

SVD, Toplu Tavsiye (Collaborative Filtering)

Diyelim ki Star Trek (ST) dizisini ne kadar begendigini 4 tane kullanıcı sezonlara göre işaretlemiş. Bu örnek veriyi alttaki gibi gösterelim.

```
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 göre Tom, ST dizisinin 3. sezonunu 4 seviyesinde sevmiş. 0 değeri o sezonun seyredilmediğini gösteriyor.

Toplu Tavsiye algoritmaları verideki diğer kişilerin bir ürünü, diziyi, vs. ne kadar begendığının verisinin diğer “benzer” kişilere tavsiye olarak sunulabilir, ya da ondan önce, bir kişinin daha almadığı ürünü, seyretmediği sezonu, dinlemediği müziği ne kadar “begeneceğinin” tahmin eder. Kaggle sitesi üzerinden yapılan unlu Netflix yarışmasının amacı buydu - ayrıca tahmin edilen ve gerçek beğeni notunun hata payının hesabı için RMSE hesabı kullanılmıştı.

Peki benzerliğin kriteri nedir, ve benzerlik nelerin arasında ölçülür?

Benzerlik, ürün seviyesinde, ya da kişi seviyesinde yapılabilir. Eğer ürün seviyesinde ise, tek bir ürün için tüm kullanıcıların verdiği nota bakılır. Eğer kullanıcı seviyesinde ise, tek kullanıcının tüm ürünlere verdiği beğeni notları vektörü kullanılır.

Mesela 1. sezondan hareketle, o sezonu begenen kişilere o sezona benzer diğer sezonlar tavsiye edilebilir. Kisiden hareketle, mesela John’a benzeyen diğer kişiler bulunarak onların begendigi ürünler John’a tavsiye edilebilir.

Ürün ya da kişi bazında olsun, benzerliği hesaplamanın da farklı yolları var. Genel olarak benzerlik ölçütünün 0 ile 1 arasında değişen bir sayı olmasını tercih ediyoruz ve tüm ayarları ona göre yapıyoruz. Diyelim ki elimizde beğeni notlarını taşıyan  $A, B$  vektörleri var (baska veri türü de taşıyor olabilir tabii), ve bu vektörlerin içinde beğeni notları var. Benzerlik çeşitleri şöyle:

### Oklit Benzerligi (Euclidian Similarity)

Bu benzerlik  $1/(1 + \text{mesafe})$  olarak hesaplanır. Mesafe karelerin toplamının karekoku (yani Oklitsel mesafe, ki isim buradan geliyor). Bu yuzden mesafe 0 ise (yani iki “sey” arasında 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 arasında bir deger dondurur, alttaki hesap onu normalize ederek 0 ile 1 arasina ceker.

### Kosinus Benzerligi (Cosine Similarity)

İki vektoru geometrik vektor olarak gorur ve bu vektorlerin arasında olusan aci (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 edegimiz 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[:, :].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,
```

```

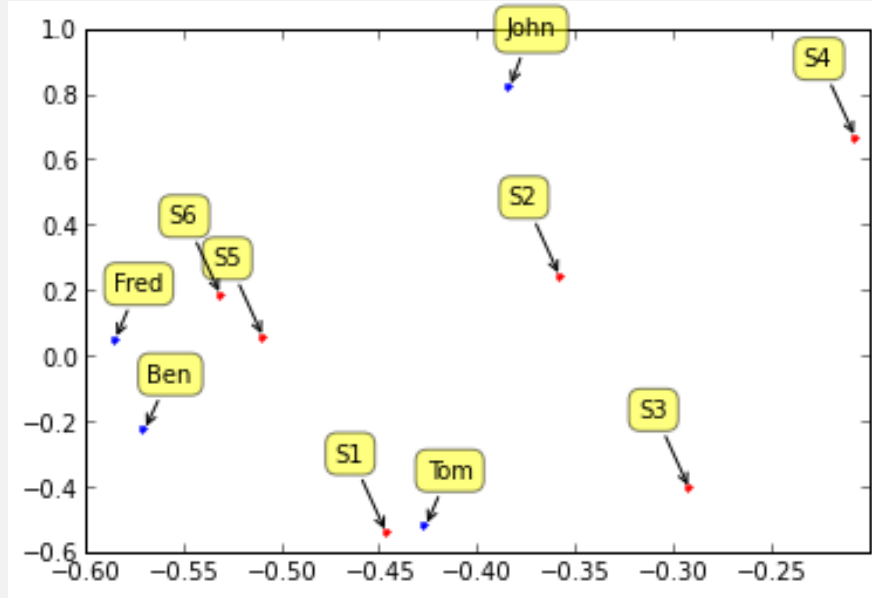
xy = (x, y), xytext = style,
textcoords = 'offset points', ha = 'right', va = 'bottom',
bbox = dict(boxstyle = 'round,pad=0.5', fc = 'yellow', alpha = 0.5),
arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=0'))

