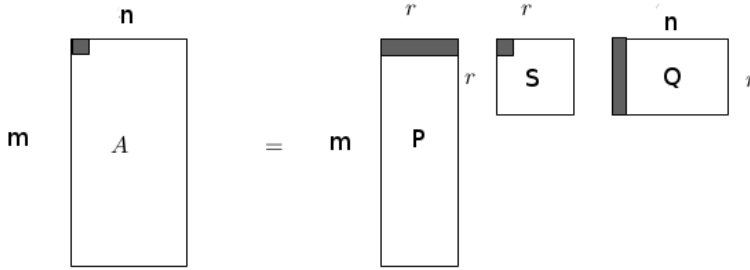


## Yaklasiksal SVD ile Tavsiye Sistemleri

Gecmis verilere bakarak bir kullanicinin hic seyretmedigi bir filme nasil not verecegini tahmin etmek unlu Netflix yarismasinin konusuydu. Onceki bir yazi *SVD, Toplu Tavsiye*'de benzer bir veri seti Movielens üzerinde SVD uygulayarak once boyut azaltmistik, azaltilmis boyut uzerinden yeni (o film icin notu bilinmeyen) bir kullanicinin diger mevcut kullanicilara mesafesini hesaplamis, ve boylece begeni acisindan en cok benzedigi diger kullaniciyi bulmustuk (birkac tane de bulunabilir). Bu kullanicinin bir film icin verdigi notu yeni kullanıcı için tahmin olarak baz almistik. SVD'yi kullanmanın bir yontemi daha var, Netflix yarismasinda kullanim [1] bir yaklasim soyle; alttaki SVD ayristirmasina bakalim,



1. kullanicini 1. filme verdigi not ustte koyu gosterilen satirlarin carpimi ile oluyor, eger ufak harfler ve kullanıcı (user) için  $u$ , film için  $i$  indisini, ve  $q, p$  vektorlerini  $Q, P$  matrislerinin sirasiyla kolon ve satirlarini gostermek için kullanirsak, ayristirma sonrasi begeni degeri (onemli bir kismi daha dogrusu)  $q_i^T p_u$  carpimindadir. Carpim icinde  $S$ 'ten gelecek tekil degeri (singular value) ne olacak? Simdi formulasayonda bir degisiklik dusunelim, bu degerin carpim disina alindigini hayal edelim; bu degerin kabaca birkac toplanin sonucuna donustugunu dusunelim. Matematiksel olarak imkansiz degil. Bu toplam terimleri, toplam olduklari için, bir baz seviyesi tanimlayabilirler. Bir kullanicinin ne kadar yanli (bias) not verdigini, ya da bir filmin nasil not almaya meyilli oldugunu modelleyebilirler (ki bu da bir yanlilik olcusu). Ayrica tum filmlere verilen notlari yanliliği da ayri bir terim olarak gosterilebilir. Tum bunlari bir araya koyarsak, bir begeni notunu tahmin edecek formül soyle gosterilebilir,

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$

$\mu$  bir skalar, tum filmlere verilen ortalamayi gosteriyor, ki tum begenilerin sayisal ortalamasi uzerinden basit bir sekilde hizla hesaplanabilir.  $\hat{r}_{ui}$ 'ya bir tahmin dedik cunku modelimizdeki vektorlerin degerlerini bulduktan sonra (egitim verisiyle bu hesabi yapacagiz) modeli kullanarak gercek not  $r_{ui}$  için bir tahmin olarak kullanılacak.

Yanlilik hakkında bazi ornekler vermek gerekirse, diyelim ki kullanıcı Bob not verirken yuksek seviyede oy vermeye meyilli. Bu durumda bu kullanicinin ortalama hatta dusuk oy vermesi onun bir filminden hakikaten hic hoslanmadigini sinyallebilir. Ya da bir film genellikle ortalama oy almaktadir, bu durumda

ona cok iyi not veren bir kisinin bu filmi cok begendigi ortaya cikir. Modeldeki yanlilik parametreleri bu durumu saptayabilirler. Eger verimizde / gercek dunyada yanlilik var ise, modelin bu bilgiyi kullanmasi onun basarisini arttiracaktir.

