

## Felzenswalb Gruplamasi (Felzenswalb Clustering)

Minimum Kapsayan Agac (Minimum Spanning Tree -MST-) kavramini kullanan Felzenswalb kumelemesini gorecegiz. MST'yi daha once isledik. Literaturde Felzenswalb metotunun imaj gruplamasi icin kullanildigini gorebilirsiniz, biz imaj gruplamasi yapan algoritma icinden veri kumelemesi yapan kısmi cikarttik ve ayri bir sekilde paylasiyoruz.

Bu gruplama algoritmasının daha once paylastigimiz Kruskal'in MST koduna yapılacak birkac ekleme sayesinde elde edilebilmesi hakikaten ilginc. Normal MST cizitin ayri bolgelerinde ayri agaclar yaratir ve bunlari yavas yavas buyutur, gerektiği noktalarda onlari birlestirir. Felzenswalb sadece bu birlestirme mantigini biraz degistirip, ayri agacları bir grup olarak kabul eder, ve bu grupların kendi icinde benzerligin maksimal gruplararası benzerligin minimal olacak hale getirir. Bu sekilde bildik Kruskal isletilince cok hizli isleyen hizli bir gruplama algoritması elde edilmiş olur!

Felzenswalb'in matematiginde once imaj bolgelerinin (ya da veri kumeleri olarak dusunebiliriz) ikili karsilastirmaya bir olcut gerekir. Bu bolumde bir beyan D'yi ortaya koyacagiz, ki bu beyan, imajdaki iki bileşen (ki imaj gruplamasının dogru olarak bulmaya calisacagi bileşenler) arasında bir sinir olup olmadigina dair kanitin olcusu olacak. Beyanın temeli sudur: iki bileşen arasındaki sinirin boyunda yer alan her iki tarafın ogelerinin farklıligına bak, ve onu her bileşenin kendi icindeki farklıliga gore oranla. Yani bu beyan, bir bileşenin ic farklıligını dis farklıligına kiyaslar, ve bu sebeple verinin yerel karakteristikleri gozetmiş olur. Kiyaslama mesela, global, verinin her yerinde aynen gecerli olacak bir sabit esik degerine vs. bagli degildir.

### Tanim

Bir bileşen  $C \subseteq V$ , ki  $C$  bir bileşendir (component) ve  $V$  cizitin tum noktalaridir, *ic farklıligini*, o  $C$ 'nin minimum kapsayan agacının, yani  $MST(C)$ 'sinin en büyük kenar ağırlığı olarak aliyoruz. Bu ic farklıligı  $Int(C)$  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  $(v_i, v_j) \in E$  ağırlığı olarak belirtilir.

### Tanim

İki bileşen  $C_1, C_2 \subseteq V$  arasındaki farkı o iki bileşeni birlestiren kenarlardan en ufagi olarak aliyoruz. İki bileşenin arasında birden fazla baglanti olması 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  $C_1, C_2$  arasında bir kenar yok ise  $\text{Dif}(C_1, C_2) = \infty$  kabul ediliyor.

Prensip olarak iki bileşen arasındaki en minimal bağlantının problem çıkartılabileceği düşünülebilirdi, niye en az, niye ortalama vs değil? Fakat pratikte bu ölçüt çok iyi işlediğini gördük. Hatta iyi olmaktan öte, bu ölçütü minimal yerine medyan, ya da diğer çeyrekssel (quantile) ölçütü degistirdiğimiz zaman (ki bunu yaparak aykırı değerlere -outlier- karşı daha dayanıklı olmasını istemistik), algoritma çetrefilligi NP-Zor haline geliyor. Yani gruplama kriterinde ufak bir değişiklik problemin çözüm zorluluğunda muthis bir değişim ortaya çıkartıyor.

Şimdi iki bileşenin karşılaştırma beyanı  $D$ 'nin tanımına geldik.  $D$  ölçütü,  $\text{Dif}(C_1, C_2)$ 'nin  $\text{Int}(C_1)$  ya da  $\text{Int}(C_2)$ 'den herhangi birinden daha büyük olup olmadığının bakar. Ayrıca bu karşılaştırmayı bir eşik değeri üzerinden pay ekleyerek yapar, eğer irdeme olumlu ise, iki bileşen arasında sınır vardır, yoksa yoktur.

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

Minimum iç fark  $M\text{Int}$  ise şöyle tanımlıdır,

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

Eşik fonksiyonu  $\tau$  üstteki irdelediğimiz fark hesaplarının belli derecelerde disaridan etkilemek için koyulmuştur. Eger bunu kullanmasaydık sadece  $\text{Int}$  fonksiyonunu kullanmamız gerekecekti, fakat bu ölçüt tek başına ufak bir bileşenin yerel karakteristiklerini göstermesi açısından yeterli değildir. Asiri durumda mesela  $|C| = 1, \text{Int}(C) = 0$ , yani en küçük  $C$  durumudur bu ( $|C|$  bileşenin içindeki öğe sayısı), içinde tek öğe vardır, ve hiçbir kenar yoktur,  $\text{Int}(C) = 0$ .

