Oge Kumeleri, Ikisel Matris Ayristirmasi (Binary Matrix Factorization), FPGrowth

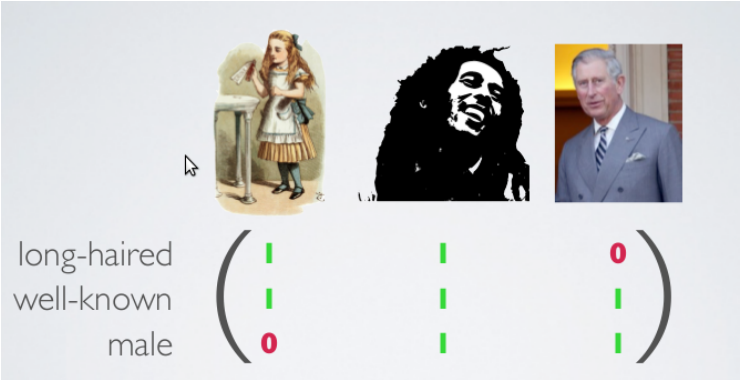
Veri madenciligi denince pek cok kisinin aklina gelen ilk ornek, aslinda, sık bulunan öğe kumeleri (frequent itemsets) ornegidir: "filanca ulkeden sitemize gelen musterilerin ayni zamanda vs ozelliklerinin oldugunu da kesfettik" gibi.

Benzer bir ornek, ki bu alan oge kumelerinin aslinda en onemli cikis sebeplerinden birisidir, alisveris sepeti analizidir. Musterinin her alisverisinde sepetinde belli mallar vardir, ve bu mallarin hangilerinin ayni anda, ayni sepette oldugu analiz edilmeye ugrasilir. Eger surekli ekmek ve recel ayni anda aliniyorsa, bu bilgi kullanilarak belki mallarin daha iyi konumlandirilmasi yapilacaktir, vs. Sık bulunan oge kumeleri teknikleri bazen degisik adlar altinda da gecebiliyor, mesela alaka madenciligi (association mining) gibi. Algoritma olarak kullanilan pek cok teknik var, APriori iyi bilinenlerden, FPGrowth ondan daha hizli calisan ve daha tercih edilen bir teknik.

Bir diger alternatif ikisel matris ayristirmasi (binary matrix factorization -BMF-) kullanmaktir [3]. Aynen SVD'de oldugu gibi BMF de bir matrisi ayristirir, fakat uc matris yerine iki matrise ayristirir ve hem sonuc matrisi hem de ayristirilan matrisler sadece 0 ya da 1 degerini tasiyabilirler. Yani bu ayristirma sonuc matrislerinin ikisel olmasini mecbur tutar, negatif olmayan matris ayristirmasinin (non-negative matrix factorization) sonuc matrisinin pozitif degerler tasimasini mecbur kilmasi gibi. Bunlar birer kisitlama (constraint) ve bu sonuc o kisitlamalara gore ortaya cikiyor. *Dikkat*: BMF icin toplama islemi $1 + 0 = 1, 1 + 1 = 1, 0 + 0 = 0$ olarak tekrar tanimlanir, yani mantiksal OR islemi haline gelir.

Ayristirma oncesi hangi kerte (rank) k degerine gecmek istedigimizi biz belirtiriz. BMF'nin oge kumeleri madenciligi icin faydasi surada: oge kumeleri ararken baktigimiz ogeler kategorik seylerdir, alisveris sepeti orneginde mesela ekmek, recel gibi. Kategorik ogeleri daha once one-hot kodlamasi (encoding) ile $1/0$ degerleri tasiyan yeni kolonlara gecirebildigimizi gormustuk. Yani tamamen kategorik degerler tasiyan veriler tamamen $1/0$ tasiyacak sekilde tekrar kodlanabilir, yani ikisel matris haline getirilebilir. Bu ikisel matrisi ayristirdigimiz zaman ve kendileri de ikisel olan iki yeni matris elde ettigimizde ise bir anlamda boyut indirgemesi yapmis oluruz, yani sanki ana matrisi "ozetleriz". Iste bu ozet, ozellikle carpilan "baz" matris, oge kumelerinin hangileri oldugu hakkinda ipuclari iceriyor olabilir.

