

## 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 algoritmasinin daha once paylastigimiz Kruskal'in MST koduna yapılacak birkac ekleme sayesinde elde edilebilmesi hakikaten ilginç. 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 benzerligi maksimal, gruplararası benzerligi minimal olacak hale getirir. Bu sekilde bildik Kruskal isletilince çok hızlı isleyen hızlı bir gruplama algoritması elde edilmiş olur!

Felzenswalb veri olarak, MST gibi, bir cizit alır ve bu cizit veri noktalarının arasındaki yakınlık bilgisini iceren bir matris olarak verilebilir. Mesela 5 veri noktası var ise, 0. nokta ile 1. nokta arasındaki '10' büyüklüğündeki bir mesafe  $A(0,1) = 10$  olarak kaydedilebilir. Kumeler birbirine yakın ogeler arasından seçilir.

Algoritmanın önemli avantajlarından biri kume sayısının (KMeans'de olduğu gibi) önceden tanımlanmasına gerek olmamasıdır. Belli esik değerleri tanımlanınca kume sayısı kendiliğinden bulunur. Tabii "disaridan verilen bir parametreyi baska biriyle degistirmiş mi olduk?" sorusu akla gelebilir, Felzenswalb'in aldığı hiperparametreler kabaca ayarlanabilen ve veri kaynağı bağlamında akla uygun şeyler, ve belli değerler etrafında stabilite ortaya çıkabiliyor. Kiyasla "kume sayısı" ciddi bir rakam ve degismesi mümkün değil.

Felzenswalb'in matematiginde önce imaj bölgelerinin (ya da veri kümeleri olarak düşünebiliriz) arasında ikili karşılaştırma için bir ölçüt gerekir. Bu bölümde bir beyan  $D$ 'yi ortaya koyacağız, ki bu beyan, imajdaki iki bileşen (ki imaj gruplamasının doğru olarak bulmaya çalışacağı bileşenler) arasında bir sınır olup olmadigina dair kanıtın ölçüsü olacak. Beyanın temeli şudur: iki bileşen arasındaki sınırın boyunda yer alan her iki tarafın ogelerinin farklılığına bak, ve onu her bileşenin kendi içindeki farklılığa göre oranla. Yani bu beyan, bir bileşenin iç farklılığını dış farklılığına kıyaslar, ve bu sebeple verinin yerel karakteristikleri gözettiği olur. Kıyaslama mesela, global, verinin her yerinde aynen geçerli olacak bir sabit esik değerine vs. bağlı değildir.

### Tanım

Bir bileşen  $C \subseteq V$ , ki  $C$  bir bileşendir (component) ve  $V$  cizitin tüm noktalarıdır, *ic farklılığını*, o  $C$ 'nin minimum kapsayan ağacının, yani  $MST(C)$ 'sinin en büyük kenar ağırlığı olarak alıyoruz. Bu iç farklılığı  $Int(C)$  olarak belirtirsek,

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

ki  $w((v_i, v_j))$  bir cizit  $G = (V, E)$ 'yi oluşturan 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 birleştiren kenarlardan en ufak olarak alıyoruz. İki bileşenin arasında birden fazla bağlantı olması mümkündür, tüm bunlara bakıyoruz, ve en ufakını alıyoruz.

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

Eğer  $C_1, C_2$  arasında bir kenar yok ise  $\text{Dif}(C_1, C_2) = \infty$  kabul ediliyor.

Prensip olarak iki bileşen arasındaki en minimal bağlantının problem çıkartabileceği düşünülebilirdi, niye en az, niye ortalama vs değil? Fakat pratikte bu ölçütün çok iyi işlediği görülmüştür. Hatta iyi olmaktan öte, bu ölçütü minimal yerine medyan, ya da diğer çeyreksele (quantile) ölçüte değiştirildiği zaman (ki bunu yaparak aykırı değerlere -outlier- karşı daha dayanıklı olmasını istenmişti), algoritma çetrefilliği 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.

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

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. Eğer bu kullanılmazaydı sadece  $\text{Int}$  fonksiyonu kullanılması 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 büyütecektir, ve  $\text{Dif}$ 'in ondan büyük olması daha zorlaşacaktır.

Tabii dikkat edelim,  $k$  bir “bilesen 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  $S = (C_1, \dots, C_r)$

---

```

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  $S^m$ 

```

---

Ustteki dongu içindeki en son irdilemede icsel farktan bahsediliyor, bu tabii ki  $MInt(C_1, C_2)$ . Daha formel şekilde  $MInt(C_1^{q-1}, C_2^{q-1})$  çünkü bileşenlerin içerikleri hangi adımda olduğumuza göre değişebilir,  $q$  adımında bir önceki  $q - 1$ 'den bize “miras kalan” gruplamalar ve bileşenler üzerinden iş yapıyoruz. Bir sonraki adıma ya birleşmiş, ya da birleşmemiş (aynı) gruplamaları aktarıyoruz.