Egitim

Egitim icin ne yapmali? Minimize edecegimiz bir hedef fonksiyonu kuralim, ki cogunlukla bu karesi alinmis hata ile olur. Mesela gercek not  $r_{ui}$  degerinden tahmin notu  $\hat{r}_{ui}$ 'yi cikartip karesini alabiliriz. Bu islemi tum  $u, i$ 'ler icin yaparak sonuclari toplariz, ve bu toplami minimize etmeye ugrasabiliriz. Yani

$$\begin{aligned} & \min_{b^*, q^*, p^*} \sum_{u,i} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \\ &= \min_{b^*, q^*, p^*} \sum_{u,i} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \end{aligned}$$

Kisaltma olarak  $e_{ui}$  tanımlayalım, bu faydali olabilir, formüldeki ilk parantez içindeki kisimda kullanmak uzere,

$$e_{ui} := r_{ui} - \hat{r}_{ui}$$

$\lambda$  ile carpilan bolum regularizasyon icin. Istatistik, yapay ogrenimde modelimizin asiri uygunluk (overfitting) yapmasini engellemek icin regularizasyon kullanilir, bunun icin istedigimiz degiskenlerin fazla buyumesini cezalandiririz, ustteki minimizasyon modelinde bu ceza icin tum degerlerin buyuklugunu (magnitude) hesapladik -skalar degerlerin karesini, vektor degerlerinin kare norm'unu alarak- ve bu buyuklukleri bizim disaridan tanımlayacagimiz bir sabitle carpimi uzereinden minimizasyon problemine direk dahil ettik. Boylece bu buyuklukler azaltılma hedefinin parcasi haline geldiler. Yani hem  $e_{ui}^2$  hem de hatayi olusturan degerlerin kendileri minimize edilecek. Bu minimizasyon sirasinda bazi degiskenlerin sifira inip o  $u, i$  icin tamamen etkisiz hale gelmesi bile mumkundur (ki bu bize o parametrenin onemsiz oldugunu sinyalledebilecegi icin faydalidir).

Rasgele Gradyan Inisi (Stochastic Gradient Descent -SGD-)

Modeli nasil minimize ederiz? Bu model konveks (convex) degil, ki konvekslik bilindigi gibi fonksiyonun duzgün bir cukur gibi oldugu problemlerdir. Boyle cukur fonksiyonlarında herhangi bir noktadan baslarsiniz, gradyani hesaplarsiniz, ve bu gradyan hep optimal inis noktasini (daha dogrusu tersini) gosterir, ve yolda giderken takilip kalabileceginiz yerel minimumlar mevcut degildir, ve sonunda cukur dibine ulasilir. Bizim problemimizde  $q_i^T p_u$  var, bu degiskenlerin ikisi de bilinmiyor, ve bu carpimin karesi alindigi icin genel karesellligi (quadratic) kaybetmis oluyoruz. Fakat yine de SGD bu problemi cozebiliyor. Bunun sebeplerini, SGD SVD'nin hikayesiyle beraber yazinin sonunda bulabilirsiniz.

SGD icin gradyanlar lazim, her degisken icin minimizasyon toplami icindeki

kismin (bu kısma E diyelim) ayrı ayrı kısmi turevini almak lazım. Mesela  $b_u$  için

$$\frac{\partial E}{\partial b_u} = -2e_{ui} + 2\lambda b_u$$

Gradyan her zaman en yüksek çıkışı gösterir, o zaman hesapsal algoritma onun tersi yönüne gitmelidir. Bu gidisin adım büyüklüğünü kontrol etmek için disaridan bizim belirledigimiz bir  $\gamma$  sabiti ile carpim yapabiliriz, ve bir numara daha, sabit 2 degerlerinin  $\gamma$  icinde eritilebilecegini farzederek onlari sileriz. Yani adım  $\gamma(e_{ui} - \lambda b_u)$  haline geldi. Bir dongu icinde eski  $b_u$  bulunacak, gordugumuz yonde adım atilacak, yani adım onceki degere toplanacak, ve yeni deger elde edilecek. Diger degiskenler icin turev alip benzer islemleri yaparsak, sonuc soyle,

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda \cdot b_u)$$

$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda \cdot b_i)$$

$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda \cdot q_i)$$

$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda \cdot p_u)$$

Her degisken icin baslangic noktasi rasgele olarak secilebilir, ya da en iyi degerler deneme/ yanilma ya da capraz saglama (crossvalidation) ile bulunabilir. Ayni durum  $\gamma, \lambda$  sabitleri icin de gecerli. Biz bu ornekte deneme / yanilma yontemini sectik.

