

Rasgele İzdüşümü (Random Projection) ile SVD

Eğer ana matrisimiz  $A$ 'nin çok fazla kolonu var ise bunu bir şekilde azaltmanın yollarını arayabiliriz. [1]'e göre bunu yapmanın yollarından biri rasgele izdusum hesabıdır. İlk önce  $n \times k$  boyutunda bir Gaussian rasgele matris  $\Omega$  üretiriz. Ardından

$$Y = A\Omega$$

hesaplanır. Bu  $Y$  üzerinde QR ayrıştırması yaparız, ve elde edilen  $Q$  ile

$$B = Q^T A$$

hesabını yaparız. Ardından bu matris üzerinde SVD ayrıştırması yaparız,

$$B = \hat{U}\Sigma V^T$$

ve

$$U = Q\hat{U}$$

matrisini hesaplarız. Ana fikir suradan geliyor,

$$A = QQ^T A$$

ki bu standart bir cebir numarası olurdu,  $Q$  yerine rasgele izdusumdan gelen yaklaşık  $Q$ 'yu kullanabiliriz, o zaman

$$A \approx \tilde{Q}\tilde{Q}^T A$$

olacaktır. Yani izdusumdan gelen  $Q, R$  gerçek versiyona yakın. Üstteki carpımda  $R$  yerine  $B$  harfi kullanıyoruz, ki  $B = \tilde{Q}^T A$  oluyor, yani

$$A \approx \tilde{Q}B$$

ya da

$$B \approx \tilde{Q}^T A$$

.

O zaman İstatistik notlarımız altındaki *Paralel Matris Carpımı*, *Ax*, *QR* ve *SVD* yazısında olduğu gibi  $B$ 'nin SVD'sini alarak yaklaşık bir  $U$  elde etmek mümkün olacaktır.

Bu yaklasiksal metot isler cunku noktalar yaklasiksal bir altuzaya yansittikten sonra elde edilen yeni noktalarin arasindaki mesafelerin fazla bozulmadan muhafaza edildiği soylenir, daha detayli soylemek gerekirse,  $n$ -boyutlu verileri  $O(\log n/\epsilon^2)$  boyutundaki bir rasgele altuzaya yansitmak, pozitif olasilikla, yeni noktalarin arasindaki mesafeleri sadece  $1 \pm \epsilon$  olcusunde degistirir [2].  $Y$ 'nin,  $A$ 'nin "menzilini" iyi temsil ettigi de soylenir.

Test olarak suradaki [3] veri seti uzerinde gorelim:

```
import numpy.random as rand
import numpy.linalg as lin
import pandas as pd

k = 7 # izdusum uzayinin boyutlari
df = pd.read_csv("w1.dat", sep=';', header=None)
A = np.array(df)[: , 1:]

print "A", A.shape

rand.seed(1000)

Omega = rand.randn(A.shape[1], k)

Y = np.dot(A, Omega)

print "Y", Y.shape

Q, R = lin.qr(Y)

# niye devrigi ile is yaptigimizi altta anlatiyoruz
BT = np.dot(A.T, Q)

print "Q", Q.shape
print "BT", BT.shape

x, x, V = lin.svd(BT)

print 'V', V.shape

Uhat = V.T # cunku B=USV', B'=VSU' U of B icin V' lazim

print "Uhat", Uhat.shape

U = np.dot(Q, Uhat)

print "U", U.shape
```

```

plt.plot(U[:,0],U[:,1], 'r+')

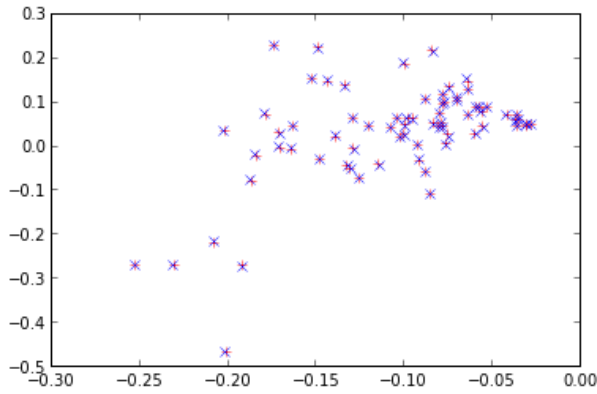
plt.hold(True)

# gercek SVD ile karsilastir

U, Sigma, V = lin.svd(A);
plt.plot(U[:,0],-U[:,1], 'bx')
plt.savefig('rnd_1.png')

A (71, 30)
Y (71, 7)
Q (71, 7)
BT (30, 7)
V (7, 7)
Uhat (7, 7)
U (71, 7)

```



Mavi noktalar  $A$  üzerinde "gercek" SVD sonucu, kırmizilar yansitma sonrasi elde edilen  $U$ . Sonuclar cok iyi.

$B$  yerine  $B^T$

Kodlama acisindan, ya da buyuk veri baglaminda baska amaclar [4] icin  $B = Q^T A$  yerine  $B^T = A^T Q$  hesabi yapmak istenilebilir. Niye? Cunku cikti olarak  $n \times k$  matrisi istiyor olabiliriz,  $k \times n$  matrisi istemiyoruz, yani cok olanin satirlar olmasini istiyoruz, kolonlar olmasini istemiyoruz.

O zaman, elde edilen  $B^T$  ise,  $B$  uzerinde degil  $B^T$  uzerinde SVD alacagiz demektir, bu da sonuclari birazcik degistirir, yani

$$B = U\Sigma V^T$$

$$B^T = V\Sigma U^T$$

haline gelir. Yani  $B$ 'nin  $U$ 'sünü elde etmek için  $B^T$ 'nin SVD'si sonrasında ele geçen sonucta  $(U_{BT}^T)^T$  yapmak gerekir. Her şeyin hafızada yapıldığı durumda bu fark yaratmaz, fakat "ilerisi için", yani esle / indirge ortamları için akılda tutmak faydalı olur.

#### Kaynaklar

- [1] Halko, N., Randomized methods for computing low-rank approximations of matrices
- [2] Gupta, A., Dasgupta, S., An Elementary Proof of a Theorem of Johnson and Lindenstrauss
- [3] [archive.ics.uci.edu/ml/datasets/Breast+Cancer](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer)
- [4] [arxiv.org/abs/1310.4664](http://arxiv.org/abs/1310.4664)