

Felzenswalb Gruplamasi (Felzenswalb Clustering)

Imaj Bolgelerinin Ikili Karsilastirmasi

Bu bolumde bir beyani D ortaya koyacagiz, ki bu beyan, imajdaki iki bilesen (ki imaj gruplamasinin dogru olarak bulmaya calisacagi bilesenler) arasinda bir sinir olup olmadigina dair kanitin olcusu olacak. Beyanin temeli sudur: iki bilesen arasindaki sinirin boyunda yer alan her iki tarafin ogelerinin farklilikina bak, ve onu her bilesenin kendi icindeki farklilikiga gore oranla. Yani bu beyan, bir bilesenin ic farklilikini dis farklilikina kiyaslar, ve bu sebeple verinin yerel karakteristikleri gozetmis olur. Kiyaslama mesela, global, verinin her yerinde aynen gecerli olacak bir sabit esik degerine vs. bagli degildir.

Tanim

Bir bilesen $C \subseteq V$, ki C bir bilesendir (component) ve V cizitin tum noktalaridir, *ic farklilikini*, o C 'nin minimum kapsayan agacinin, yani $MST(C)$ 'sinin en buyuk kenar agirligi olarak aliyoruz. Bu ic farklilikiga $Int(C)$ olarak belirtirsek,

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

ki $w((v_i, v_j))$ bir cizit $G = (V, E)$ 'yi olusturan bir kenar $(v_i, v_j) \in E$ agirligi olarak belirtilir.

Tanim

iki bilesen $C_1, C_2 \subseteq V$ arasindaki farki o iki bileseni birlestiren kenarlardan en ufagi olarak aliyoruz. Iki bilesenin arasinda birden fazla baglanti olmasi mumkundur, tum bunlara bakiyoruz, ve en ufagini aliyoruz.

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j))$$

Eger C_1, C_2 arasinda bir kenar yok ise $Dif(C_1, C_2) = \infty$ kabul ediliyor.

Prensip olarak iki bilesen arasindaki en minimal baglantinin problem cikartabilecegi dusunulebilirdi, niye en az, niye ortalama vs degil? Fakat pratikte bu olcutun cok iyi isledigini gorduk. Hatta iyi olmaktan ote, bu olcutu minimal yerine medyan, ya da diger ceyreksel (quantile) olcute degistirdigimiz zaman (ki bunu yaparak aykiri degerlere -outlier- karsi daha dayanikli olmasini istemistik), algoritma cetrefilligi NP-Zor haline geliyor. Yani gruplama kriterinde ufacik bir degisiklik problemin cozum zorlulugunda muthis bir degisim ortaya cikartiyor.

Simdi iki bilesenin karsilastirma beyani D 'nin tanimina geldik. D olcutu, $Dif(C_1, C_2)$ 'nin $Int(C_1)$ ya da $Int(C_2)$ 'den herhangi birinden daha buyuk olup olmadigina bakar. Ayrica bu karsilastirmayi bir esik degeri uzerinden pay ekleyerek yapar, eger irdeme olumlu ise, iki bilesen arasinda sinir vardir, yoksa yoktur.

$$D(C_1, C_2) = \begin{cases} \text{Dogru} & \text{Eger } \text{Dif}(C_1, C_2) > \text{MInt}(C_1, C_2) \text{ ise} \\ \text{Yanlis} & \text{Diger durumda} \end{cases}$$

Minimum ic fark MInt ise soyle tanimlidir,

$$\text{MInt}(C_1, C_2) = \min(\text{Int}(C_1) + \tau(C_1), \text{Int}(C_2) + \tau(C_2))$$

Esik fonksiyonu τ ustteki irdeledigimiz fark hesaplarinin belli derecelerde disari-dan etkilemek icin koyulmustur. Eger bunu kullanmasaydik sadece Int fonksiy-onunu kullanmamiz gerekecekti, fakat bu olcut tek basina ufak bir bilestenin yerel karakteristiklerini gostermesi acisinden yeterli degildir. Asiri durumda mesela $|C| = 1, \text{Int}(C) = 0$, yani en kucuk C durumudur bu ($|C|$ bilestenin icindeki oge sayisi), icinde tek oge vardır, ve hicbir kenar yoktur, $\text{Int}(C) = 0$.

Bu sebeple iyi bir τ bilestenin buyuklugunu hesaba katarak, ona ters oranli bir rakam olusturursa iyi olur, mesela bir sabit k uzerinden,

$$\tau(C) = \frac{k}{|C|}$$

Bu demektir ki ufak bilestenler icin daha kuvvetli bir ispat ariyoruz, cunku ku-cuk $|C|$, τ 'yu buyutecektir, ve Dif'in ondan buyuk olmasi daha zorlasacaktır. Tabii dikkat edelim, k bir "bilesen sayisi" degildir, yani fonksiyonuna dikkatli bakarsak, eger bilestenler arasinda yeterince buyuk bir fark var ise ufak bilesten-lere hala izin verilmistir.