Rasgelelik, aynen *Lojistik Regresyon* orneginde oldugu gibi verinin rasgeliliginden geliyor, her veri noktasini teker teker sirayla isliyoruz aslinda fakat bu “siranin” rasgele oldugunu farzettigimiz icin ozyineli algoritmamiz rasgelelik elde ediyor. Python kodu altta, egitim icin kod sadece bir kere verinin uzerinden geciyor. Basa donup birkac kere (hatta yuzlerce) veriyi isleyenler de olabiliyor.

```
from numpy.linalg import linalg as la
import numpy as np
import random
import pandas as pd, os

def create_training_test(df, collim=2, rowlim=200):
    test_data = []
    df_train = df.copy()
    for u in range(df.shape[0]):
        row = df.ix[u]; idxs = row.index[row.notnull()]
        if len(idxs) > collim:
            i = random.choice(idxs); val = df.ix[u,i]
```

```

        test_data.append([u,i,val])
        df_train.ix[u,i] = np.nan
    if len(test_data) > rowlim: break
    return df_train, test_data

def ssvd(df_train,rank):
    print 'rank',rank
    lam = 0.02 # regularizasyon
    gamma = 0.001 # adim katsayisi
    m,n = df_train.shape
    c = 0.03
    b_u = np.ones(m) * c
    b_i = np.ones(n) * c
    p_u = np.ones((m, rank)) * c
    q_i = np.ones((rank, n)) * c
    r_ui = np.array(df_train)
    for u in range(m):
        row = df_train.ix[u]; idxs = row.index[row.notnull()]
        for i in idxs:
            i = int(i)
            r_ui_hat = np.dot(q_i[:,i].T,p_u[u,:])
            e_ui = r_ui[u,i] - r_ui_hat
            q_i[:,i] = q_i[:,i] + gamma * (e_ui*p_u[u,:].T - lam*q_i[:,i])
            p_u[u,:] = p_u[u,:] + gamma * (e_ui*q_i[:,i].T - lam*p_u[u,:])
    return q_i,p_u

```

Kodun önemli bir özelliği sudur, boş yani `nan` değeri içeren notlar eğitim sırasında atlanır. SGD seyrek verilerle de işleyebilen bir eğitim yöntemidir. Bu durumda verinin seyrekliği (sparsity) bizim için çok faydalı, çünkü o veri noktalarına bakılmayacak, `row.notnull()` ile boş olmayan öğelerin indis değerlerini alıyoruz. Bilindiği üzere Movielens, Netflix verileri oldukça seyrek, kullanıcı binlerce film içinden onlar, yüzler bağlamında not verimi yapar, geri kalan değerler boştur.

### Pratikte Atlanabilecek Formüller

Ustteki formülasyon içinde genel, film ve kullanıcı yanlılıkları formül içinde belirtilmiştir. Pratikte sayısal hesap açısından bu yanlılıkları daha SVD başlamadan önce veriden çıkartmak, ve yanlılıkları rasgele güncelleme mantığından çıkartmak daha iyi sonuç veriyor. Bu çıkartma şöyle yapılır; önce film ortalamaları hesaplanır, her kullanıcı için o kullanıcının o film ortalamasına olan farkı (offset) hesaplanır ve bu fark eğitim sırasında not olarak kullanılır. Böylece SVD tüm yanlılıklardan arınmış bir fark hesabi ile çalışır sadece, ve numerik olarak bu daha avantajlıdır. Yoksa ilk atılan adımda fark çok büyük olacağı için bu fark gradyan inisinin yönünü ağırlikli olarak belirleyecektir, ve bu ilk atılan adım hesabın geri kalanına asiri baskın çıkabilecektir (dominate). Zaten SVD'nin kuzeni olan PCA'den de bildiğimiz gibi, bu yöntem ortalamadan arındırılmış (demeaned) farklarla iş yapıyor.

### Niye Azar Azar (Incremental) Hesap?

Madem  $U, S, V$  hesabi yapar gibi  $q_i, p_u$  hesapliyoruz, niye bu hesabi adim adim yapıyoruz?

Bu sorunun cevabi azar azar hesap yaparken hafizaya sadece o satiri alarak daha az bellek kullanmamiz. Milyarlarca boyut ve milyarlarca satir isliyor olsaydik bu matrisin tamamini hafizaya almamiz mumkun olmazdi.

