SVD, Toplu Tavsiye (Collaborative Filtering)

Diyelim ki Star Trek (ST) dizisini ne kadar begendigini 4 tane kullanici sezonlara gore isaretlemis. Bu ornek veriyi alttaki gibi gosterelim.

```
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, ST dizisinin 3. sezonunu 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 sunabilir, ya da ondan once, bir kisinin daha almadigi urunu, seyretmedigi sezonu, dinlemedigi muzigi ne kadar "begeneceginin" tahmin eder. Kaggle sitesi uzerinden yapilan unlu Netflix yarismasinin amaci buydu - ayrica tahmin edilen ve gercek begeni notunun hata payinin hesabi icin RMSE hesabi kullanilmisti.

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 begeni notlarini tasiyan A, B vektorleri var (baska veri turu de tasiyor olabilir tabii), 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 usttteki 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. Bu en onemli kolon ya da satirlari alarak azaltilmis bir boyut icinde benzerlik hesabi yapmak islemlerimizi hizlandirir. Bu azaltilmis boyutta kumeleme algoritmalarini devreye sokabiliriz; 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
```

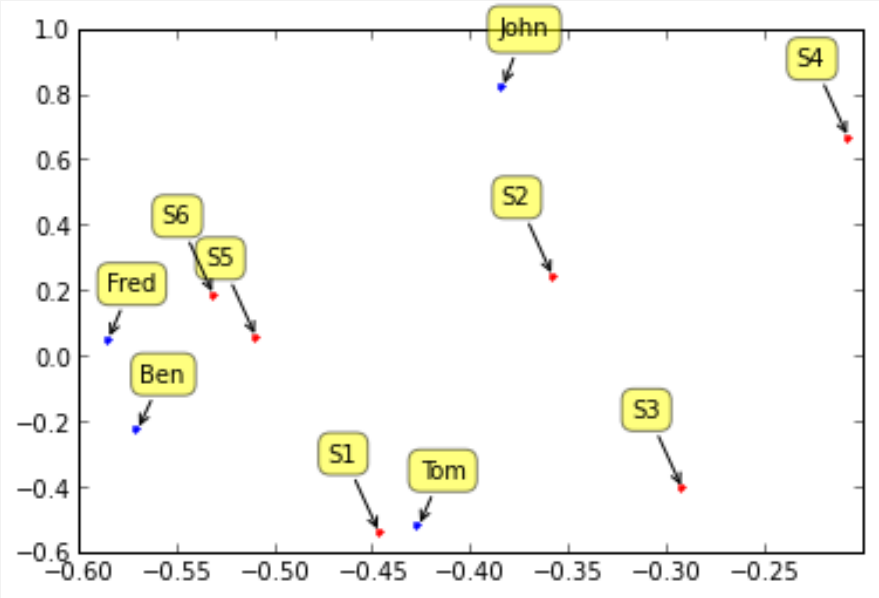
```
(array([[-0.44721867, -0.53728743],
       [-0.35861531,  0.24605053],
       [-0.29246336, -0.40329582],
       [-0.20779151,  0.67004393],
       [-0.50993331,  0.05969518],
       [-0.53164501,  0.18870999]]),
 array([[-0.57098887, -0.22279713],
       [-0.4274751 , -0.51723555],
       [-0.38459931,  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 aynen diger verilerin indirgendigi gibi azaltilmis boyuta "indirgememiz" 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])
```

Ustte eye ve Sigma ile ufak bir takla attik, bunun sebebi svd cagrisindan gelen Sigma sonucunun bir vektor olmasi ama usttteki islem icin kosegen bir "matrise" ihtiyacimiz olmasi. Eger birim (identity) matrisini alip onu Sigma ile carparsak, bu kosegen matrisi elde ederiz.