Bir ornek uzerinde gorelim, mesela altta Alice (A), Bob Marley (B) ve Prens Charles (C) verileri var. Bu kisiler icin saci uzun mu (long-haired), ünlü mü (well-known) ve bay mı (male) verileri var.



Bu matris uzerinde ikisel ayristirma yaparsak, k = 2

Eger kontrol etmek istersek, matris carpimi yapmamiz gerekir, bunun icin

```
a = np.array([[1,  0],
              [1,  1],
              [0,  1]], dtype=bool)
b = np.array([[1,  1,  0],
              [0,  1,  1]], dtype=bool)

print np.dot(a,b)
```

```
[[ True  True False]
 [ True  True  True]
 [False  True  True]]
```

0 ve 1 degerleri gormek icin 1 ile carpmak yeterli

```
print 1*np.dot(a,b)
```

```
[[1 1 0]
 [1 1 1]
 [0 1 1]]
```

Sonuc baslangic matrisi ile ayni, demek ki `bool` tipi matris tanimlayinca Numpy caprimi `dot`, carpim sirasindaki toplama islemi icin aritmetik toplama yerine VEYA (OR) kullanmasi gerektigini anladi.

Simdi ayristirmayi analiz edelim, ozellikle sol taraftaki carpilan "baz" matrise bakalim.. *Matris Carpimi* yazisindan hareketle, bu yazidaki kolon kombinasyon bakisini kullanalim (tabii toplamanin BMF icin OR oldugunu unutmadan), o zaman soldaki baz matrisin dikey, kolon bazli olarak, bir ozet oldugunu gorebiliyoruz. Cunku carpan sag taraf bu kolonlari alip onlari belli sekillerde "kombine ederek" nihai (orijinal) matrisi ortaya cikartabilmeli. Bu sebeple soldaki carpilan matris bir ozet olmali / baz olusturmali, ve bunun yan etkisi olarak kolonlardaki

degerlerde belli bir kalip / oruntu (pattern) olmali. O zaman her baz kolonunda birbiriyle alakali olan ogeler ayni anda 1 degeri tasiyor olacaktir.

Sonuca gore uzun sacli ve unlu olmak (1. kolon) arasinda baglanti varmis , ayrica erkek olmak ve unlu olmak (2. kolon) arasinda da baglanti varmis :) Veriye gore boyle en azindan.. Bu sonucu orijinal matrise bakarak ta kontrol edebiliriz.

Ayristirma Kodlamasi

BMF ozel bir hesaptir ve Numpy / Scipy icinde mevcut degildir. Bunun icin ozel bir kutuphane cagrisi kullanabiliriz. Nimfa denen paket icinde [4] gerekli kodlar var. Kurduktan sonra usttteki ornegi soyle cozebiliriz;

```
import nimfa
import pandas as pd
import scipy.sparse as sp

def __fact_factor(X):
    return X.todense() if sp.isspmatrix(X) else X

A = np.array([[1., 1., 0],
              [1., 1., 1.],
              [0, 1., 1.]])

fctr = nimfa.mf(A,
              seed = "nndsvd",
              rank = 2,
              method = "bmf",
              max_iter = 40,
              initialize_only = True,
              lambda_w = 1.1,
              lambda_h = 1.1)

res = nimfa.mf_run(fctr)

threshold = 0.2
res1 = __fact_factor(res.basis())
res2 = __fact_factor(res.coef())
res1 = np.abs(np.round(res1 - 0.5 + threshold))
res2 = np.abs(np.round(res2 - 0.5 + threshold))
res1 = pd.DataFrame(res1, index=['long-haired','well-known','male'])
res2 = pd.DataFrame(res2, columns=['A','B','C'])
print res1
print '\n'
print res2
```

```
              0  1
long-haired   1  0
well-known    1  1
male          0  1


   A  B  C
0  1  0  0
1  0  1  1
```

Sonuc neredeyse tipatip ayni; sadece carpan matriste [0,B] kordinati 1 degil, fakat bize lazim olan baz matris ayni cikti.