Basit bir ornek

```
import pandas as pd
import ssvd
d = np.array(
[[ 5., 5., 3., nan, 5., 5.],
 [ 5., nan, 4., nan, 4., 4.],
 [ nan, 3., nan, 5., 4., 5.],
 [ 5., 4., 3., 3., 5., 5.],
 [ 5., 5., nan, nan, nan, 5.]
])
data = pd.DataFrame(d, columns=['0', '1', '2', '3', '4', '5'],
                    index=['Ben', 'Tom', 'John', 'Fred', 'Bob'])
avg_movies_data = data.mean(axis=0)
data_user_offset = data.apply(lambda x: x-avg_movies_data, axis=1)
q_i, p_u = ssvd.ssvd(data_user_offset, rank=3)
print 'q_i', q_i
print 'p_u', p_u
u = 4; i = 2 # Bob icin tahmin yapalim
r_ui_hat = np.dot(q_i[:,i].T, p_u[u,:]) + avg_movies_data.ix[i] + \
            np.nan_to_num(data_user_offset.ix[u,i])
print r_ui_hat

rank 3
q_i [[ 0.02999728  0.02999728  0.02999795  0.02999862  0.02999725  0.02999659]
 [ 0.02999728  0.02999728  0.02999795  0.02999862  0.02999725  0.02999659]
 [ 0.02999728  0.02999728  0.02999795  0.02999862  0.02999725  0.02999659]]
p_u [[ 0.03003012  0.03003012  0.03003012]
 [ 0.0299783  0.0299783  0.0299783 ]
 [ 0.02998083  0.02998083  0.02998083]
 [ 0.02996942  0.02996942  0.02996942]
 [ 0.03002645  0.03002645  0.03002645]]
3.33603552974
```

Test Etmek

Test verisi olusturmak icin egitim verisinde rasgele olarak bazi notlari sectik, bunlari bir kenara kaydederek onlari ana matrisindeki degerini sildik (yerine nan koyarak), ve bir kismi silinmis yeni bir egitim matrisi yarattik, `create_training_test` islevinde bu gorulebilir. Bu islevde her kullanicidan sadece bir tane not verisi aliyoruz, ve bunu sadece belli bir sayida, `collim` kadar, not vermis kullanicilar icin yapıyoruz, ki boylece az sayida not vermis kullanicilarin verisini azaltmamis oluyoruz. Ayrica belli miktarda, `rowlim` kadar test noktası elde edince is bitti kabul ediyoruz. Test verisi yaratmak icin %80-%20 gibi bir ayrim yapmadik, yani egitim verisindeki tum kullanicilari ve onlari neredeyse tum verisini egitim icin kullaniyoruz.

Movielens verisine gelelim. *SVD, Toplu Tavsiye* yazisindeki `movielens_prep.py` ile gerekli eğitim dosyası üretildiğini farzederek,

```
import pandas as pd, os
df = pd.read_csv("%s/Downloads/movielens.csv" % os.environ['HOME'], sep=';')
print df.shape
df = df.ix[:,1:3700] # id kolonunu atla,
df.columns = range(3699) # kolon degerlerini tekrar indisle
print df.shape

(6040, 3731)
(6040, 3699)

avg_movies = df.mean(axis=0)
df_user_offset = df.apply(lambda x: x-avg_movies, axis=1)
print df.ix[6,5], avg_movies.ix[5], df_user_offset.ix[6,5]

4.0 3.87872340426 0.121276595745
```

Eğitim ve test verisi yaratıyoruz,

```
import ssvd
df_train, test_data = ssvd.create_training_test(df_user_offset, rowlim=500, collim=300)
print len(test_data)

501

q_i, p_u = ssvd.ssvd(df_train, rank=25)

rank 25
```

Test

```
rmse = 0; n = 0
for u,i,real in test_data:
    r_ui_hat = np.dot(q_i[:,i].T,p_u[u,:])
    rmse += (real-r_ui_hat)**2
    n += 1
print "rmse", np.sqrt(rmse / n)

rmse 0.944142356597
```

Sonuc oldukça iyi.

Formulasyonun Hikayesi

SGD SVD'nin hikayesi şöyle. Yıl 2009, Netflix Yarışması [11] katılımcılarından Simon Funk (gerçek adı Brandyn Webb) SGD SVD yaklaşımını kodlayıp veri üzerinde işletince birden bire sıralamada ilk 3'e fırlar; Webb artık çok unlu olan blog yazısında [1] yaklaşımı detayıyla paylaşıp forum'da haberini verince bu haber tam bir bomba etkisi yaratır. Pek çok kişi yaklaşımı kopyalar, hatta kazanan BellKor ürününde Webb'in SVD yaklaşımının kullanıldığı biliniyor.

