')
plt.xlabel('height (inch)')
plt.ylabel('weight (pound)')
plt.savefig('mixbern_3.png')
```



Fakat yine de sekil olarak iki kumeyi gorebiliyoruz. Acaba oyle bir makine ogrenimi algoritmasi olsa da, biz bir karisim oldugunu tahmin edip, sonra o karisimi veriye uydururken, etiket degerlerini de kendiliginden tahmin etse? Bu tam bir veri madenciligi denemesi olurdu.

Bu ise baslamadan once etiketler ile karisimlarin arasindaki baglantiyi gorelim. Her nokta icin bilinen / bilinmeyen etiket kavramindan, matematiksel olarak direk karisimlara gecis yapabilmemiz lazim.

Diyelim ki her nokta icin 0/1 degerini tasiyabilecek "gizli" bir z rasgele degiskeni var, o zaman $p(x)$ 'i su sekilde acabiliriz

$$p(x) = \sum_z p(x, z)$$

Bu mantikli degil mi? Ortak dagilim $p(x, z)$ icinden $p(x)$ 'i cekip cikarmak, $p(x, z)$ icin bir bilesen (marginal) hesabi yapmak demektir, o zaman ortak dagilimin icindeki tum z degerlerini toplamak gerekir. Devam edelim, Bayes Teorisi'ni kullanarak

$$= \sum_z p(x, z) = \sum_z p(z)p(x|z)$$

elde ederiz. Burada $p(z)$, yani z 'nin 0/1 degerine "sahip olup olmadiginin olasiligi" bizi π_z 'ye goturur, yani

$$\sum_z p(z)p(x|z) = \sum_z \pi_z N_z(x|\mu_z, \sigma_z)$$

Unutmayalım, z bir rasgele degisken, ve sahip oldugu olasiliga gore, her veri noktası için, 0 ya da 1 uretiyor. $p(z)$ dedigimiz zaman z tek basına, baska hiçbir parametre ona gecilmiyor, o zaman zaten tanım itibariyle "ta en bastan belirli" bir olasilikten baska bir seye sahip olamaz, bu da karisim orani π_z 'den baskası degildir.

Notasyon

Simdi notasyonu biraz daha berraklastiralim. Oncelikle, ozellikle Bayes modelleri iceren formulasyonlarda, $p(x)$, $p(z)$ gibi kullanımlar gorulur, fakat aslında orada iki tane farklı yogunluk fonksiyonu (density function) kastedilir, $p_x(x)$ ve $p_z(z)$. Surekli p kullanılan turden kullanımın biraz ustunkoru (sloppy) olduğu dogrudur, kimisi için bu daha kısa yoldan formulasyondur, literaturu takip eden herkes bunun nereden geldigini bilir, sadece konuya ilk baslayanlar için biraz kafa karistirici olabiliyor.

Ayrıca $p(z)$ derken $p(z = k)$ demek istiyoruz, yani

$$p(x) = \sum_{k=1}^K p(z = k)p(x|z = k)$$

ki K karisimdaki Gaussian sayisidir. Aynen ustte olduğu gibi etiketin bilindigi, "verili" olduğu durumda kosullu olasilik $p(x|z = k)$, karisimdaki Gaussian'lardan bir tanesidir, ki o da ustte $N_z(x|\mu_z, \sigma_z)$ olarak gosterilmisti, simdi k kullanırsak $N(x|\mu_k, \sigma_k)$ olacaktır.

