

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 *  
  
d = np.array(  
    [[5, 5, 0, 5],  
     [5, 0, 3, 4],  
     [3, 4, 0, 3],  
     [0, 0, 5, 3],  
     [5, 4, 4, 5],  
     [5, 4, 5, 5]])  
  
data = DataFrame (d.T,  
    columns=['S1', 'S2', 'S3', 'S4', 'S5', 'S6'],  
    index=['Ben', 'Tom', 'John', 'Fred'])  
data
```

	S1	S2	S3	S4	S5	S6
Ben	5	5	3	0	5	5
Tom	5	0	4	0	4	4
John	0	3	0	5	4	5
Fred	5	4	3	3	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 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.ix['Fred'])
print np.array(data.ix['John'])
print np.array(data.ix['Ben'])
print pearson(data.ix['Fred'],data.ix['John'])
print pearson(data.ix['Fred'],data.ix['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.ix['Fred'],data.ix['John'])
print cos_sim(data.ix['Fred'],data.ix['Ben'])
```

```
0.898160909799
```

```
0.977064220183
```

Simdi tavsiye mekanigine geelim. 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. 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,

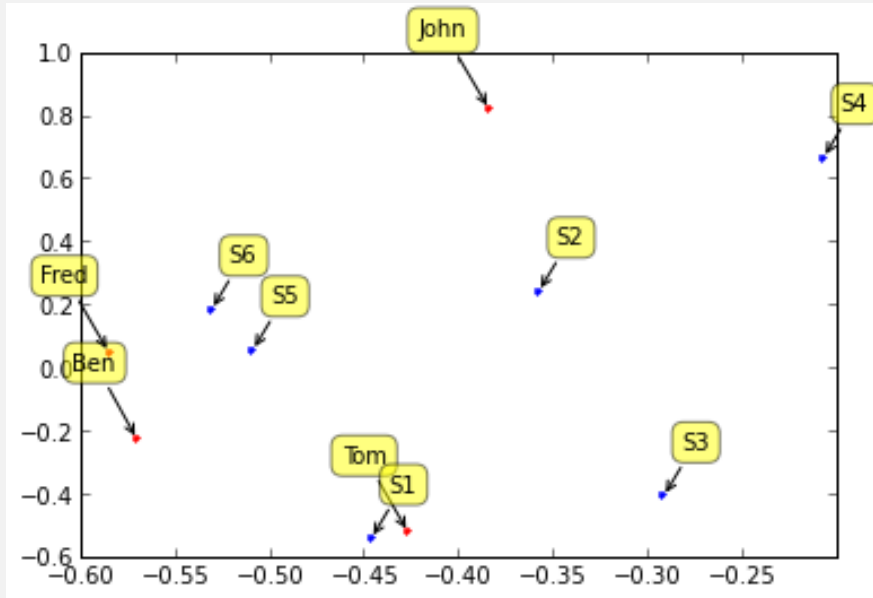
```
from numpy.linalg import linalg as la
U,Sigma,VT=la.svd(data)
print data.shape
print U.shape
u = U[:, :2]
v=VT[:, :].T
print u
print v
print u.shape, v.shape
```

```
(4, 6)
(4, 4)
[[-0.57098887 -0.22279713]
 [-0.4274751  -0.51723555]
 [-0.38459931  0.82462029]
 [-0.58593526  0.05319973]]
[[-0.44721867 -0.53728743]
 [-0.35861531  0.24605053]
 [-0.29246336 -0.40329582]
 [-0.20779151  0.67004393]
 [-0.50993331  0.05969518]
 [-0.53164501  0.18870999]]
(4, 2) (6, 2)
```

degerleri elimize geceri. 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,

$$U_k^{-1}A = U_k^{-1}USV_k$$

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

$$U_k^{-1}A = SV_k$$

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

Cok fazla ters alma islemi var, her iki tarafın 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 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,  4.83147593e-18],  
       [ 4.83147593e-18,  1.00000000e+00]])
```

Dogal olarak  $\dots e-17$  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

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

