

Ozdegerler ve Ozvektorler ile Imaj Bolmek, Gruplamak

Sentetik görüŖ (machine vision) dalında, karŖımıza ıkan en temel problemlerden biri, pikselleri guruplayarak bir nesneyi tanımlamaktır. Bildiđimiz gibi, robot gözden gelen sayısal bilgiler ışığında, 3-boyutlu dünya bilgisayar için 2 boyutlu bir dünyaya dönüşür. Bu dünyada, pikseller arasındaki bađlantı kaybolmuştur. Yani, elimizdeki veriye tarafsız bir şekilde bakarsak, iki pikselin hangi nesneye ait olduđunu belirten 'gizli' bir kodlama bulmamız imkansızdır. İmgecikler arasında yapmamız gereken bu bađlantıyı, algoritmalar kullanarak sonradan yapmaya mecbur kalıyoruz. Yani, insan gözünün aynen yaptıđı gibi.

İşte burada, guruplama (clustering) yöntemleri denen bir dizi algoritma ve 'düşünce şekli' yardımımıza yetiŖiyor. Çok temel bir konu olduđu için, guruplama hakkında bir çok araştırmacı harıl harıl yeni yaklaşımlar bulmak ile meşguller. Fakat daha hala tek bir kuram diyebileceğimiz 'hep işleyen' bir yaklaşım bulunabilmiş deđil. Her deđişik ortam için, deđişik guruplama yöntemleri kullanılıyor.

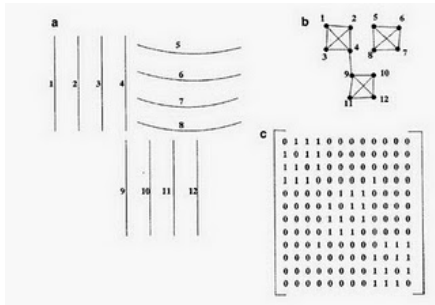
Her yöntemin başarı miktarı ötekine göre farklı. Burada özetleyeceđim yöntem, doğrusal cebir ve çizge spektrum (spectrum) analizi yaparak guruplamayı başarıyor. Bu kelimelerin anlamlarını aŖađıda belirtelim.

Dođrusal cebir, matematik derslerinden hatırlayabileceğimiz gibi, üstü olmayan bilinmezli denklemleri çözmenin aritmetiđi demektir. Yani doğrusal cebir denklemlerindeki bilinmeyenler, x , y , z gibi deđerlerdir. x^2 , y^2 gibi bilinmezlerle bu dalda uğraşılmaz.

Dođrusal cebir oldukça iyi araştırılmış ve kuramları yerine oturmuş bir matematik dalıdır. Her ne kadar üniversitelerde analiz (calculus) dersi kadar önem verilmesine de, bilgisayarların daha güçlenmesi ile doğrusal cebir daha da çok ragbet görür oldu. Dođrusal cebirin bizim için ilgin olan tarafı, son zamanlarda çizge kuramı (graph theory) ile kurulan bađlantılarıdır. Yani, doğrusal cebirden bildiğimiz matris kavramının, çizgeleri çözmek için kullanılabilmesinin keşfedilmesi.

Bir sayısal görüntü ile çizge arasındaki bađlantı nedir diye merak edebilirsiniz... Daha sonra, çizge ile matris arasındaki bađlantı nedir diye merak edilebilir... Bu iki bađlantıyı teker teker özetleyeceđiz.

Daha rahat göz önüne getirebilmek için, aŖađıdaki resme bakabiliriz.



Bu resimde gördüğümüz (b) şeklinde görünen çizge, (a) şeklinde gösterilen 'ekrاندaki' nesnelerin birbirine olan alakasına göre çizilmiş bir çizgedir. Yani, (a) da

görülen ner nesneyi (b) çizgesi üzerinde bir düğüm noktası olarak belirtirsek, o zaman iki nesnenin birbiri ile, herhangi bir ilişkisi durumunda, iki düğüm arasında bir bağlantı kurarız. Bu işlemden sonra elimize gecen çizgeye bitişiklik çizgesi diyoruz.

Bir çizge düğümler ve bunların arasındaki bağlantılardan ibârettir.