İki Gaussian olduğu durumda z 'nin 0/1 degerine sahip olup olmadigından bahsettik, ya da K ikiden daha büyük olduğu durumlarda, $z = k$ olup olmama durumu. Aslında bir temsili yontem daha var, z rasgele degiskenini sadece bir hucresinde 1 ya da 0 tasiyan bir katlı terimli (multinomial) dagilim, yani bir vektor olarak gostermek. Yani $z = [0 \ 0 \ 1 \ .. \ 0]^T$ seklinde. Bu temsili yonteme K-icinde-1 (1-of-K) temsili yontemi deniyor. O zaman

$$p(z) = \prod_{k=1}^K \pi_k^{z_k} \quad (1)$$

ve

$$p(x|z) = \prod_{k=1}^K N(x|\mu_k, \Sigma_k)^{z_k} \quad (2)$$

Peki verinin log olabilirliği (log likelihood) nedir?

Bilindigi gibi olabilirlik hesap veri noktalarının teker teker yogunluk fonksiyonuna gecilmesi, ve sonucların birbiri ile carpilmasıdır, log olabilirlik ise onun log alınmış halidir (cunku log alınıncı carpımlar toplam haline donusur, böylece mutlak (absolute) olarak gittikçe buyuyen bir sayı ile işlem yapılabilir, oteki türlü olasılık

degeri oldugu icin 1'den kucuk sayilarin surekli birbiri ile carpimi, nihai carpimi asiri kucultur, bu da bilgisayarın numerik hesap sinirlarini zorlayabilir.

X 'i tum x 'leri iceren bir matrix olarak kabul edelim

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k p(x_n|\mu_k, \Sigma_k) \right\}$$

Genellikle olabilirlik fonksiyonu maksimize edilerek icindeki parametrelerin bu maksimum noktada tasidigi degerler bulunmaya ugrasilir. Fakat bizim esas ilgilendigimiz "bilinmeyen" etiketler, o yuzden maksimizasyon yapmadan once bu etiketleri de bir sekilde olabilirigin icine dahil etmemiz lazim. (1) ve (2)'yi kullanirsak,

$$p(X, Z|\mu, \Sigma, \pi) = \prod_{n=1}^N \prod_{k=1}^K \pi_k^{z_{nk}} N(x_n|\mu_k, \Sigma_k)$$

Bunun log'unu alirsak

$$\ln p(X, Z|\mu, \Sigma, \pi) = \ln \sum_{n=1}^N \sum_{k=1}^K z_{nk} \{ \ln \pi_k + \ln N(x_n|\mu_k, \Sigma_k) \}$$

EM (Expectation Maximization) metodu, bu olurluk fonksiyonunu baz alarak, μ, Σ, π bilindigi durumda etiketleri, etiketler bilindigi durumda μ, Σ, π degerlerini tahmin eder, bu iki bilinmeyen grup arasinda ozyineli (iteratif) olarak gidip gelir. Tabii konunun cok fazla detayi var [1], oncelikle EM'in ustteki durumda yakinsak (convergence) bir davranis sergiledigi bilindir, yani bir optimum vardir, va belli uc nokta sartlari haricinde bu degere yaklasilmasi garantidir, vs.