Simdi mesela 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
```

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

0.1 Movielens 1M Verisi

Bu veri seti 6000 kullanici tarafindan yaklasik 4000 tane filme verilen not / derece (rating) verisini iceriyor, 1 milyon tane not verilmis, yani 4000 * 6000 = 24 milyon olasilik icinde sadece 1 milyon veri noktasi dolu. Bu oldukca seyrek bir matris demektir.

Verinin ham hali diger ders notlarini da iceren ust dizinlerde var, veriyi SVD ile kullanilir hale getirmek icin movielens_prep.py adli script kullanilir. Islem bitince movielens.csv adli bir dosya script'te gorulen yere yazilacak. Bu dosyada olmayan derecelendirmeler bos olacaktir . Bu bosluklari sifirlarsak, scipy seyrek matrisi o noktalari atlayacaktir. Ardindan bu seyrek matris uzerinde SVD isletilebilir.

```
import pandas as pd, os
df = pd.read_csv("%s/Downloads/movielens.csv" % os.environ['HOME'],sep=';')
print df.shape
df = df.ix[:,1:] # id kolonunu atla
df = df.ix[:,:3700] # sadece filmleri
movie_avg_rating = np.array(df.mean(axis=0))
df = np.array(df)
print df.shape
```

```
(6040, 3731)
(6040, 3700)
```

Birlestirici script kullanicilarin demografik bilgisini de matrise ekliyor. Biz ustte basitlestirme amacli bu kismi hafizada tekrar cikarttik, ama isteyen olursa bu ek ile de ilginc ek analizler yapilabilir.

Veriyle calisma seklimiz soyle olacak: gercek dunya sartlarini yakinen taklit edebilmek amaciyla rasgele bazi veri notlari sececegiz, ve bu notlari, nereden geldigini bir kenara yazip SVD'ye vermeden once matristen silecegiz.

```
import scipy.sparse as sps
nonzero_idx = []; df_train = df.copy()
np.random.seed(0)
while True:
    i = np.random.randint(0, df.shape[0])
    j = np.random.randint(0, df.shape[1])
    if not np.isnan(df[i,j]):
        nonzero_idx.append([i,j]); df_train[i,j] = 0.
        if len(nonzero_idx) == 200: break # 200 nokta sec

df_train[np.isnan(df_train)] = 0.0
dfs_train = sps.coo_matrix(df_train)

for ii,(i,j) in enumerate(nonzero_idx):
    print i,j,df[i, j]
    if ii == 10: break
```

```
659 3219 4.0
6001 1316 4.0
201 2431 3.0
307 1104 4.0
5995 3341 3.0
5566 136 4.0
4040 573 3.0
3291 2867 5.0
816 3166 5.0
3821 2711 4.0
3441 2274 5.0
```

Ustte secilen verilerden birkacinin i,j kordinatlari ve derece verisi goruluyor. Bu verileri alirken yerlerine sifir degeri koyduk, boylece coo_matris ile seyrek matris yaratirken o sifirlar hafizaya alinmayacak.

Artik bu matris uzerinde SVD isletebiliriz, hatta seyrek ortamda calismasi icin ozel yazilmis seyrek SVD kullanacagiz.

```
import scipy.sparse.linalg as slin
import scipy.linalg as la
U,Sigma,VT=slin.svds(dfs_train,k=10)
```

Bu is bittikten sonra elimizde U matrisi var. Bu matrisi soyle kullanacagiz. Veriden cektigimiz test noktalarini simdi algoritmamiza verecegiz ve derecelendirme verisini tahmin ettirecegiz. Sonra bildigimiz gercek deger ile bu tahmini karsilastiracagiz. Karsilastirma icin RMSE adli bir olcut kullanilacak, yani tahmin ve gercek deger farkinin kareleri alip toplanacak, sonuc test nokta sayisina bolunup karekoku alinacak. Bu tavsiye sistemleri (recommendation engines) icin iyi bilinen bir tekniktir, Netflix yarismasinda da mesela bu olcut kullanilmistir.

Tahmin icin bize bir kullanici ve film sorulacak. Algoritmamiza gore U matrisi icinde bu kullaniciya

en yakin diger kullaniciyi bulacagiz. Bunu daha hizli bir sekilde yapabilecegiz cunku SVD ile boyut azaltmasi yaptik, ve daha az kolonu baz alarak karsilastirma yapabiliriz. Benzerlik olcutu icin yazinin basindaki alternatiflere bakabilirsiniz, biz simdilik Oklit (euclid) benzerligi kullandik.

Kodlama acisindan U'nin i'inci satirini diger tum U'lari ile yanyana getirip benzerlik hesabi yaptik, bu tur isler icin map bicilmis kaftan. Pythonic kodlama stili zaten daha az for dongusu kullanmayi, daha fonksiyonel kodlama yapilmasini ozendiriyor.

Bazi puf noktalar: benzerlikler olculurken i'inci veri kendisi ile yanyana gelir dogal olarak ve tabii ki "aynen" benzer sonucunu rapor eder, bunu engellemek icin (cunku tekrar i'inci kullaniciyi degil, baska bir kullaniciyi bulmak istiyoruz), i'nin kendisine olan mesafesini yapay olarak cok buyuk negatif bir sayiya (-100) set ederiz.

Ayni sekilde benzerlik hesaplari yapildiktan sonra o hesaplarin uzerinden bir daha geceriz ve j'inci urune oy vermemis tum kullanicilarin benzerlik hesabini yine yapay olarak -100'e set ederiz. Cunku o kullanicilarla da ilgilenmiyoruz. Aradigimiz urune oy vermis olan kullanicilarla ilgileniyoruz.

O diger kullanici bulunduktan sonra o kullanicinin aradigimiz verdigi urune atadigi dereceyi tahminimiz olarak kullaniyoruz. Altta 10 ornek uzerinde sonuclar gorulebilir.