Cebire gelelim. Aynen doğrusal cebir de olduğu gibi, çizge üzerinde kurulmuş bir kuram ve yöntem de var. Bu iki konu uzun süre ayrı yollarda geliştiler ve râfine edildiler. Fakat yakın zamanda matematikçiler çizge problemlerini çözmek için doğrusal cebir kullanmaya başladılar. Meselâ bahsettiğimiz bitişiklik çizgesini 'bitişiklik matrisine' çevirirsek, doğrusal cebirin yöntemlerini kullanarak, çizge hakkında bazı sonuçlara varmamız mümkün oluyor. Bu çok ilginç ve harika bir bağlantı, ve bir takım yapıcı yan etkileri var.

Bitişiklik matrisine örnek olarak (c) şekline bakabilirsiniz.

Bu matrisi yaratırken, her çizge üzerindeki her düğüme bir sayı verdiğimizizi unutmayalım; o zaman bu kodlamaya göre düğüm 1 ve düğüm 3 arasında bir ilişki var ise, matrise bakıp X eksen = 1, ve Y eksen = 3 üzerine tekâbül eden matris değerinin 1 olarak tanımlıyoruz.

Arasında ilişki olmayan düğümler, matris üzerinde 0 değeri taşıyorlar.

İşte bu matris üzerinde özdeğer, özvektör yöntemleri kullanarak çizge hakkında sonuçlara varmak mümkün oluyor.

0 ve 1 değerleri yerine yakınlığı piksel değerleri arasındaki farka bağlı olarak ta hesaplayabiliriz.

Gruplamak

Bir imaji nasıl gruplara ayırabiliriz? Ekran piksellerini, çizge (graph) düğümleri olarak gösterebiliriz, sonra bu çizgeyi yakınsallık (affinity) matrisine çevirebiliriz. Bu matris üzerinde öyle işlemler yapalım ki, elimize Wx denen bir vektör geçsin; bu vektörün $1..N$ üyeleri, $1..N$ piksellerinin x gurubuna üyelik katsayısı olsun. Katılım değerleri en fazla olan vektör (gurup), ekran üzerindeki en büyük nesne demektir!

Matematiksel olarak şöyle bir formül kuralım, sadece temsil etmeye uğraşıyoruz, yani bir gurup ve içinde olan pikseller arasında bağlantı kurmak istiyoruz. Piksel ve herhangi bir gurup arasındaki ilişkiyi formül ile kağıt üzerine dökmeyi amaçlıyoruz. Matematik, sayılar arasında alâka kurma sanatıdır. Elimizde şimdilik bir algoritma olmasa bile, temsili olarak bir alâka kurmak mümkün.

$$\sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j = \mathbf{x}^T \mathbf{A} \mathbf{x}$$

Bu formüle göre, W_{ij} , çizge üzerinde gösterilen i ve j düğümü arasındaki bağlantı ağırlığı olacak. x vektörünün içindeki her değer, çizgedeki düğümlerin bu x gurubuna dâhil olma katsayısı olacak. Formülün sol tarafına göre, bu tanımları her i ve j değeri için yaparak sonuçlarını toplamış oluyoruz.

Dikkat, toplam sonucu tek bir sayı, yani bir skalar. Nelerin birbiri ile carpildiği optimizasyon için çok önemli, i ve j arasındaki ağırlığı, i 'nin üyelik ağırlığı ve j 'nin üyelik ağırlığı ile carpiyoruz, bunları tüm diğer kombinasyonlar için yapıyoruz, ama bu carpımları topluyoruz. Carpım daha fazla büyütür, ve maksimizasyon için bu büyüklük daha on planda olacaktır.

Ve bu toplamanın 'en büyük' olduğu yer, görüntü üzerindeki en büyük nesnenin olduğu yerdir! Yani elimizde bir matematiksel maksimizasyon problemi var.

Caprimi tekrar kontrol edelim

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Diyelim ki $i = 2$, $j = 1$. O zaman a_2 , b_{21} ve c_1 'in birbiriyle carpılması gerekir. Hakikaten caprimi elle kontrol edersek bunun olduğunu göreceğiz. İçinde $a_1 \cdot b_{21}$ caprimini içeren terim, sonra c_1 ile caprılacaktır.

