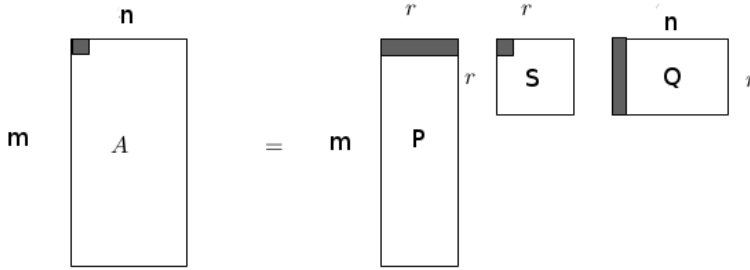


## Yaklasiksal SVD ile Tavsiye Sistemleri

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 kullanılan [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 formularyonda 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 film den 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 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

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

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 kisma 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 dışarıdan bizim belirledigimiz bir  $\gamma$  sabiti ile carpım yapabiliriz, ve bir numara daha, sabit 2 değerlerinin  $\gamma$  içinde eritilebileceğini farzederek onları sileriz. Yani adım  $\gamma(e_{ui} - \lambda b_u)$  haline geldi. Bir dongu içinde eski  $b_u$  bulunacak, gordugumuz yonde adım atılacak, yani adım önceki değere toplanacak, ve yeni değer elde edilecek. Diğer degiskenler için turev alıp benzer işlemleri yaparsak, sonuç şöyle,

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

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

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

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

Her degisken için başlangıç noktası rasgele olarak seçilebilir, hatta seçilmelidir; İnternet’te bu konu hakkında “efendim tüm değerleri 0.1 değeri yapsanız olur” gibi yorumlar okuyabilirsiniz, bu durumda  $q_i, p_u$  degiskenlerinin tüm hücreleri aynı değerde kalır! Yani  $q_3$  mesela, tamamen 0.6 değerine sahip olur, ki bu istenen bir şey olmaz.  $\gamma, \lambda$  sabitleri için en iyi değerler deneme / yanılma ya da capraz saglama (crossvalidation) ile bulunabilir, biz bu örnekte deneme / yanılma yöntemini seçtik.

Rasgelelik, aynen *Lojistik Regresyon* örneğinde olduğu gibi verinin rasgeleliğinden geliyor, her veri noktasını teker teker sırayla işliyoruz aslında fakat bu “siranın” rasgele olduğunu farzettigimiz için ozyineli algoritmamız rasgelelik elde ediyor. Python kodu alta, eğitim için kod sadece bir kere verinin üzerinden geçiyor. Basa donup birkaç kere (hatta yüzlerce) veriyi işleyenler 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 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 (de-meaned) 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.

Ayrica olmayan verileri atlamak bu SVD hesabini bir “veri tamamlama problemi” haline getiriyor, piyasadaki neredeyse diger tum kutuphaneler olmayan veriyi sifir olarak kabul eder. Bu farkli bir problemidir. Bu diger SVD yontemlerinde her ne kadar olan veriyi ortolasak ve sifir degerleri sifir averajla ayni anlama gelmeye baslasa bile, yine de olmayan veriyi sifir kabul etmek onu atlamaktan farklidir [12].

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 onlari ana matrisindeki degerini sildik (yerine nan koyarak), ve bir kısmi 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 böylece az sayıda not vermiş kullanıcıların verisini azaltmamış oluyoruz. Ayrıca belli miktarda, rowlim kadar test noktası elde edince iş bitti kabul ediyoruz. Test verisi yaratmak için %80-%20 gibi bir ayırım yapmadık, yani eğitim verisindeki tüm kullanıcıları ve onların neredeyse tüm verisini eğitim için kullanıyoruz.

Movielens verisine geelim. *SVD, Toplu Tavsiye* yazısındaki `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 değerlerini 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, 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
```

Sonuç fena değil.

Formülasyonun 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] yaklasimi detayıyla 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 arastirince 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 sakınca yoktur.

Fakat boyle buluslarda yine de bazi garantiler temel alinmis olabilir, arastirmaci tamamen balıklama 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 farkedir. 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 metodu 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 olacaktır. Bu bulusun puf noktası bu bilgide, ve ustteki teorik benzerliklerde, onlari biliyor olmakta yatiyor. Eger bunlar biliniyor ise, ve saglam lineer cebir bilgisi ile gerektiği zaman onlari ne kadar esnetebilecegimizi 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

