SVD, Toplu Tavsiye (Collaborative Filtering)

Diyelim ki Star Trek (ST) dizini ne kadar begendigini herkes sezonlara gore isaretliyor. Bu veriyi altta bulabilirsiniz.

```
from pandas import *

data = DataFrame (
    [[5, 5, 0, 5],
     [5, 0, 3, 4],
     [3, 4, 0, 3],
     [0, 0, 5, 3],
     [5, 4, 4, 5],
     [5, 4, 5, 5]],
    columns=['Ben','Tom','John','Fred'],
    index=['S1','S2','S3','S4','S5','S6']
    )
data
```

```
    Ben  Tom  John  Fred
S1    5    5     0     5
S2    5    0     3     4
S3    3    4     0     3
S4    0    0     5     3
S5    5    4     4     5
S6    5    4     5     5
```

Veriye gore Tom, 3. sezonu 4 seviyesinde sevmis. 0 degeri o sezonun seyredilmedigini gosteriyor. Toplu Tavsiye algoritmalari verideki diger kisilerin bir urunu, diziyi, vs. ne kadar begendiginin verisinin diger "benzer" kisilere tavsiye olarak sunulmasidir, ya da ondan once, bir kisinin daha almadigi urunu, seyretmedigi sezonu, dinlemedigi muzigi ne kadar "begeneceginin" tahmin edilmesidir.

Peki benzerligin kriteri nedir, ve benzerlik nelerin arasinda olculur?

Benzerlik, urun seviyesinde, ya da kisi seviyesinde yapilabilir. Eger urun sevisinde ise, tek bir urun icin tum kullanicilarin verdigi nota bakilir. Eger kullanici seviyesinde ise, tek kullanicinin tum urunlere verdigi begeni notlari vektoru kullanilir.

Mesela 1. sezondan hareketle, o sezonu begenen kisilere o sezona benzer diger sezonlar tavsiye edilebilir. Kisiden hareketle, mesela John'a benzeyen diger kisiler bulunarak onlarin begendigi urunler John'a tavsiye edilebilir.

Urun ya da kisi bazinda olsun, benzerligi hesaplamanin da farkli yollari var. Genel olarak benzerlik olcutunun 0 ile 1 arasinda degisen bir sayi olmasini tercih ediyoruz ve tum ayarlari ona gore yapiyoruz. Diyelim ki ki elimizde A, B vektorleri var, ve bu vektorlerin icinde begeni notlari var. Benzerlik cesitleri soyle:

Oklit Benzerligi (Euclidian Similarity)

Bu benzerlik $1/(1+mesafe)$ olarak hesaplanir. Mesafe karelerin toplaminin karekoku (yani Oklitsel mesafe, ki isim buradan geliyor). Bu yuzden mesafe 0 ise (yani iki "sey" arasinda hic mesafe yok, birbirlerine cok yakinlar), o zaman hesap 1 dondurur (mukemmel benzerlik). Mesafe arttikca bolen buyudugu icin benzerlik sifira yaklasir.

Pearson Benzerligi

Bu benzerligin Oklit'ten farkliligi, sayi buyuklugune hassas olmamasidir. Diyelim ki birisi her sezonu 1 ile begenmis, digeri 5 ile begenmis, bu iki vektorun Pearson benzerligine gore birbirine esit cikar. Pearson -1 ile +1 arasinda bir deger dondurur, alttaki hesap onu normalize ederek 0 ile 1 arasina ceker.

Kosinus Benzerligi (Cosine Similarity)

Iki vektoru geometrik vektor olarak gorur ve bu vektorlerin arasinda olusan aciyi (daha dogrusu onun kosinusunu) farklilik olcutu olarak kullanir.