```
(6,)  
(4, 2)  
[-0.37752201 -0.08020351]
```

Ustte eye ve Sigma ile ufak bir takla attik, bunun sebebi svd cagrisindan gelen Sigma sonucunun bir vektor olması ama ustteki islem için kosegen bir “matrise” ihtiyacımız olması. Eger birim (identity) matrisini alıp 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(u):  
    print data.index[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 yuksek not verdigi ama Bob’un hic not vermedigi sezonlari 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 iceriyor, 1 milyon tane not verilmiş, yani  $4000 * 6000 = 24$  milyon olasilik içinde sadece 1 milyon veri noktası dolu. Bu oldukça seyrek bir matris demektir.

Verinin ham hali diger ders notlarını da iceren ust dizinlerde var, veriyi SVD ile kullanilir hale getirmek için movielens\_prep.py adli script kullanilir. Islem bitince movielens.csv adli bir dosya script’te gorulen yere yazilacak. Bu dosyada olmayan derecelendirmeler bos olacaktır . Bu bosluklari sifirlarsak, scipy seyrek matrisi o noktaları atlayacaktır. Ardından bu seyrek matris üzerinde SVD isletilebilir.

```
import pandas as pd, os  
import numpy as np  
import scipy.sparse as sps  
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 al  
df_train = df.ix[:5000,:]  
df_test = df.ix[5001:,:]  
df_train[np.isnan(df_train)] = 0.0  
movie_avg_rating = np.array(df_train.mean(axis=0))
```

```
df_train = df_train - movie_avg_rating
dfs_train = sps.coo_matrix(df_train)
```

```
df_train = np.array(df_train)
df_test = np.array(df_test)
```

```
print df_train.shape
print df_test.shape
```

```
(6040, 3731)
```

```
(5001, 3700)
```

```
(1039, 3700)
```

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

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

rmse = 0; n = 0
for i,test_row in enumerate(df_test):
    for j, test_val in enumerate(test_row):
        if np.isnan(test_val): continue
        curr = test_row.copy()
        curr[j] = np.nan
        curr[np.isnan(curr)] = 0.

        proj_row = np.dot(np.dot(la.inv(Sigma),VT),curr)

        sims = np.array(map(lambda x: euclid(x, proj_row), U[:,:__top_k__]))
        isim = np.argmax(sims)

        if df_train[isim, j]+movie_avg_rating[j] == 0: continue

        est = df_train[isim, j]+movie_avg_rating[j]
        real = df_test[i, j]

        print i, 'icin en yakin', isim, ',urun',j, 'icin oy', est, 'gercek', real
        rmse += (real-est)**2
        n += 1
        break
    if i == 20: break

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

0 için en yakın 1903 ,urun 144 için oy 5.0 gercek 5.0  
1

icin en yakın 239 ,urun 144 için oy 5.0 gercek 5.0  
2

icin en yakın 2045 ,urun 844 için oy 4.0 gercek 4.0  
3

icin en yakın 4636 ,urun 0 için oy 3.0 gercek 4.0  
4

icin en yakın 139 ,urun 845 için oy 4.0 gercek 5.0  
5

icin en yakın 427 ,urun 1107 için oy 4.0 gercek 5.0  
6

icin en yakın 3620 ,urun 31 için oy 4.0 gercek 4.0  
7

icin en yakın 1870 ,urun 0 için oy 4.0 gercek 3.0  
8

icin en yakın 4816 ,urun 106 için oy 5.0 gercek 5.0  
9

icin en yakın 3511 ,urun 0 için oy 3.0 gercek 4.0  
10

icin en yakın 3973 ,urun 1212 için oy 5.0 gercek 4.0  
11

icin en yakın 2554 ,urun 287 için oy 4.0 gercek 5.0  
12

icin en yakın 4733 ,urun 31 için oy 4.0 gercek 3.0  
13

icin en yakın 2339 ,urun 9 için oy 4.0 gercek 3.0  
14

icin en yakın 3036 ,urun 10 için oy 4.0 gercek 3.0  
15

icin en yakın 2748 ,urun 253 için oy 5.0 gercek 5.0  
16

icin en yakın 450 ,urun 16 için oy 4.0 gercek 4.0



17

icin en yakin 1133 ,urun 9 icin oy 5.0 gercek 2.0  
18

icin en yakin 3037 ,urun 253 icin oy 5.0 gercek 4.0  
19

icin en yakin 1266 ,urun 107 icin oy 3.0 gercek 3.0  
20

icin en yakin 537 ,urun 253 icin oy 5.0 gercek 5.0  
rmse 0.975900072949

#### 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>