BMF hakkinda bazi ek bilgiler: [2]'ye gore en az hatali BMF hesaplamak NP-hard zorlugunda, yani 3SAT gibi, ya da Seyahat Eden Satis Elemani (Traveling Salesman) problemi gibi ki bu problem kombinatoryel (combinatorial) optimizasyon problemi, yani cozum icin tum olasiliklar denendigi ve kisayolun mevcut olmadigi cesitten problem. Fakat, yaklasiksal BMF metotlari oldukca hizli, ayrica seyreklik cok fark yaratiyor (yani seyreklik iyi) ki kategorik veriler gercek dunyada cogunlukla seyrek olarak goruluyor. Eldeki 2000 tane mal cesidi icinden bir sepette ancak 5-10 tane urun oluyor mesela, tum 2000 tane mali bir sepete koymak mumkun degil.

Simdi baska bir ornek gorelim.

```
data = [
['outlook=sunny', 'temparature=hot', 'humidity=high', 'windy=false', 'play=no'],
['outlook=sunny', 'temparature=hot', 'humidity=high', 'windy=true', 'play=no'],
['outlook=overcast', 'temparature=hot', 'humidity=high', 'windy=false', 'play=yes'],
['outlook=rainy', 'temparature=mild', 'humidity=high', 'windy=false', 'play=yes'],
['outlook=rainy', 'temparature=cool', 'humidity=normal', 'windy=false', 'play=yes'],
['outlook=rainy', 'temparature=cool', 'humidity=normal', 'windy=true', 'play=no'],
['outlook=overcast', 'temparature=cool', 'humidity=normal', 'windy=true', 'play=yes'],
['outlook=sunny', 'temparature=mild', 'humidity=high', 'windy=false', 'play=no'],
['outlook=sunny', 'temparature=cool', 'humidity=normal', 'windy=false', 'play=yes'],
['outlook=rainy', 'temparature=mild', 'humidity=normal', 'windy=false', 'play=yes'],
['outlook=sunny', 'temparature=mild', 'humidity=normal', 'windy=true', 'play=yes'],
['outlook=overcast', 'temparature=mild', 'humidity=high', 'windy=true', 'play=yes'],
['outlook=overcast', 'temparature=hot', 'humidity=normal', 'windy=false', 'play=yes'],
['outlook=rainy', 'temparature=mild', 'humidity=high', 'windy=true', 'play=no']
]
```

Hava ile alakali bazi veriler [1] bunlar; bu veriler tahmin (outlook), sicaklik (temparature), nem (humidity), ruzgar (windy), disarida oyun oynayan var mi (play). Mesela ilk satirda tahmin gunesli, isi sicak, nem yuksek, ruzgar yok ve oyun oynayan yok. Bu sekilde bir suru satir. Biz bu veride bir kalip olup olmadigina bakacagiz.