```
import math, random, copy, sys

def expectation_maximization(t, nbclusters=2, nbiter=3, \
                             normalize=False, epsilon=0.001, \
                             monotony=False, datasetinit=True):
    def draw_params():
        if datasetinit:
            tmpmu = np.array([1.0*t[random.uniform(0,nbobs),:], \
                               np.float64])
        else:
            tmpmu = np.array([random.uniform(min_max[f][0], \
                                              min_max[f][1]) \
                               for f in xrange(nbfeatures)], np.float64)
    return {'mu': tmpmu, \
            'sigma': np.matrix(np.diag(\
                                [(min_max[f][1]-min_max[f][0])/2.0 \
```

```

        for f in xrange(nbfeatures)])),\
        'proba': 1.0/nbclusters}

nbobs = t.shape[0]
nbfeatures = t.shape[1]
min_max = []
# find xranges for each features
for f in xrange(nbfeatures):
    min_max.append((t[:,f].min(), t[:,f].max()))

### Normalization
if normalize:
    for f in xrange(nbfeatures):
        t[:,f] -= min_max[f][0]
        t[:,f] /= (min_max[f][1]-min_max[f][0])
min_max = []
for f in xrange(nbfeatures):
    min_max.append((t[:,f].min(), t[:,f].max()))
### /Normalization

result = {}
quality = 0.0 # sum of the means of the distances to centroids
random.seed()
Pclust = np.ndarray([nbobs,nbclusters], np.float64) # P(clust/obs)
Px = np.ndarray([nbobs,nbclusters], np.float64) # P(obs/clust)
# iterate nbiter times searching for the best "quality" clustering
for iteration in xrange(nbiter):
    # Step 1: draw nbclusters sets of parameters #
    params = [draw_params() for c in xrange(nbclusters)]
    old_log_estimate = sys.maxint # init, not true/real
    log_estimate = sys.maxint/2 + epsilon # init, not true/real
    estimation_round = 0
    # Iterate until convergence (EM is monotone) <=>
    # < epsilon variation
    while (abs(log_estimate - old_log_estimate) > epsilon\
           and (not monotony or log_estimate < old_log_estimate)):
        restart = False
        old_log_estimate = log_estimate
        # Step 2: compute P(Cluster/obs) for each observations #
        for o in xrange(nbobs):
            for c in xrange(nbclusters):
                # Px[o,c] = P(x/c)
                Px[o,c] = pnorm(t[o,:],\
                                params[c]['mu'], params[c]['sigma'])
        #for o in xrange(nbobs):
        # Px[o,:] /= math.fsum(Px[o,:])

```



```

for o in xrange(nbobs):
    for c in xrange(nbclusters):
        # Pclust[o,c] = P(c|x)
        Pclust[o,c] = Px[o,c]*params[c]['proba']
#    assert math.fsum(Px[o,:]) >= 0.99 and\
#           math.fsum(Px[o,:]) <= 1.01
for o in xrange(nbobs):
    tmpSum = 0.0
    for c in xrange(nbclusters):
        tmpSum += params[c]['proba']*Px[o,c]
    Pclust[o,:] /= tmpSum
#assert math.fsum(Pclust[:,c]) >= 0.99 and\
#       math.fsum(Pclust[:,c]) <= 1.01
# Step 3: update the parameters (sets {mu, sigma, proba}) #
#print "iter:", iteration, " estimation#:", estimation_round,\
#      " params:", params
for c in xrange(nbclusters):
    tmpSum = math.fsum(Pclust[:,c])
    params[c]['proba'] = tmpSum/nbobs
    # restart if all converges to one cluster
    if params[c]['proba'] <= 1.0/nbobs:
        restart = True
        #print "Restarting, p:",params[c]['proba']
        break
m = np.zeros(nbfeatures, np.float64)
for o in xrange(nbobs):
    m += t[o,:]*Pclust[o,c]
params[c]['mu'] = m/tmpSum
s = np.matrix(np.diag(np.zeros(nbfeatures, np.float64)))
for o in xrange(nbobs):
    s += Pclust[o,c]*\
        (np.matrix(t[o,:]-params[c]['mu']).transpose()*\
         np.matrix(t[o,:]-params[c]['mu']))
params[c]['sigma'] = s/tmpSum
#print "-----"
#print params[c]['sigma']

### Test bound conditions and restart consequently if needed
if not restart:
    restart = True
    for c in xrange(1,nbclusters):
        if not np.allclose(params[c]['mu'],
                           params[c-1]['mu'])\
        or not np.allclose(params[c]['sigma'],
                           params[c-1]['sigma']):
            restart = False

```

```

        break
    if restart:      # restart if all converges to only
        old_log_estimate = sys.maxint      # init, not true/real
        log_estimate = sys.maxint/2 + epsilon # init, not true/real
        params = [draw_params() for c in xrange(nbclusters)]
        continue
    ### /Test bound conditions and restart

    # Step 4: compute the log estimate #
    log_estimate = math.fsum([math.log(math.fsum(\
        [Px[o,c]*params[c]['proba'] \
        for c in xrange(nbclusters)]))\
        for o in xrange(nbobs)])
    #print "(EM) old and new log estimate: ",\
    #      old_log_estimate, log_estimate
    estimation_round += 1