$$\cos\theta = \frac{A \cdot B}{||A||||B||}$$

```
from numpy import linalg as la
def euclid(inA,inB):
    return 1.0/(1.0 + la.norm(inA - inB))

def pearson(inA,inB):
    if len(inA) < 3 : return 1.0
    return 0.5+0.5*np.corrcoef(inA, inB, rowvar = 0)[0][1]

def cos_sim(inA,inB):
    num = float(np.dot(inA.T,inB))
    denom = la.norm(inA)*la.norm(inB)
    return 0.5+0.5*(num/denom)
```

```
print np.array(data['Fred'])
print np.array(data['John'])
print np.array(data['Ben'])
print pearson(data['Fred'],data['John'])
print pearson(data['Fred'],data['Ben'])
```

```
[5 4 3 3 5 5]
[0 3 0 5 4 5]
[5 5 3 0 5 5]
0.551221949943
0.906922851283
```

```
print cos_sim(data['Fred'],data['John'])
print cos_sim(data['Fred'],data['Ben'])
```

```
0.898160909799
0.977064220183
```

Simdi tavsiye mekanigine gelelim. En basit tavsiye yontemi, mesela kisi bazli olarak, bir kisiye en yakin diger kisileri bulmak (matrisin tamamina bakarak) ve onlarin begendikleri urunu istenilen kisiye tavsiye etmek. Benzerlik icin ustteki olcutlerden birini kullanmak.

Fakat belki de elimizde cok fazla urun, ya da kullanici var. Bir boyut azaltma islemi yapamaz miyiz?

Evet. SVD yontemi burada da isimize yarar.

$A = USV$

elde edecegimiz icin, ve S icindeki en buyuk degerlere tekabul eden U, V degerleri siralanmis olarak geldigi icin U, V'nin en bastaki degerlerini almak bize "en onemli" bloklari verir. U'nun mesela en onemli iki kolonu bize iki boyuttaki sezon kumelerini verebilir, V'nin en onemli iki (en ust) satiri bize iki boyutta bir kisi kumesi verebilir.

O zaman begeni matrisi uzerinde SVD uygulayalim,

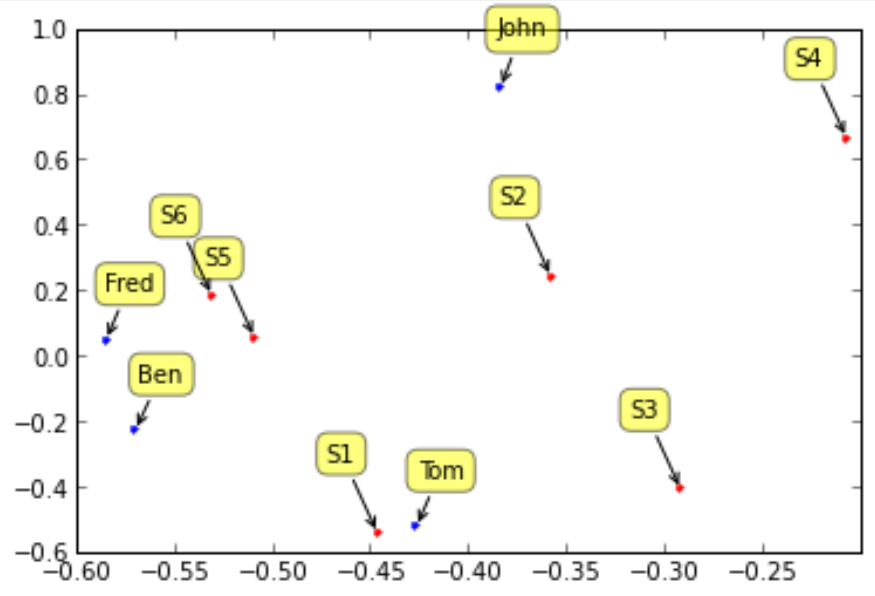
```
from numpy.linalg import linalg as la
U,Sigma,VT=la.svd(data)
u = U[:,:2]
v=VT[:2,:].T
u,v
```

```
(array([[-0.44721867, -0.53728743],
       [-0.35861531,  0.24605053],
       [-0.29246336, -0.40329582],
       [-0.20779151,  0.67004393],
       [-0.50993331,  0.05969518],
       [-0.53164501,  0.18870999]]),
 array([[-0.57098887, -0.22279713],
       [-0.4274751 , -0.51723555],
       [-0.38459931,  0.82462029],
       [-0.58593526,  0.05319973]]))
```