FPGrowth

Oge kumeleri bulmak icin BMF haricinde bir yontem FPGrowth yontemidir [1,2]. Bu yontem once her ogeden (tek basina) kac tane oldugunu sayar, belli bir esik degeri `minsup` altinda olanlari atar, sonucu siralar. Bu liste bir yapisina isaret eden bir baslik yapisi haline gelir. Agacin kendisini olusturmak icin veri satirlari teker teker islenir, her satirdaki her oge icin baslik yapisindaki en fazla degeri tasiyan oge once olmak uzere tepeden baslanip alta dogru uzayan bir agac yapisi olusturulur. Agactaki her dugum altindaki dugumun sayisal toplamini tasir. Madencilik icin alttan baslanarak yukari dogru cikilir (amac en uste ulasmak) ve bu sirada ogeler `minsup` altinda ise, atilirlar. Sonucta ulasilan ve atilmayan yollar bir oge kumesini temsil ederler. [2]'deki kodu [1]'den aldigimiz ustteki veriye uygularsak, sonuc soyle:

```
import fp
items = fp.fpgrowth(data, minsup=6)
for x in items:
    if len(x) > 1: print x

<fp.node instance at 0x5017ef0>
   Null Set   1
     play=yes   9
       humidity=high   1
         windy=true   1
           temparature=mild   1
       windy=false   6
         humidity=high   2
           temparature=mild   1
         humidity=normal   4
           temparature=mild   1
       humidity=normal   2
         windy=true   2
           temparature=mild   1
     humidity=high   2
       windy=true   2
         temparature=mild   1
     windy=false   2
       humidity=high   2
         temparature=mild   1
     humidity=normal   1
       windy=true   1
   Null Set   1
     play=yes   6
   Null Set   1
     play=yes   6
set(['play=yes', 'humidity=normal'])
set(['play=yes', 'windy=false'])
```

Bulunan sonuclar iki tane (tek ogeli sonuclar da var ama onlari eledik). Bunlar hakikaten veri icindeki kaliplari temsil ediyorlar. Fena degil.

Kiyas icin BMF uzerinden madencilik yapalim. Once one-hot kodlamasi yapalim, ve ornek icin bir veri satirini ekrana basalim,

```
from sklearn.feature_extraction import DictVectorizer
import pandas as pd, re

def one_hot_dataframe(data, cols, replace=False):
    vec = DictVectorizer()
    mkdict = lambda row: dict((col, row[col]) for col in cols)
    tmp = data[cols].apply(mkdict, axis=1)
    vecData = pd.DataFrame(vec.fit_transform(tmp).toarray())
    vecData.columns = vec.get_feature_names()
    vecData.index = data.index
    if replace is True:
        data = data.drop(cols, axis=1)
        data = data.join(vecData)
    return (data, vecData, vec)
```

```
cols = ['outlook','temparature','humidity','windy','play']
df = pd.DataFrame(data,columns=cols)
# kolon ismini veriden cikart, cunku tekrar geri koyulacak
# fpgrowth icin veri icinde olmasi lazim
df = df.applymap(lambda x: re.sub('.*?=','',x))
df2, _, _ = one_hot_dataframe(df, cols, replace=True)
# tek ornek ekrana bas
print df2.ix[0]
```

```
humidity=high       1
humidity=normal     0
outlook=overcast    0
outlook=rainy       0
outlook=sunny       1
play=no             1
play=yes            0
temparature=cool    0
temparature=hot     1
temparature=mild    0
windy=false         1
windy=true          0
Name: 0, dtype: float64
```

Simdi BMF isletelim, k = 4

```
import nimfa
import scipy.sparse as sp

def __fact_factor(X):
    return X.todense() if sp.isspmatrix(X) else X

fctr = nimfa.mf(np.array(df2).T, seed = "nndsvd",
              rank = 4, method = "bmf",
              max_iter = 40, initialize_only = True,
              lambda_w = 1.1, lambda_h = 1.1)

res = nimfa.mf_run(fctr)

threshold = 0.2
res1 = __fact_factor(res.basis())
res2 = __fact_factor(res.coef())
res1 = np.abs(np.round(res1 - 0.5 + threshold))
res2=  np.abs(np.round(res2 - 0.5 + threshold))
res1 = pd.DataFrame(res1,index=df2.columns)
print res1
```

```
                  0  1  2  3
humidity=high     1  0  0  1
humidity=normal   0  1  0  0
outlook=overcast  0  0  1  0
outlook=rainy     1  0  0  0
outlook=sunny     0  0  0  1
play=no           0  0  0  1
play=yes          0  1  1  0
temparature=cool  0  0  0  0
temparature=hot   0  0  0  0
```

```
temparature=mild  1  0  0  0
windy=false       0  0  1  0
windy=true        1  0  0  0
```