Bu sebeple iyi bir  $\tau$  bileşenin büyüklüğünü hesaba katarak, ona ters oranlı bir rakam oluştursa iyi olur, mesela bir sabit  $k$  üzerinden,

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

Bu demektir ki ufak bileşenler için daha kuvvetli bir ispat arıyoruz, çünkü küçük  $|C|$ ,  $\tau$ 'yu buyutecektir, ve  $\text{Dif}$ 'in ondan büyük olması daha zorlasacaktır. Tabii dikkat edelim,  $k$  bir "bileşen sayısı" değildir, yani fonksiyonuna dikkatli bakarsak, eğer bileşenler arasında yeterince büyük bir fark var ise ufak bileşenlere hala izin verilmistir.

Algoritma şöyledir, girdi olarak  $G = (V, E)$  alır, ve  $V$ 'yi  $S$  bileşenlerine ayırır ki her  $S$  içinde ona ait olan kenarlar vardır, yani  $S = (C_1, \dots, C_r)$

Üstteki dongu içindeki en son irdilemede ıcsel farktan bahsediliyor, bu tabii ki  $M\text{Int}(C_1, C_2)$ . Daha formel şekilde  $M\text{Int}(C_1^{q-1}, C_2^{q-1})$  çünkü bileşenlerin içerikleri hangi adımda olduğumuza göre değişebilir,  $q$  adımında bir önceki  $q - 1$ 'den

---

```

felzenswalb(G)
1      E kenarlarını  $\pi = (o_1, \dots, o_m)$  şeklinde küçükten büyüğe doğru sırala.
2      İlk basta  $S^0$  gruplamasını al. Bu durumda her kenar  $v_i$  kendi bileşeni içindedir.
3      for  $q = 1, \dots, m$ 
4           $S^{q-1}$  gruplamasını baz alıp  $S^q$  gruplamasını şöyle yarat;  $q$ 'inci sıradaki
5          kenarın birleştirdiği noktaları  $v_i, v_j$  olduğunu farz edelim, yani  $o_q = (v_i, v_j)$ .
6          Eğer  $v_i, v_j$   $S^{q-1}$  gruplaması içinde farklı iki bileşen içindeyseler, ve  $w(o_q)$  her
7          iki bileşenin icsel farkına kıyasla çok küçük ise, bu iki bileşeni birleştir,
8          yoksa hiçbir şey yapma.
9      return  $S = S^m$ 

```

---

bize “miras kalan” gruplamalar ve bileşenler üzerinden iş yapıyoruz. Bir sonraki adıma ya birleşmiş, ya da birleşmemiş (aynı) gruplamaları aktarıyoruz.

Felzenswalb gruplamasının Python ile yazılmış örneği alttadır, daha hızlı işleyen C++ bazlı kodu burada [2] bulabilirsiniz.

```

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

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

S = {}

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

class Felzenswalb:
    def __init__(self, min_size, c):
        self.min_size_ = min_size
        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]]

```

```

E = sorted(E)
T = set()
C, R = {u:u for u in G}, {u:0 for u in G}    # Comp. reps and ranks
S = {u:1 for u in range(len(G))}

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

for w, u, v in 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, S)
            ts[u] = w + threshold(S[u],self.c_)

for _, u, v in E:
    if find(C, u) != find(C, v):
        if S[C[u]] < self.min_size_ or S[C[v]] < self.min_size_:
            union(C, R, u, v, S)

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

```

Basit bir ornek

```

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

(5, 5)
[1, 1, 3, 3, 1]

import scipy.sparse as sps
import scipy.io as io, random
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)

```

```

import felz
clf = felz.Felzenswalb(min_size=20,c=800)
clf.fit(X4)

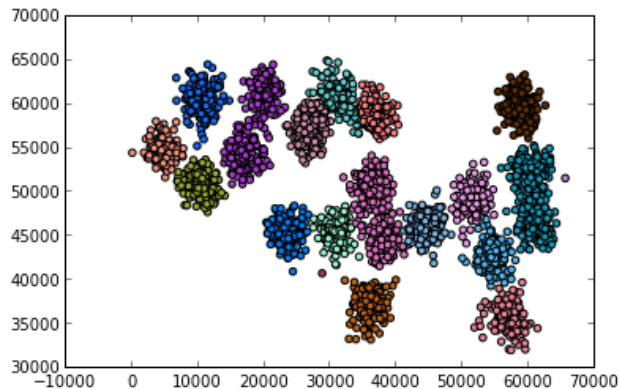
(3000, 3000)

syn['cluster'] = clf.labels_
print len(syn['cluster'].unique()), 'clusters found'
print syn[:5]

19 clusters found
      a      b  cluster
0  54620  43523      120
1  52694  42750      120
2  53253  43024      120
3  54925  42624      120
4  54973  43980      120

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](http://scikit-image.org/docs/dev/auto_examples/plot_segmentations.html)
- [2] Bayramli B., <https://github.com/burakbayramli/felzenszwalb>