degerleri elimize gecer. U ve VT matrisleri

```
def label_points(d,xx,yy,style):
    for label, x, y in zip(d, xx, yy):
        plt.annotate(
            label,
            xy = (x, y), xytext = style,
            textcoords = 'offset points', ha = 'right', va = 'bottom',
            bbox = dict(boxstyle = 'round,pad=0.5', fc = 'yellow', alpha = 0.5),
            arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=0'))
```

```
plot(u[:,0],u[:,1],'r.')
label_points(data.index, u[:, 0], u[:, 1],style=(-10, 30))
plot(v[:,0],v[:,1],'b.')
label_points(data.columns, v[:, 0], v[:, 1],style=(20, 20))
```



Cok guzel! SVD bize urun bazinda sezon 5 ve 6'nin bir kume olusturdugunu, Ben ve Fred'in de kisi bazinda ayri bir kume oldugunu gosterdi.

Azaltilmis boyutlari nasil kullaniriz? Yeni bir kisiyi (mesela Bob) ele alinca, bu kisinin verisini oncelikle azaltilmis boyuta "indirgemek" gerekiyor. Cunku artik islem yaptigimiz boyut orasi. Peki bu indirgemeyi nasil yapariz? SVD genel formulunu hatirlarsak,

$A = USV$

Azaltilmis ortamda

$A = U_k S_k V_k$

Diyelim ki gitmek istedigimiz nokta azaltilmis V, o zaman V_k'yi tek basina birakalim,

$U_k^{-1} A = U_k^{-1} U S V_k$

U_k, V_k matrisleri ortonormal, o zaman $U_k^{-1} U_k = I$ olacak, yani yokolacak

$U_k^{-1} A = S V_k$

$S^{-1} U_k^{-1} A = V_k$

Cok fazla ters alma islemi var, her iki tarafin devrigini alalim

$A^T (U_k^{-1})^T (S^{-1})^T = V_k^T$

$U_k^{-1} = U_k^T$ o zaman devrigin devrigini almis oluyoruz, tekrar basa donuyoruz, U_k degismeden kaliyor

$A^T U_k (S^{-1})^T = V_k^T$

S ise kosegen matris, onun tersi yine kosegen, kosegen matrisin devrigi yine kendisi

$A^T U_k S^{-1} = V_k^T$

Bazi kod ispatlari, u'nun ortonormal olmasi:

```
np.dot(u.T,u)
```

```
array([[  1.00000000e+00,  -6.51063405e-17],
       [ -6.51063405e-17,   1.00000000e+00]])
```

Dogal olarak ..e-17 gibi bir sayi sifira cok yakin, yani sifir kabul edilebilir. Devrik ve tersin ayni oldugunu gosterelim: Iki matrisi birbirinden cikartip, cok kucuk bir sayidan buyukluge gore filtreleme yapalim, ve sonuc icinde bir tane bile True olup olmadigini kontrol edelim,

```
not any(U.T-la.inv(U) > 1e-15)
```

```
True
```

Yeni Bob verisi

```
bob = np.array([5,5,0,0,0,5])
```

O zaman

```
S_k = np.eye(2)*Sigma[:2]
bob_2d = np.dot(np.dot(bob.T,u),la.inv(S_k))
bob_2d
```

```
array([-0.37752201, -0.08020351])
```

Simdi kosinus benzerligi kullanarak bu izdusumlenmis yeni vektorun hangi diger vektorlere benzedigini bulalim.

```
for i,user in enumerate(v):
   print data.columns[i],cos_sim(user,bob_2d)
```

```
Ben 0.993397525045
Tom 0.891664622942
John 0.612561691287
Fred 0.977685793579
```

Bu sonuca gore yeni kullanici Bob en cok Ben ve Fred'e benziyor. Bu iki kullanicinin yuksek not verdigi ama Bob'un hic not vermedigi sezonlari alip Bob'a tavsiye olarak sunabiliriz!

Kaynaklar

http://www.igvita.com/2007/01/15/svd-recommendation-system-in-ruby/

Harrington, P., Machine Learning in Action