    # Pick/save the best clustering as the final result
    quality = -log_estimate
    if not quality in result or quality > result['quality']:
        result['quality'] = quality
        result['params'] = copy.deepcopy(params)
        result['clusters'] = [[o for o in xrange(nbobs)\
            if Px[o,c] == max(Px[o,:])]\
            for c in xrange(nbclusters)]
    return result

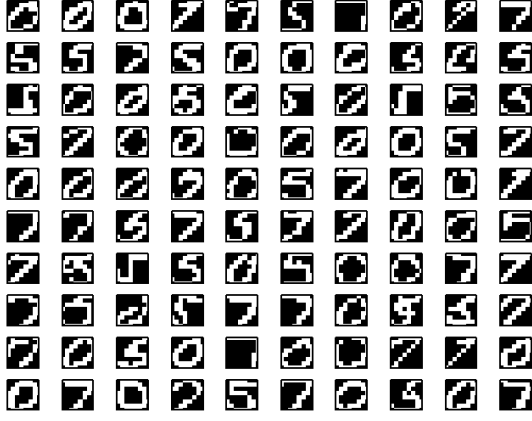
data = np.loadtxt('biometric_data_simple.txt',delimiter=',')
data = data[:,1:3]
res = expectation_maximization(data)
print res

{'params': [{'mu': array([ 66.12666308, 135.12466099]), 'sigma': matrix([[ 14.13
[ 50.64636849, 397.62384374]]), 'proba': 0.8166232863917395}, {'mu': arra
[ 25.75436975, 1336.81524343]]), 'proba': 0.1833767136082604}], 'quality

```

Ustteki kod `biometric_data_simple.txt` verisi üzerinde isletildiginde rapor edilen μ, Σ degerlerini grafikleyince basta paylastigimiz cevrit görüntuleri cikacaktır, yani kumeleme basariyla isletilmistir. Çok Degiskenli Bernoulli Karisimi (Mixture of Multivariate Bernoulli)

Bir karisim Gaussian'lardan olustugu gibi, degisik dagilimlardan da mutesekkil olabilir. Mesela her biri 8x8 boyutlarda ve 64 ogeli duz bir vektor olarak temsil edilen, icinde ikisel (binary) veri tasiyan, yani siyah / beyaz olarak temsil edilen karakterleri gruplama problemimizi ele alalim.



Bu karakterlerin hepsini `bindigit.py` ile sirayla gorebilirsiniz. Mesela ust soldan saga dogru giderken 3 tane sifira benzer karakter goruyoruz, sonra yediye benzer iki goruntu goruyoruz, vs. Burada temsil etmemiz gereken, demek ki, bu 64 hucreli sadece 1 ve 0 degeri tasiyan degerleri temsil etmek. Boy ve agirlikta iki hucreli vektörde sadece reel sayilar vardi. Simdi 64 hucreli vektörde ikisel degerler var. Boyle bir veriyi hangi dagilim en iyi temsil eder? Sadece tek 1 ve 0 olsaydi, o zaman Bernoulli dagilimi kullanirdik,

$$p(x) = \alpha^x (1 - \alpha)^{1-x}$$

α bu dagilimi tanımlayan 0 ve 1 arasında bir degerdir.

Bernoulli'leri cok degiskenli olarak kullanamaz miyiz? Kullaniriz.

$$p(x) = \prod_{d=1}^D \alpha_d^{x_d} (1 - \alpha_d)^{1-x_d}$$

Bu durumda x cok boyutlu, D boyutlu bir vektor, yani $[0 \ 1 \ 1 \ 0 \ \dots \ 1]$ seklinde olacak.

Eger cok degiskenli Bernoulli'lerin karisimini elde etmek istiyorsak, $p(x|z)$ Gaussian yerine cok degiskenli Bernoulli olacak, yani

$$p(x|z) = \prod_{d=1}^D \alpha_{zd}^{x_d} (1 - \alpha_{zd})^{1-x_d}$$

α_{zd} , karisimdaki z 'inci dagilimin d 'inci hucresindeki olasilik degerini verecektir. Tum karisimin dagilimi daha once oldugu gibi

$$p(x) = \sum_z p(z) p(x|z)$$