Bu metotun kesfi hangi basamaklardan geçti? Beni meraklandiran minimizasyon formulasyonun konveks olmamasıydı – genellikle optimizasyon problemlerinde

konveksliğin mevcudiyeti aranır, çünkü bu durumda sonuca yaklaştırmak (convergence) için bir garanti elde edilir. Bu durumda konvekslik yoktu. “O zaman Webb nasıl rahat bir şekilde SGD kullanabildi?” sorusunun cevabını merak ediyorduk yani. Biraz araştırınca Bottou ve LeCun gibi araştırmacıların yazılarına ulaştık [4]. Onlara göre konvekslik olmaması yapay öğrenim araştırmacılarını korkutmamalı, eğer sayısal (empirically) işleyen bir algoritma var ise, teorik ispat gelene kadar bu metotun kullanılmasında sakınca yoktur.

Fakat böyle buluşlarda yine de bazı garantiler temel alınmış olabilir, araştırmacı tamamen balıklama atlayış yapmaz. Webb’in kendisine bu soruları sorduk ve bize buluşun hangi seviyelerden geçtiğini anlattı. Geriye sarıyoruz, Webb Netflix’den çok önce yapay sinir ağlarını araştırmaktadır, ve Sanger, Oja’nın [5,6] yayınlarını baz alarak kurduğu bir YSA için bir çözüm bulduğunu farkeder. Sayısal çözümde özdeğer/vektor bulmaya yarayan Ustel Metotun (power method) bir şeklini kullanmıştır, ki Sanger’in Genel Hebbian Algoritmasının (GHA) ustel metot ile bağlantıları var, ve bu GHA yayınında “egitilince” özdeğer/vektor ve PCA hesabi yapabilen bir YSA’dan bahsediliyor. Daha önemlisi GHA 1 olasılıkla (yani kesin) bu sonuçlara erişebiliyor.

Daha sonra Webb bu çözümü arkadaşları Gorrell ile tartışırken Gorrell ona problem formülasyonunun SVD olarak görülebileceğini söyler. Bilindiği gibi özdeğer/vektor hesabi ile SVD yakın akraba sayılır. İkili bu bağlamda birkaç yayın da yaparlar. Daha sonra Netflix yarışması başladığında Webb çözüm için gradyan baz alarak SGD kullanılabileceğini fark ediyor, ki SGD ile ustel metot arasında teorik bağlantı var [7]. Ve sonuç olarak SGD SVD metodu ortaya çıkıyor.

Tabii ki “SGD SVD ne kadar SVD sayılır?” gibi bir soru sorulabilir. Evet, regularizasyon bazı gayri-lineerlikleri probleme sokar, zaten bu çözümü “yaklaşıksal” yapan kısım da budur. Fakat belli şartlarda, regularizasyon olmasa çözüm tam SVD olacaktır. Bu buluşun puf noktası bu bilgide, ve üstteki teorik benzerliklerde, onları biliyor olmakta yatıyor. Eğer bunlar biliniyor ise, ve sağlam lineer cebir bilgisi ile gerektiği zaman onları ne kadar esnetebileceğimizi biliriz. Konu hakkındaki daha fazla detay [10]’da bulunabilir.

## Tavsiye

```
movies = pd.read_csv("../stat/stat_pandas_ratings/movies.dat", \
    sep='::', names=['idx', 'movie', 'type'])
def eval_user(u):
    res = {}
    for i in range(df_user_offset.shape[1]):
        r_ui_hat = np.dot(q_i[:,i].T, p_u[u,:])
        res[i] = float(r_ui_hat) + avg_movies.ix[i] + df_user_offset.ix[u,i]
    res = sorted(res.items(), key=lambda x:x[1], reverse=True)

    print "\n\nTavsiyeler\n\n"
    for j, (si, sm) in enumerate(res):
        print movies[movies['idx'] == si]['movie']
        if j == 10: break
```

```

print "\n\nMevcut Filmler\n\n"
row = df.ix[u]
idxs = row.index[row.notnull()]
for j,idx in enumerate(idxs):
    print movies[movies['idx'] == idx]['movie'], row[idx]
    if j == 10: break

eval_user(300)
eval_user(2900)

