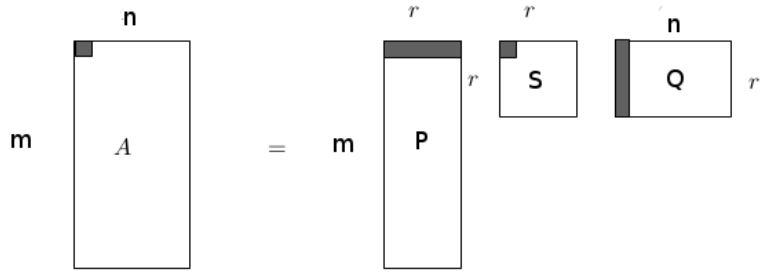
Yaklasiksal SVD ile Tavsiye Sistemleri

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 uzerinde 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 kullanici icin tahmin olarak baz almistik. SVD'yi kullanmanin bir yontemi daha var, Netflix yarismasinda kullanilan [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 kullanici (user) icin u, film icin i indisini, ve q, p vektorlerini Q, P matrislerinin sirasiyla kolon ve satirlarini gostermek icin 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 formulasyonda bir degisiklik dusunelim, bu degerin carpim disina alindigini hayal edelim; bu degerin kabaca birkac toplamin sonucuna donustugunu dusunelim. Matematiksel olarak imkansiz degil. Bu toplam terimleri, toplam olduklari icin, 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 notlarin yanliligi da ayri bir terim olarak gosterilebilir. Tum bunlari bir araya koyarsak, bir begeni notunu tahmin edecek formul soyle gosterilebilir,

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

μ 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} icin bir tahmin olarak kullanilacak.

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

ona cok iyi not veren bir kisinin bu filmi cok begendigi ortaya cikar. 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

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

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

Kisaltma olarak e_{ui} tanimlayalim, bu faydali olabilir, formuldeki ilk parantez icindeki kisimda kullanmak uzere,

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

λ 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 tanimlayacagimiz bir sabitle carpimi uzerinden minimizasyon problemine direk dahil ettik. Boylece bu buyuklukler azaltilma 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 sinyalleyebilecegi icin faydalidir).

Rasgele Gradyan Inisi (Stochastic Gradient Descent -SGD-)

Modeli nasil minimize ederiz? Bu model konveks (convex) degil, ki konvekslik bilindigi gibi fonksiyonun duzgun bir cukur gibi oldugu problemlerdir. Boyle cukur fonksiyonlarinda 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 kisma E diyelim) ayri ayri kismi turevini almak lazim. Mesela b_u icin

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

Gradyan her zaman en yuksek cikisi gosterir, o zaman hesapsal algoritma onun tersi yonune gitmelidir. Bu gidisin adim buyuklugunu kontrol etmek icin disaridan bizim belirledigimiz bir γ sabiti ile carpim yapabiliriz, ve bir numara daha, sabit 2 degerlerinin γ icinde eritilebilecegini farzederek onlari sileriz. Yani adim $\gamma(e_{ui} - \lambda b_u)$ haline geldi. Bir dongu icinde eski b_u bulunacak, gordugumuz yonde adim atilacak, yani adim 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, hatta secilmelidir; Internet'te bu konu hakkinda "efendim tum degerleri 0.1 degeri yapsaniz olur" gibi yorumlar okuyabilirsiniz, bu durumda q_i, p_u degiskenlerinin tum hucreleri ayni degerde kalir! Yani q_3 mesela, tamamen 0.6 degerine sahip olur, ki bu istenen bir sey olmaz. γ, λ sabitleri icin en iyi degerler deneme/ yanilma ya da capraz saglama (crossvalidation) ile bulunabilir, 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, pandas as pd

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,k):
    lam = 0.02 # regularizasyon
    gamma = 0.01 # adim katsayisi
    m,n = df_train.shape
    b_u = np.random.uniform(0, 0.1, size=m)
    b_i = np.random.uniform(0, 0.1, size=n)
    p_u = np.random.rand(m,k)
    q_i = np.random.rand(k, n)
    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 onemli bir ozelligi sudur, bos yani `nan` degeri iceren notlar egitim sirasinda atlanir. SGD seyrek verilerle de isleyebilen bir egitim yontemidir. Bu durumda verinin seyrekligi (sparsity) bizim icin cok faydali, cunku o veri noktalarina bakilmayacak, `row.notnull()` ile bos olmayan ogelerin indis degerlerini aliyoruz. Bilindigi uzere Movielens, Netflix verileri oldukca seyrektir, kullanici binlerce film icinden onlar, yuzler baglaminda not verimi yapar, geri kalan degerler bostur.

Pratikte Atlanabilecek Formuller

Ustteki formulasyon icinde genel, film ve kullanici yanliliklari formul icinde belirtilmistir. Pratikte sayisal hesap acisindan bu yanliliklari daha SVD baslamadan once veriden cikartmak, ve yanlililiklari rasgele guncelleme mantigindan cikartmak daha iyi sonuc veriyor. Bu cikartma soyle yapilir; once film ortalamalari hesaplanir, her kullanici icin o kullanicinin o film ortalamasina olan farki (offset) hesaplanir ve bu fark egitim sirasinda not olarak kullanilir. Boylece SVD tum yanliliklardan arinmis bir fark hesabi ile calisir sadece, ve numerik olarak bu daha avantajlidir. Yoksa ilk atilan adimda fark cok buyuk olacagi icin bu fark gradyan inisinin yonununu agirlikli olarak belirleyecektir, ve bu ilk atilan adim hesabin geri kalanina asiri baskin cikabilecektir (dominate). Zaten SVD'nin kuzeni olan PCA'den de bildigimiz gibi, bu yontem ortalamadan arindirilmis (demeaned) farklarla is yapiyor.