Suna dikkat etmek lazim – karisim deyince mesela $[0 \ 1 \ 1 \ \dots \ 1]$ vektörü ile $[1 \ 0 \ 1 \ \dots \ 1]$

vektörünü "toplayıp" yeni bir vektör elde etmiyoruz. Bu vektörleri tüm karışımın yoğunluğu $p(x)$ 'e geçince bize bir olasılık değeri veriliyor. Bunun hesaplanması, perde arkasında teker teker karışımındaki tüm bileşenlerin yoğunluğuna teker teker sormak, ve geriye bir cevap vermeden önce ağırlığı kullanarak dengelemek.

Ya da üretimsel (generative) olarak olaya bakarsak, $p(x)$ 'in temsil ettiği yoğunluğa "zar attırarak" ile $[1\ 1\ 1\ \dots\ 0]$, $[1\ 0\ 0\ \dots\ 0]$ gibi vektörler ürettiriyor olabiliriz. Tabii ki bu üretim yoğunluğun kontrolünde olarak, daha olası türden vektörlerin, daha fazla ortaya çıkması anlamına gelecekti.

Üretimsel derken, her veri noktası için bu üretimsel algoritmanın tamamı şöyle:

for $i = 1$ to N

 Olasılık vektörü π 'ye göre zar at

 Sonuca göre $m \leftarrow M$ modelden bir tanesini seç

 O modele $N(x_i|\mu_m, \Sigma_m)$ (ya da onun Bernoulli karşılığı) x_i

Altta yine EM kullanarak gruplama yapan yani etiketleri otomatik olarak bulan kodu sunuyoruz. En sonda `np.argmax(1R.T,axis=0)` ifadesini göreceksiniz. `1R`, `NxD` boyutlu bir matristir, her veri noktasının D kümenin her birine olan aidiyatını olasılık değeri olarak tasir, `argmax` ifadesi satırsal bazda bu aidiyatların en büyüğünün "indisini" dondurur, eğer 3 tane küme var derseniz, o zaman

`[1 1 1 0 2 0 2 ...]`

gibi bir sonuç görülecektir. Demek ki 1. nokta 1. kümeye, 4. nokta 0. kümeye aittir. Hakikaten de basta paylaştığımız resimlere bakarsanız, ilk 3 karakterin birbirine benzediği farkedilecektir.

Sonuç olarak verdiğimiz bu algoritmalar idare edilmeyen (unsupervised) algoritmalar olarak bilinir, çünkü algoritma "kendi başına" giderek noktaların hangi kümeye ait olduğunu hesaplamaktadır. İdare edilen (supervised) yöntemlerde olduğu gibi bir önceden işaretlenmiş örnekler yoktur.

Ne Zaman EM, Ne Zaman MCMC?

Bayes Teorisi ile sofistike olasılıksal modeller oluşturulabilir ve bu modellerin hepsi MCMC ile hesaplanabilir, çözülebilir. Peki ne zaman EM kullanılmalı, ne zaman MCMC kullanılmalı.

Bu noktada Makine Öğrenimi alanında unlu bir hoca Kevin Murphy'nin sözünü hatırlamak iyi olabilir, demistir ki "EM fakir adamın Bayes metotudur". EM her yerde kullanılamayabilir, ama kullanılabilirse cabuk işlediği bilinir, ve probleme uygulanması basittir. Probleminizde karışımlar var ise, etiketleme yapıyorsa, EM ilk akla gelen yöntemlerden biri olabilir. Tabii ki karışımlar ve diğer her tür Bayes problemi MCMC ile de hesaplanabilir.