```

#### Tavsiyeler

```

3441    Frequency (2000)
Name: movie, dtype: object
105    Muppet Treasure Island (1996)
Name: movie, dtype: object
Series([], dtype: object)
2    Grumpier Old Men (1995)
Name: movie, dtype: object
4    Father of the Bride Part II (1995)
Name: movie, dtype: object
1    Jumanji (1995)
Name: movie, dtype: object
3    Waiting to Exhale (1995)
Name: movie, dtype: object
0    Toy Story (1995)
Name: movie, dtype: object
5    Heat (1995)
Name: movie, dtype: object
6    Sabrina (1995)
Name: movie, dtype: object
7    Tom and Huck (1995)
Name: movie, dtype: object

```

#### Mevcut Filmler

```

Series([], dtype: object) 5.0
0    Toy Story (1995)
Name: movie, dtype: object 3.0
1    Jumanji (1995)
Name: movie, dtype: object 4.0
3    Waiting to Exhale (1995)
Name: movie, dtype: object 3.0
4    Father of the Bride Part II (1995)
Name: movie, dtype: object 5.0
9    GoldenEye (1995)
Name: movie, dtype: object 4.0
15   Casino (1995)
Name: movie, dtype: object 4.0
19   Money Train (1995)
Name: movie, dtype: object 5.0

```



```
20    Get Shorty (1995)
Name: movie, dtype: object 3.0
23    Powder (1995)
Name: movie, dtype: object 4.0
30    Dangerous Minds (1995)
Name: movie, dtype: object 5.0
```

#### Tavsiyeler

```
Series([], dtype: object)
0    Toy Story (1995)
Name: movie, dtype: object
1    Jumanji (1995)
Name: movie, dtype: object
2    Grumpier Old Men (1995)
Name: movie, dtype: object
3    Waiting to Exhale (1995)
Name: movie, dtype: object
4    Father of the Bride Part II (1995)
Name: movie, dtype: object
5    Heat (1995)
Name: movie, dtype: object
6    Sabrina (1995)
Name: movie, dtype: object
7    Tom and Huck (1995)
Name: movie, dtype: object
8    Sudden Death (1995)
Name: movie, dtype: object
9    GoldenEye (1995)
Name: movie, dtype: object
```

#### Mevcut Filmler

```
19    Money Train (1995)
Name: movie, dtype: object 4.0
23    Powder (1995)
Name: movie, dtype: object 4.0
37    It Takes Two (1995)
Name: movie, dtype: object 3.0
45    How to Make an American Quilt (1995)
Name: movie, dtype: object 5.0
48    When Night Is Falling (1995)
Name: movie, dtype: object 5.0
66    Two Bits (1995)
Name: movie, dtype: object 5.0
78    Juror, The (1996)
Name: movie, dtype: object 4.0
104   Nobody Loves Me (Keiner liebt mich) (1994)
Name: movie, dtype: object 5.0
118   Race the Sun (1996)
Name: movie, dtype: object 4.0
```

134 From the Journals of Jean Seberg (1995)  
Name: movie, dtype: object 2.0  
142 Brothers McMullen, The (1995)  
Name: movie, dtype: object 3.0

## Kaynaklar

- [1] <http://sifter.org/~simon/journal/20061211.html>
- [2] Koren, Bell, *Recommender Systems Handbook*, [http://www.cs.bme.hu/nagyadat/Recommender\\_systems\\_handbook.pdf](http://www.cs.bme.hu/nagyadat/Recommender_systems_handbook.pdf)
- [3] <http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf>
- [4] <http://www.cs.nyu.edu/~yann/talks/lecun-20071207-nonconvex.pdf>
- [5] [http://courses.cs.washington.edu/courses/cse528/09sp/sanger\\_pca\\_nn.pdf](http://courses.cs.washington.edu/courses/cse528/09sp/sanger_pca_nn.pdf)
- [6] <http://users.ics.aalto.fi/oja/Oja1982.pdf>
- [7] <http://arxiv.org/pdf/1308.3509>
- [8] [http://www.maths.qmul.ac.uk/~wj/MTH5110/notes/MAS235\\_lecturenotes1.pdf](http://www.maths.qmul.ac.uk/~wj/MTH5110/notes/MAS235_lecturenotes1.pdf)
- [9] <http://heim.ifi.uio.no/~tom/powerandqrslides.pdf>
- [10] <http://math.stackexchange.com/questions/649701/gradient-descent-on-non-convex-function-works-but-how>
- [11] Netflix Odulu, <http://www.netflixprize.com>