Aynı algoritmanın biraz daha fazla formül içeren hali [3]

---

```

felzenswalb(G)
1      Butun kenarları küçükten büyüğe doğru sırala.
2      İlk basta kenar  $v_i$  kendi bileşeni içinde olsun, buna  $S^0$  gruplaması diyelim.
3      for tum kenarlar  $e_i = (v_1, v_2) \in E$  için
4           $v_1 \in C_1$  ve  $v_2 \in C_2$ 'nin birbirinden ayrı, ayrı bileşenler içinde
5          ama aynı kenarı içeren noktalar olduğunu düşünelim,
6          if  $w(e_i) \leq MInt(C_1, C_2)$ 
7               $C_1$  ve  $C_2$ 'yi birleştir
8          else
9               $S^i = S^{i-1}$ 
10     return  $S$ 

```

---

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[v]
    else:
        C[u] = v
        S[v] = 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]
```

Biraz daha ceterfil bir ornek

```
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)
```

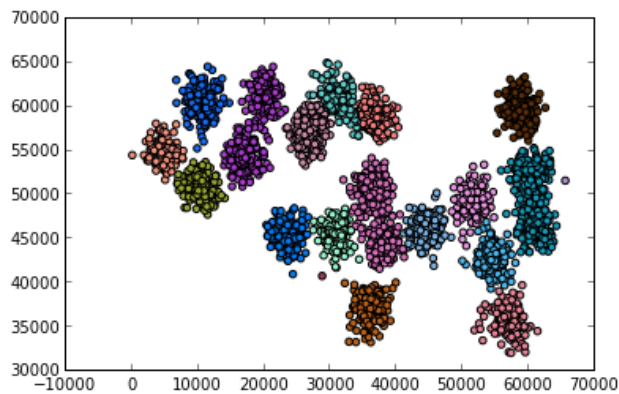
```
import felz
clf = felz.Felzenswalb(min_size=20,c=800)
clf.fit(X4)

(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')
```



Simdi *SVD ile Kumeleme* yazisinda gordugumuz kelime gruplamasi ornegini Felzenswalb ile gruplayalim.

```
import scipy.linalg as lin
import scipy.sparse as sps
import itertools, sys
sys.path.append('../svdcluster/')
import leven

words = np.array(
    ['the', 'be', 'to', 'of', 'and', 'a', 'in', 'that', 'have',
     'I', 'it', 'for', 'not', 'on', 'with', 'he', 'as', 'you',
     'do', 'at', 'this', 'but', 'his', 'by', 'from', 'they', 'we',
     'say', 'her', 'she', 'or', 'an', 'will', 'my', 'one', 'all',
     'would', 'there', 'their', 'what', 'so', 'up', 'out', 'if',
     'about', 'who', 'get', 'which', 'go', 'me', 'when', 'make',
     'can', 'like', 'time', 'no', 'just', 'him', 'know', 'take',
     'people', 'into', 'year', 'your', 'good', 'some', 'could',
     'them', 'see', 'other', 'than', 'then', 'now', 'look',
     'only', 'come', 'its', 'over', 'think', 'also', 'back',
     'after', 'use', 'two', 'how', 'our', 'work', 'first', 'well',
     'way', 'even', 'new', 'want', 'because', 'any', 'these',
     'give', 'day', 'most', 'us'])

(dim,) = words.shape
f = lambda (x,y): leven.levenshtein(x,y)
res=np.fromiter(itertools.imap(f, itertools.product(words, words)),dtype=np.uint8)
A = sps.coo_matrix(np.reshape(res, (dim,dim)))
print A.shape

(100, 100)
```

Kumelemeyi yapalim, min\_size=2 sectik cunku ufak kumeler de mumkun.

```
import felz
clf = felz.Felzenswalb(min_size=1.5,c=0.2)
clf.fit(A)
labels = np.array(clf.labels_)
c = len(np.unique(labels))
print c, 'clusters found'
```

```
(100, 100)
16 clusters found
```

```
for c in np.unique(labels):
    print 'cluster', c
    print words[labels==c]
```

```
cluster 9
['a' 'I' 'as' 'at' 'up' 'also' 'use' 'because' 'us']
cluster 10
['in' 'it' 'with' 'which' 'its' 'first']
cluster 13
['of' 'for' 'on' 'from' 'or' 'one' 'if' 'people' 'only' 'after' 'our'
'work']
cluster 15
['the' 'be' 'have' 'he' 'by' 'they' 'we' 'her' 'she' 'my' 'their' 'who'
'get' 'me' 'when' 'time' 'year' 'them' 'see' 'other' 'then' 'over' 'back'
'even' 'give']
cluster 18
['to' 'not' 'do' 'so' 'go' 'no' 'know' 'into' 'good' 'now' 'look' 'two'
'how' 'new' 'most']
cluster 22
['this' 'his' 'him' 'think']
cluster 31
['and' 'an' 'all' 'can' 'want' 'any']
cluster 39
['that' 'what' 'than']
cluster 42
['but' 'out' 'about' 'just']
cluster 59
['make' 'like' 'take']
cluster 63
['you' 'your']
cluster 66
['would' 'could']
cluster 75
['some' 'come']
cluster 88
['will' 'well']
cluster 89
['say' 'way' 'day']
cluster 95
['there' 'these']
```

## Kaynaklar

[1] Pedro F. Felzenszwalb and Daniel P. Huttenlocher, *Efficient Graph-Based Image Segmentation*, <http://cs.brown.edu/~pff/segment/>

[2] Bayramli B., <https://github.com/burakbayramli/felzenszwalb>

[3] Mihai-Cotizo Sima, *An extension of Felzenszwalb-Huttenlocher segmentation to 3D point clouds*, 2012