Niye Azar Azar (Incremental) Hesap?

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

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,k=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
```

```
q_i [[ 0.77564986  0.1768796   0.56224297  0.42870962  0.94723452  0.55772707]
 [ 0.74100171  0.18146607  0.07704434  0.17631144  0.48868432  0.90470742]
 [ 0.70541948  0.28729857  0.091619    0.46184649  0.64498754  0.27410096]]
p_u [[ 0.41549541  0.59784095  0.45227968]
 [ 0.7756159   0.1197948   0.2921504 ]
 [ 0.82274016  0.58432802  0.46501307]
 [ 0.1877425   0.21364635  0.1924456 ]
 [ 0.80554597  0.59254732  0.9775562 ]]
3.92146103113
```

Test Etmek

Test verisi olusturmak icin egitim verisinde rasgele olarak bazi notlari sectik, bunlari bir kenara kaydederek onlarin ana matris icindeki 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 yapiyoruz, ki boylece az sayida not vermis kullanicilarin verisini azaltmamis oluyoruz. Ayrica belli miktarda, `rowlim` kadar test noktasi elde edince is bitti kabul ediyoruz. Test verisi yaratmak icin %80-%20 gibi bir ayrim yapmadik, yani egitim verisindeki tum kullanicilari ve onlarin neredeyse tum verisini egitim icin kullaniyoruz.

Movielens verisine gelelim. *SVD, Toplu Tavsiye* yazisindaki `movielens_prep.py`

ile gerekli egitim dosyasi uretildigini 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
```

Egitim ve test verisi yaratiyoruz,

```
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,k=8)
```

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 0.936136196897
```

Sonuc fena degil.

Formulasyonun Hikayesi

SGD SVD'nin hikayesi soyle. Yil 2009, Netflix Yarismasi [11] katilimcilarindan Simon Funk (gercek adi Brandyn Webb) SGD SVD yaklasimini kodlayip veri uzerinde isletince birden bire siralamada ilk 3'e firlar; Webb artik cok unlu olan blog yazisinda [1] yaklasimi detayiyla paylasip forum'da haberini verince bu haber tam bir bomba etkisi yaratir. Pek cok kisi yaklasimi kopyalar, hatta kazanan BellKor urununde Webb'in SVD yaklasiminin kullanildigi biliniyor.

Bu metotun kesfi hangi basamaklardan gecti? Beni meraklandiran minimizasyon formulasyonun konveks olmamasiydi – genellikle optimizasyon problemlerinde

konveksligin mevdudiyeti aranir, cunku bu durumda sonuca yaklasmak (convergence) icin bir garanti elde edilir. Bu durumda konvekslik yoktu. "O zaman Webb nasil rahat bir sekilde SGD kullanabildi?" sorusunun cevabini merak ediyorduk yani. Biraz arastirinca Bottou ve LeCunn gibi arastirmacilarin yazilarina ulastik [4]. Onlara gore konvekslik olmamasi yapay ogrenim arastirmacilarini korkutmamali, eger sayisal (empirically) isleyen bir algoritma var ise, teorik ispat gelene kadar bu metotun kullanilmasinda sakinca yoktur.

Fakat boyle buluslarda yine de bazi garantiler temel alinmis olabilir, arastirmaci tamamen baliklama atlayis yapmaz. Webb'in kendisine bu sorulari sorduk ve bize bulusun hangi seviyelerden gectigini anlatti. Geriye sariyoruz, Webb Netflix'den cok once yapay sinir aglarini arastirmaktadir, ve Sanger, Oja'nin [5,6] yayinlarini baz alarak kurdugu bir YSA icin bir cozum buldugunu farkeder. Sayisal cozumde ozdeger/vektor bulmaya yarayan Ustel Metotun (power method) bir seklini kullanmistir, ki Sanger'in Genel Hebbian Algoritmasinin (GHA) ustel metot ile baglantilari var, ve bu GHA yayininda "egitilince" ozdeger/vektor ve PCA hesabi yapabilen bir YSA'dan bahsediliyor. Daha onemlisi GHA 1 olasilikla (yani kesin) bu sonuclara erisebiliyor.

Daha sonra Webb bu cozumu arkadasi Gorrell ile tartisirken Gorrell ona problem formulasyonunun SVD olarak gorulebilecegini soyler. Bilindigi gibi ozdeger/vektor hesabi ile SVD yakin akraba sayilir. Ikili bu baglamda birkac yayin da yaparlar. Daha sonra Netflix yarismasi basladiginda Webb cozum icin gradyan baz alarak SGD kullanabilecegini farkediyor, ki SGD ile ustel metot arasinda teorik baglanti var [7]. Ve sonuc olarak SGD SVD metotu ortaya cikiyor.

Tabii ki "SGD SVD ne kadar SVD sayilir?" gibi bir soru sorulabilir. Evet, regularizasyon bazi gayri-lineerlikleri probleme sokar, zaten bu cozumu "yaklasiksal" yapan kisim da budur. Fakat belli sartlarda, regularizasyon olmasa cozum tam SVD olacaktir. Bu bulusun puf noktasi bu bilgide, ve ustteki teorik benzerliklerde, onlari biliyor olmakta yatiyor. Eger bunlar biliniyor ise, ve saglam lineer cebir bilgisi ile gerektigi zaman onlari ne kadar esnetebilecegimizi biliriz. Konu hakkindaki 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

[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

[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

[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