```

```

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 formulu 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 tarafın devrigini alalım

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

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

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

$S$  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 ustteki 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 kullanıcı Bob, en çok Ben ve Fred'e benziyor. Sonuca eristik! Artık bu iki kullanıcının yüksek not verdiği ama Bob'un hiç not vermediği sezonları alıp Bob'a tavsiye olarak sunabiliriz.

## 0.1 Movielens 1M Verisi

Bu veri seti 6000 kullanıcı tarafından yaklaşık 4000 tane filme verilen not / derece (rating) verisini içeriyor, 1 milyon tane not verilmiş, yani  $4000 * 6000 = 24$  milyon olasılık içinde sadece 1 milyon veri noktası dolu. Bu oldukça seyrek bir matris demektir.

Verinin ham hali diğer ders notlarını da içeren üst dizinlerde var, veriyi SVD ile kullanılır hale getirmek için movielens\_prep.py adlı script kullanılır. İşlem bitince movielens.csv adlı bir dosya script'te görülen yere yazılacak. Bu dosyada olmayan derecelendirmeler boş olacaktır. Bu boşlukları sıfırlarsak, scipy seyrek matrisi o noktaları atlayacaktır. Ardından bu seyrek matris üzerinde SVD işletilebilir.

```
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)
```

Birleştirici script kullanıcıların demografik bilgisini de matrise ekliyor. Biz üstte basitleştirme amaçlı bu kısmı hafızada tekrar çıkarttık, ama isteyen olursa bu ek ile de ilginç ek analizler yapılabilir.

Veriyle çalışma şeklimiz şöyle olacak: gerçek dünya şartlarını yakinen taklit edebilmek amacıyla rasgele bazı veri notları seçeceğiz, ve bu notları, nereden geldiğini bir kenara yazıp SVD'ye vermeden önce matristen sileceğiz.

```

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 kullanılacak, 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 kullanıcı ve film sorulacak. Algoritmamiza gore U matrisi icinde bu kullanıcıya

en yakın diğer kullanıcıyı bulacağız. Bunu daha hızlı bir şekilde yapabileceğiz çünkü SVD ile boyut azaltması yaptık, ve daha az kolonu baz alarak karşılaştırma yapabiliriz. Benzerlik ölçütü için yazının basındaki alternatiflere bakabilirsiniz, biz simdilik Oklit (euclid) benzerliği kullandık.

Kodlama açısından U'nin i'inci satırını diğer tüm U'ları ile yanyana getirip benzerlik hesabı yaptık, bu tür işler için map biçilmiş kaftan. Pythonic kodlama stili zaten daha az for döngüsü kullanmayı, daha fonksiyonel kodlama yapılmasını özendiriyor.

Bazı puf noktalar: benzerlikler ölçülürken i'inci veri kendisi ile yanyana gelir doğal olarak ve tabii ki “aynen” benzer sonucunu rapor eder, bunu engellemek için (çünkü tekrar i'inci kullanıcıyı değil, başka bir kullanıcıyı bulmak istiyoruz), i'nin kendisine olan mesafesini yapay olarak çok büyük negatif bir sayıya (-100) set ederiz.

Aynı şekilde benzerlik hesapları yapıldıktan sonra o hesapların üzerinden bir daha geçeriz ve j'inci ürüne oy vermemiş tüm kullanıcıların benzerlik hesabını yine yapay olarak -100'e set ederiz. Çünkü o kullanıcılarla da ilgilenmiyoruz. Aradığımız ürüne oy vermiş olan kullanıcılarla ilgileniyoruz.

O diğer kullanıcı bulunduğundan sonra o kullanıcının aradığımız verdiği ürüne atadığı dereceyi tahminimiz olarak kullanıyoruz. Altta 10 örnek üzerinde sonuçlar görülebilir.

```
__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 benzerliği en kötü yap
    sims[i] = -100
    # eğer kullanıcı o ürüne oy vermemişse, benzerliği yine kötüleştir
    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, 'kullanıcı', '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 kullanıcı için en yakın kişi 3704 ,urun 3219 için oyu 4.0 , gercek oy 4.0
6001
```

```
kullanıcı için en yakın kişi 4801 ,urun 1316 için 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>