```
def loginnerprodexp(t,a):  
    eps=1e-15
```

```

t[t>0.] = 1
tmp = np.dot(t,np.exp(a)) + eps
b=np.log(tmp)
return b

def logsumexp(a):
    return np.log(np.sum(np.exp(a), axis=0))

def EMmixtureBernoulli(Y,K,iter,tol):
    N,D=Y.shape
    OMY=1+(-1*Y) # "One minus Y", (1-Y)
    tmp=np.random.rand(N,K)
    tmp2=np.sum(tmp,axis=1).reshape((N,1))
    tmp3=np.tile(tmp2,(1,K))
    lR=np.log(np.divide(tmp, tmp3))
    L = []
    for i in range(iter):
        # lPi log Mixture params Kx1
        lPi=np.tile(-1 * np.log(N),(K,1))+logsumexp(lR).T.reshape((K,1))
        const=np.tile(logsumexp(lR).T.reshape((K,1))),(1,D))
        # lP log Bernoulli params KxD
        lP=loginnerprodexp(Y.T,lR).T - const
        # lOMP log(1-P), also KxD
        lOMP=loginnerprodexp(OMY.T,lR).T-const

        # *** E-step
        lR=np.tile(lPi.T,(N,1))+np.dot(Y,lP.T) + np.dot(OMY,lOMP.T) # + const
        Z=logsumexp(lR.T)

        lR=lR-np.tile(Z.T.reshape((N,1))),(1,K))
        L.append(np.sum(Z))
        if (i>1):
            if np.abs(L[i]-L[i-1]) < tol: break

    iters = i
    return lR,lPi,lP,L,iters

import matplotlib.image as mpimg

K=3
iter=20
Y = np.loadtxt('binarydigits.txt')

attempts=20
Lbest = -np.inf
eps=1e-15

```

```

for attempt in range(attempts):
    lRtmp,lPitmp,lPtmp,L,itors = EMmixtureBernoulli(Y,K,iter,eps)
    if L[itors]>Lbest:
        lR=lRtmp
        lPi=lPitmp
        lP=lPtmp
        Lbest=L[itors]
        itersbest=itors

# show class labels
labels = np.argmax(lR.T,axis=0)
print labels

[0 0 0 2 1 2 1 0 2 1 2 2 1 2 0 0 0 2 0 2 2 0 0 2 0 2 0 2 2 2 2 0 0 0 0 0
 0 2 2 0 0 0 0 0 2 1 0 0 2 1 1 2 1 2 2 2 0 0 2 2 2 2 2 0 2 0 0 1 2 1 2 0 2
 1 1 0 2 2 2 1 0 2 0 1 0 0 2 2 0 0 1 0 1 2 1 0 2 0 1]

```

Elde edilen sonuclara gore, ve paylastigimiz say resimlerdeki siraya bakarsak, mesela ilk uc sayi imajini birbirine benziyor olmasi lazim. Yine ayni sirada gidersek Daha sonra 4. ve 6. sayilarin birbirine benziyor olmasi lazim, ve 8. imajin ilk uc imaja benziyor olmasi lazim, vs. Resimlere bakinca bunun hakikaten boyle oldugunu goruyoruz. Demek ki kumeleme basariyla gerceklestirilmis.

[1] Aaron A. D'Souza, Using EM To Estimate A Probablity Density With A Mixture Of Gaussians, http://www-clmc.usc.edu/~adsouza/notes/mix_gauss.pdf

[2] Bernoulli mixture models for binary images, Alfons Juan, Enrique Vidal

[3] Jebara, T., Machine Learning Lecture Notes

[4] Iain Murray's Octave code on mixture of multivariate bernoullis

[5] <http://code.activestate.com/recipes/577735-expectation-maximization/download/1/>

[6] Bishop, C., Pattern Recognition and Machine Learning

[7] <http://ascratchpad.blogspot.de/2010/01/porting-code-from-matlab-octave-to.html>

[8] http://www.scipy.org/NumPy_for_Matlab_Users/

[9] <http://mathesaurus.sourceforge.net/matlab-numpy.html>