Algoritma soyledir, girdi olarak $G = (V, E)$ alır, ve V 'yi S bilestenlerine ayirir ki her S icinde ona ait olan kenarlar vardır, yani $S = (C_1, \dots, C_r)$

	$\text{felzenswalb}(G)$
1	E kenarlarini $\pi = (o_1, \dots, o_m)$ seklinde kucukten buyuge dogru sirala.
2	Ilk basta S^0 gruplamasini al. Bu durumda her kenar v_i kendi bilesteni icindedir.
3	for $q = 1, \dots, m$
4	S^{q-1} gruplamasini baz alip S^q gruplamasini soyle yarat; q 'inci siradaki
5	kenarin birlestirdigi noktalar v_i, v_j oldugunu farz edelim, yani $o_q = (v_i, v_j)$.
6	Eger v_i, v_j S^{q-1} gruplamasi icinde farkli iki bilesten icindeyseler, ve $w(o_q)$ her
7	iki bilestenin icsel farkina kiyasla cok kucuk ise, bu iki bilesteni birlestir,
8	yoksa hicbir sey yapma.
9	return $S = S^m$

Ustteki dongu icindeki en son irdelemede icsel farktan bahsediliyor, bu tabii ki $\text{MInt}(C_1, C_2)$. Daha formel sekilde $\text{MInt}(C_1^{q-1}, C_2^{q-1})$ cunku bilestenlerin icerik-leri hangi adimda oldugumuza gore degisebilir, q adiminda bir onceki $q - 1$ 'den bize "miras kalan" gruplamalar ve bilestenler uzerinden is yapiyoruz. Bir sonraki adima ya birlesmis, ya da birlesmemis (ayni) gruplamalari aktariyoruz.

```

import scipy.sparse as sps
import scipy.io as io
import itertools, numpy as np

def threshold(size, c): return c / size

def find(C, u):
    if C[u] != u:
        C[u] = find(C, C[u])           # Path compression
    return C[u]

def union(C, R, u, v):
    u, v = find(C, u), find(C, v)
    if R[u] > R[v]:                     # Union by rank
        C[v] = u
    else:
        C[u] = v
    if R[u] == R[v]:                   # A tie: Move v up a level
        R[v] += 1

class Felzenswalb:
    def __init__(self, threshold, c):
        self.threshold_ = threshold
        self.c_ = c

    def fit(self, X):
        print X.shape
        G = {}
        for i in range(X.shape[0]): G[i] = {}
        for u,v,w in itertools.izip(X.row, X.col, X.data): G[u][v] = w
        E = [(G[u][v],u,v) for u in G for v in G[u]]
        T = set()
        C, R = {u:u for u in G}, {u:0 for u in G}    # Comp. reps and ranks

        ts = {x:threshold(1,self.c_) for x in C}

        for w, u, v in sorted(E):
            if find(C, u) != find(C, v):
                if w <= ts[u] and w <= ts[v]:
                    T.add((u, v))
                    union(C, R, u, v)
                    ts[u] = w + threshold(len(C),self.c_)

        self.labels_ = [np.nan for i in range(len(C))]
        for i in range(len(C)): self.labels_[i] = int(C[i])
        self.T_ = T

import scipy.sparse as sps
import scipy.io as io
X = io.mmread('simple.mtx')
#clf = Felzenswalb(threshold=1,c=1)
clf = Felzenswalb(threshold=1,c=1.0)
clf.fit(X)
print clf.labels_

(5, 5)

```

```

[1, 1, 3, 3, 1]

import pandas as pd, os, sys
syn = pd.read_csv("../kmeans/synthetic.txt", names=['a', 'b'], sep=" ")
data = np.array(syn)

from sklearn.metrics.pairwise import euclidean_distances
X = euclidean_distances(data, data)

X2 = X.copy()
# filter out large values / distances so matrix can be sparse
X2[X > 2000] = 0.0
X3 = sps.lil_matrix(X2)
X4 = sps.triu(X3)
print 'non-zero items', len(X4.nonzero()[0])
print X4.shape

non-zero items 87010
(3000, 3000)

clf = Felzenswalb(threshold=100, c=60000)
clf.fit(X4)

(3000, 3000)

print clf.labels_[:10]

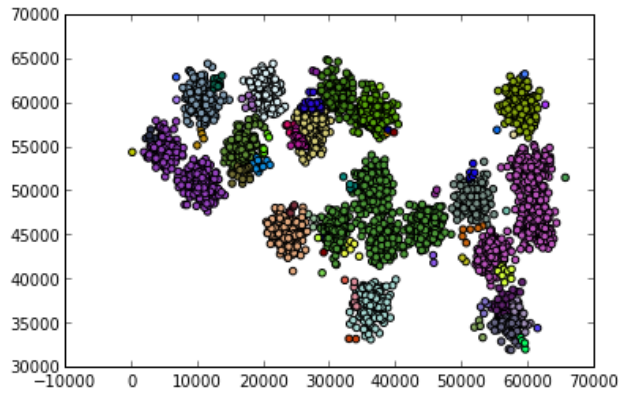
[586, 586, 586, 586, 586, 586, 586, 586, 586, 586]

syn['cluster'] = clf.labels_
print syn[:5]

   a      b  cluster
0  54620  43523     586
1  52694  42750     586
2  53253  43024     586
3  54925  42624     586
4  54973  43980     586

import random
for clust in syn['cluster'].unique():
    tmp = np.array(syn[syn['cluster'] == clust][['a', 'b']])
    plt.scatter(tmp[:,0], tmp[:,1], c=np.random.rand(3,1))
plt.savefig('mstseg_01.png')

```



Kaynaklar

[1] Pedro F. Felzenszwalb and Daniel P. Huttenlocher, *Efficient Graph-Based Image Segmentation*, http://scikit-image.org/docs/dev/auto_examples/plot_segmentations.html