Bu sonuclari kategoriksel hale cevirip tekrar ekrana basalim,

```
for i in range(4):
    print np.array(df2.columns)[res1.ix[:,i] == 1]
```

```
['humidity=high' 'outlook=rainy' 'temparature=mild' 'windy=true']
['humidity=normal' 'play=yes']
['outlook=overcast' 'play=yes' 'windy=false']
['humidity=high' 'outlook=sunny' 'play=no']
```

1. sonuc atlanabilir, buradaki "kalabalik" orada bir kalip olmadigina dair bir isaret. Ayristirma sonucu bu tur kolonlar ortaya cikabilir, diger kolonlardaki kaliplar butunu temsil etmeye *tam* yetmemisse, arta kalan her turlu gereklilik bir yerlere tikilabiliyor, bu normal. 2. sonuc FPGrowth sonucunda var, guzel. 3. sonuc ta neredeyse ayni, sadece ek olarak outlook=overcast var. Fakat, 3. sonuc aslinda onemli bir kalip iceriyor olabilir, yani kalmasi daha iyi olur. Ayrica tavsiyemiz BMF sonuclari uzerinde her sonuc icin ogelerini teker teker eleyerek kalanlarin kuvvetini kontrol etmek, yani alt yeni sonuclara bakmak. Bu sekilde 3. uzerinde eleme yaparken outlook=overcast elenince ana kalibi bulurduk.

4. sonuc ise cok onemli bir kalip ve FPGrowth bunu tamamen kacirmis!

Sebep FPGrowth'un cozume lokal olarak erismeye calisiyor olmasi, kiyasla BMF butune (global) bakiyor [3]. Bu ne demektir? Bir ayristirmanin ne oldugunu dusunursek, bir matrisi olusturan carpimi ayristiriyoruz ve bu ayristirma olduktan sonra iki matris elde ediyoruz. Bu iki matris ozgundur (unique). Yani belli bir ikisel matrisi olusturan carpim sadece tek bir sekilde olabilir. Buradan hareketle diyebiliriz ki bu ayristirma butunu goze alarak yapilmalidir, sagi, solu tutan ama kosesi tutmayan bir ayristirma olmaz. Bu sebeptendir ki ayristirma cozumunden belli bir kapsayicilik bekleyebiliriz.

FPGrowth ise olaya yerel bakiyor; agac olustururken degisik bir sira takip edilirse mesela degisik agaclar ortaya cikabilir. Ayrica her onemli iliski muhakkak ozgun bir dal yapisinda olmayabilir. Madencilik algoritmasi alt dallardan baslar ve yukariya dogru cikar, fakat bu her zaman iyi bir yontem midir?

Kodlama Notlari

Su kod `np.round(num - 0.5 + threshold)` kullanimi yuvarlama (rounding) yapiyor, cunku Nimfa 1 degeri yerine 0.9, 0.8 gibi degerler uretebiliyor, ayrica 0.1 gibi degerler de oluyor. Biz bildigimiz yuvarlama `.5` sonrasi uzerini 1 yapmak yerine belli bir esik degeri (threshold) uzerinden yuvarlama yaptik. Yani esik=0.2 ise 0.7 alta yuvarlanir ve 0 olur, 0.9 esik ustunde oldugu icin uste yuvarlanir 1 olur.

BMF icin kerte k kullanici tarafindan secilmeli, ama bu durum SVD, ya da k-Means gibi diger yapay ogrenim metotlarindan farkli degildir. Bu oynanmasi gereken, kesfedilmesi gereken bir deger.

Kaynaklar

[1] Ian H. Witten, Eibe Frank, Mark A. Hall, *Data Mining Practical Machine Learning Tools and Techniques*

[2] Harrington, P., *Machine Learning in Action*

[3] http://www.mpi-inf.mpg.de/~pmiettin/slides/BooleanMatrixFactorization Antwerp_slides.pdf

[4] http://nimfa.biolab.si/#installation