Formülün yazarı, maksimizasyon işlemine atlamadan önce, bir matematiksel sınır daha koymaya mecbur olmuş. Maksimizasyon problemlerinde, her sayıyı muazzam büyüklüklere getirerek formül sonucunu sürekli büyütme mümkün olabilirdi. Buna bir sınır getirmek için, sağ tarafta, A 'nin yanına çarpan olarak x vektörünün norm'u (yani uzunluğu) 1 olsun demiş. Altta, bu tanımın açılmış halini görüyorsunuz. (Not: X vektörünün norm'u = X 'in devriği çarpı X). Lagrange formülü şöyle gösterilebilir:

$$w^T A w - \lambda(w^T w - 1)$$

$$w^T A w - \lambda(w^T w - 1) = 0$$

$$\frac{d}{dw} w^T A w - \lambda(w^T w - 1) = 0$$

$$2Aw - 2\lambda w = 0$$

$$2Aw = 2\lambda w$$

$$Aw = \lambda w$$

Ustteki son formül özdeğer (eigenvalue), özvektörler (eigenvector) formülüdür. Rayleigh-Ritz kuramına göre, yukarıdaki formülün enbüyütülmüş x vektörü, A matrisinin en büyük özdeğerine tekabül eden özvektör olacaktır! Dügümler birbirine ne kadar iyi bağlarsa, bir gurubun içsel bağlantısı ve 'gurupluğu' o kadar iyi oluyor.

Bu son formül aşağıda

$$\lambda_{n-k} = \max_{x \perp x_{\lambda_n}, \dots, x_{\lambda_{n-k+1}}} \mathbf{x}^T \mathbf{A} \mathbf{x}$$

Örnek

Altteki imajı gruplarına ayırmaya çalışalım.



```
import matplotlib.pyplot as plt
import numpy as np

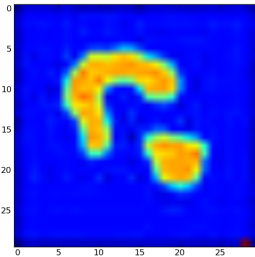
Img = plt.imread("twoObj.jpg")
n = Img.shape[0]
Img2 = Img.flatten(order='C')
nn = Img2.shape[0]

A = np.zeros((nn,nn))

for i in range(nn):
    for j in range(nn):
        A[i,j]=np.exp(-((Img2[i]-Img2[j])**2))

V,D = np.linalg.eig(A)
V = np.real(V)
a = np.real(D[0])
print a

threshold = 0 # filter
a = np.reshape(a, (n,n))
Img[a<threshold] = 255
plt.imshow(Img)
plt.show()
```



Kodda `imread` ile imaji okuduk, elimize 30x30 boyutunda bir matris gecti. Bu matrisi once “düzlestirerek” bir vektor haline getirdik, ki bu vektorun elemanlari yeni bir yakınlilik matrisinin kenarlari olacakti. Sonra bu yeni elemanlari her birini bir digeri ile karsilastirarak yakınliligini hesapladik, bunu piksel degerinin ne kadar yakınlı olduguna bakarak karar verdik, `exp` bunun için kullanildi. Ayrıca yakınlık ve uzaklık kavramini tersine cevrildi, `exp` icinde eksi olması bundan, birbirine “benzer” yani degerleri birbirine yakınlı olan piksellerin farklari az olacaktır, fakat biz bu azligi maksimizasyon problemi için bir fazlalığa cevirme istiyoruz.

Bu noktada A matrisinin özdegerlerini hesaplattik, ve geriye V, D geri geldi. Numpy özdegerleri ve ona tekabül eden özvektorleri büyüklük sırasına dizerek geri getirir, bu sayede sıfırını (ilk) D’ye bakarak en büyük özdeger tekabül eden özvektörü alabildik.

Bu vektörün değerleri ise uyeliği en yüksek olan grubu içeriyordu. Ciplak gözle bakınca bu değerlerin uyelik için pozitif, diğerleri için negatif değerler olduğunu anladık, bu yüzden eşik (threshold) değerini sıfır olarak tanımladık. Eşik altında kalan değerleri grup dışı olarak kabul ettik ve o değerlerin koordinatına 255 piksel değerini atadık, ki üstteki resimde mavi renkli gözüken pikseller bu değerleri temsil ediyor.

Kaynaklar

Sarkar ve Boyer makalesi "Değişimlerin Sayısal Ölçümünü Özellik Organizasyonu Kullanarak Yapmak: Özdeğerler ve Özvektörler". (Quantitative Measures of Change Based on Feature Organization: EigenValues and EigenVectors)

Forsyth ve Ponce kitabı "Bilgisayar Görüşü, Yeni Yaklaşım (Computer Vision, A Modern Approach)