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 kismi cikarttik ve ayri bir sekilde paylasiyoruz. Bu gruplama algoritmasinin daha once paylastigimiz Kruskal'in MST koduna yapilacak birkac ekleme sayesinde elde edilebilmesi hakikaten ilginc. Normal MST cizitin ayri bolgelerinde ayri agaclar yaratir ve bunlari yavas yavas buyutur, gerektigi noktalarda onlari birlestirir. Felzenswalb sadece bu birlestirme mantigini biraz degistirip, ayri agaclari bir grup olarak kabul eder, ve bu gruplarin kendi icinde benzerligin maksimal gruplararasi benzerligin minimal olacak hale getirir. Bu sekilde bildik Kruskal isletilince cok hizli isleyen hizli bir gruplama algoritmasi elde edilmis olur!

Felzenswalb veri olarak, MST gibi, bir cizit alir, bu cizit veri noktalarinin arasindaki yakinlik bilgisini iceren bir matris olarak verilebilir. Mesela 5 veri noktasi var ise, 0. nokta ile 1. nokta arasindaki '10' buyuklugundeki bir mesafe $A(0,1) = 10$ olarak kaydedilebilir. Kumeler birbirine yakin ogeler arasindan secilir.

Algoritmanin onemli avantajlarindan biri kume sayisinin (KMeans'de oldugu gibi) onceden tanimlanmasina gerek olmamasidir. Belli esik degerleri tanimlaninca kume sayisi kendiliginden bulunur. Tabii "disaridan verilen bir parametreyi baska biriyle degistirmis mi olduk?" sorusu akla gelebilir, Felzenswalb'in aldigi hiperparametreler kabaca ayarlanabilen ve veri kaynagi baglaminda akla uygun seyler, ve belli degerler etrafinda stabilite ortaya cikabiliyor. Kiyasla "kume sayisi" ciddi bir rakam ve degismesi mumkun degil.

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 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 farkliligina bak, ve onu her bilesenin kendi icindeki farkliliga gore oranla. Yani bu beyan, bir bilesenin ic farkliligini dis farkliligina 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 farkliligini*, o C'nin minimum kapsayan agacinin, yani $MST(C)$'sinin en buyuk kenar agirligi olarak aliyoruz. Bu ic farkliligi $Int(C)$ 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.

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

Eger C_1, C_2 arasinda bir kenar yok ise $\mathrm{Dif}(C_1, C_2) = \infty$ 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, $\mathrm{Dif}(C_1, C_2)$'nin $\mathrm{Int}(C_1)$ ya da $\mathrm{Int}(C_2)$'den herhangi birinden daha buyuk olup olmadigina bakar. Ayrica bu karsilastirmayi bir esik degeri uzerinden pay ekleyerek yapar, eger irdeleme olumlu ise, iki bilesen arasinda sinir vardir, yoksa yoktur.

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

Minimum ic fark MInt ise soyle tanimlidir,

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

Esik fonksiyonu τ ustteki irdeledigimiz fark hesaplarinin belli derecelerde disaridan etkilemek icin koyulmustur. Eger bunu kullanmasaydik sadece Int fonksiyonunu kullanmamiz gerekecekti, fakat bu olcut tek basina ufak bir bilesenin yerel karakteristiklerini gostermesi acisindan yeterli degildir. Asiri durumda mesela $|C| = 1, \mathrm{Int}(C) = 0$, yani en kucuk C durumudur bu ($|C|$ bilesenin icindeki oge sayisi), icinde tek oge vardir, ve hicbir kenar yoktur, $\mathrm{Int}(C) = 0$.

Bu sebeple iyi bir τ bilesenin 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 bilesenler icin daha kuvvetli bir ispat ariyoruz, cunku kucuk $|C|$, τ'yu buyutecektir, ve Dif'in ondan buyuk olmasi daha zorlasacaktir.

Tabii dikkat edelim, k bir "bilesen sayisi" degildir, yani fonksiyonuna dikkatli bakarsak, eger bilesenler arasinda yeterince buyuk bir fark var ise ufak bilesenlere hala izin verilmistir.

Algoritma soyledir, girdi olarak $G = (V, E)$ alir, ve V'yi S bilesenlerine ayirir ki her S icinde ona ait olan kenarlar vardir, yani $S = (C_1, .., C_r)$

`felzenswalb`(G)

E kenarlarini $\pi = (o_1, .., o_m)$ seklinde kucukten buyuge dogru sirala.
Ilk basta S^0 gruplamasini al. Bu durumda her kenar v_i kendi bileseni icindedir.
`for` $q = 1, .., m$
  S^{q-1} gruplamasini baz alip S^q gruplamasini soyle yarat; q'inci siradaki kenarin birlestirdigi noktalari v_i, v_j oldugunu farz edelim, yani $o_q = (v_i, v_j)$. Eger v_i, v_j S^{q-1} gruplamasi icinde farkli iki bilesen icindeyseler, ve $w(o_q)$ her iki bilesenin icsel farkina kiyasla cok kucuk ise, bu iki bileseni birlestir, yoksa hicbir sey yapma.
`return` $S = S^m$

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

Felzenswalb gruplamasinin Python ile yazilmis ornegi alttadir, daha hizli isleyen C++ bazli kodu surada [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
```

```python
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

```python
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]
```

```python
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)
```

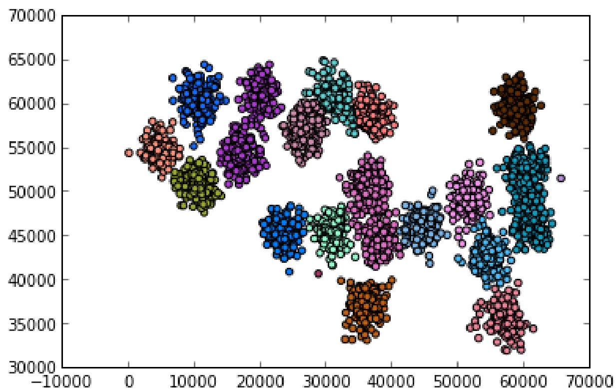
```
import felz
clf = felz.Felzenswalb(min_size=20,c=800)
clf.fit(X4)
```

```
(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=2,c=0.1)
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://scikit-image.org/docs/dev/auto_examples/plot_segmentations.html

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