```

## Numba ve Funk SVD

Eger Numba [13] kullanirsak, SVD kodunu cok daha hizli isletebiliriz. Ayrica Funk'in kodlamasi (ki alttaki kodu onu temel alacak) biraz daha ilginç, mesela en dis dongu ozellikler (feature) geziyor, onun icindeki birkac yuz kez yine kendi icinde olan tahmin/hata hesabini yapiyor, tum veri seti uzerinde. Bunun icin Movielens 100k verisi lazim, ardindan data\_m100k.py ile veri yaratilir,

```

# Requires Movielens 100k data
from scipy.io import mmread, mmwrite
import numpy as np, time, sys
from numba import jit
import os

def create_user_feature_matrix(review_matrix, NUM_FEATURES):
    num_users = review_matrix.shape[0]
    user_feature_matrix = 1./NUM_FEATURES * \
        np.random.randn(NUM_FEATURES, num_users).astype(np.float32)
    return user_feature_matrix

def create_movie_feature_matrix(review_matrix, NUM_FEATURES):
    num_movies = review_matrix.shape[1]
    movie_feature_matrix = 1./NUM_FEATURES * \
        np.random.randn(NUM_FEATURES, num_movies).astype(np.float32)
    return movie_feature_matrix

@jit(nopython=True)
def predict_rating(user_id, movie_id, user_feature_matrix, movie_feature_matrix):
    rating = 1.
    for f in range(user_feature_matrix.shape[0]):
        rating += \
            user_feature_matrix[f, user_id] * \
            movie_feature_matrix[f, movie_id]
    if rating > 5: rating = 5
    elif rating < 1: rating = 1
    return rating

@jit(nopython=True)
def sgdl_inner(feature, A_row, A_col, A_data,
               user_feature_matrix, movie_feature_matrix,
               NUM_FEATURES):
    K = 0.015
    LEARNING_RATE = 0.001

```

```

squared_error = 0
for k in range(len(A_data)):
    user_id = A_row[k]
    movie_id = A_col[k]
    rating = A_data[k]
    p = predict_rating(user_id,
                       movie_id,
                       user_feature_matrix,
                       movie_feature_matrix)

    err = rating - p
    squared_error += err ** 2
    user_feature_value = user_feature_matrix[feature, user_id]
    movie_feature_value = movie_feature_matrix[feature, movie_id]
    user_feature_matrix[feature, user_id] += \
        LEARNING_RATE * (err * movie_feature_value - K * user_feature_value)
    movie_feature_matrix[feature, movie_id] += \
        LEARNING_RATE * (err * user_feature_value - K * movie_feature_value)

return squared_error

def calculate_features(A_row, A_col, A_data,
                      user_feature_matrix, movie_feature_matrix,
                      NUM_FEATURES):
    MIN_IMPROVEMENT = 0.0001
    MIN_ITERATIONS = 200
    rmse = 0
    last_rmse = 0
    print len(A_data)
    num_ratings = len(A_data)
    for feature in xrange(NUM_FEATURES):
        iter = 0
        while (iter < MIN_ITERATIONS) or (rmse < last_rmse - MIN_IMPROVEMENT):
            last_rmse = rmse
            squared_error = sgd_inner(feature, A_row, A_col, A_data,
                                      user_feature_matrix, movie_feature_matrix,
                                      NUM_FEATURES)
            rmse = (squared_error / num_ratings)
            iter += 1
        print ('Squared error = %f' % squared_error)
        print ('RMSE = %f' % rmse)
        print ('Feature = %d' % feature)
    return last_rmse

if __name__ == "__main__":

    LAMBDA = 0.02
    NUM_FEATURES = 30

    A = mmread('%s/Downloads/A_m100k_train' % os.environ['HOME'])

    user_feature_matrix = create_user_feature_matrix(A, NUM_FEATURES)
    movie_feature_matrix = create_movie_feature_matrix(A, NUM_FEATURES)

    A = A.tocoo()

```

```

rmse = calculate_features(A.row, A.col, A.data,
                        user_feature_matrix, movie_feature_matrix,
                        NUM_FEATURES )

print 'rmse', rmse

np.savetxt("/tmp/user_feature_matrix2.dat", user_feature_matrix)
np.savetxt("/tmp/movie_feature_matrix2.dat", movie_feature_matrix)

```

## 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>
- [12] <https://dgleich.wordpress.com/2013/10/19/svd-on-the-netflix-matrix>
- [13] <http://sayilarvekuramlar.blogspot.de/2014/09/numba-llvm-ve-svd.html>