```
__top_k__ = 6

def euclid(inA,inB):
    return 1.0/(1.0 + la.norm(inA - inB))

def find_sim(i,U,j):
    sims = np.array(map(lambda x: euclid(x, U[i,:__top_k__]), U[:,:__top_k__]))
    # kendi kendine benzerligi en kotu yap
    sims[i] = -100
    # eger kullanici o urune oy vermemisse, benzerligi yine kotulestir
    for i,v in enumerate(sims):
        if df_train[i,j] == 0: sims[i] = -100
    return np.argmax(sims)

rmse = 0; n = 0
for ii,(i,j) in enumerate(nonzero_idx):
    isim = find_sim(i,U,j)
    rmse += (df_train[isim, j] - df[i, j])**2
    n += 1
    print i, 'kullanici', 'icin en yakin kisi', isim, ',urun',j, 'icin oyu', df[isim, j],
        ', gercek oy', df[i, j]
    if ii == 10: break

print "rmse", np.sqrt(rmse / n)
```

```
659 kullanici icin en yakin kisi 3704 ,urun 3219 icin oyu 4.0 , gercek oy 4.0
6001

kullanici icin en yakin kisi 4801 ,urun 1316 icin oyu 5.0 , gercek oy 4.0
201
```

```
kullanici icin en yakin kisi 2495 ,urun 2431 icin oyu 4.0 , gercek oy 3.0
307

kullanici icin en yakin kisi 5160 ,urun 1104 icin oyu 5.0 , gercek oy 4.0
5995

kullanici icin en yakin kisi 2528 ,urun 3341 icin oyu 5.0 , gercek oy 3.0
5566

kullanici icin en yakin kisi 2488 ,urun 136 icin oyu 4.0 , gercek oy 4.0
4040

kullanici icin en yakin kisi 3310 ,urun 573 icin oyu 3.0 , gercek oy 3.0
3291

kullanici icin en yakin kisi 5871 ,urun 2867 icin oyu 4.0 , gercek oy 5.0
816

kullanici icin en yakin kisi 1368 ,urun 3166 icin oyu 3.0 , gercek oy 5.0
3821

kullanici icin en yakin kisi 1691 ,urun 2711 icin oyu 5.0 , gercek oy 4.0
3441

kullanici icin en yakin kisi 535 ,urun 2274 icin oyu 5.0 , gercek oy 5.0
rmse 1.08711461301
```

Bu alanda RMSE 0.9 civari cok iyidir, ustte seed ile oynayarak verinin degisik kisimlarindan test ornekleri cekip cikarabilirsiniz. Ustte biz 200 icinden 10'ununu kontrol ettik, tamamina bakilabilir.

Kaynaklar

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

Harrington, P., Machine Learning in Action

http://stats.stackexchange.com/questions/31096/how-do-i-use-the-svd-in-collaborative-filtering