

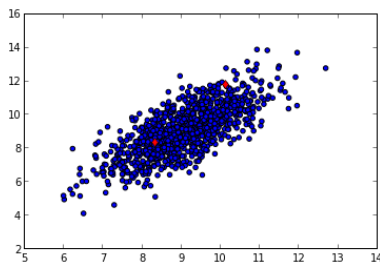
Temel Bileşen Analizi (Principal Component Analysis -PCA-)

PCA yöntemi boyut azaltan yöntemlerden biridir, takip edilmeden (unsupervised) isleyebilir. Ana fikir veri noktalarının izdusumunun yapılacağı yonlar bulmaktır ki bu yonlar bağlamında (izdusum sonrası) noktaların arasındaki varyans (variance) en fazla olsun, yani noktalar en "yaygın" şekilde bulunsunlar. Böylece birbirinden daha uzaklaşan noktaların mesela daha rahat kümelenebileceğini umabiliriz. Yani bir diğer amaç hangi değişkenlerin varyansının daha fazla olmasının görülmesi üzerine, o değişkenlerin daha önemli olabileceğinin anlaşılması. Örnek olarak alttaki grafiğe bakalım,

```
from pandas import *
data = read_csv("testSet.txt", sep="\t", header=None)
print data[:10]
```

	0	1
0	10.235186	11.321997
1	10.122339	11.810993
2	9.190236	8.904943
3	9.306371	9.847394
4	8.330131	8.340352
5	10.152785	10.123532
6	10.408540	10.821986
7	9.003615	10.039206
8	9.534872	10.096991
9	9.498181	10.825446

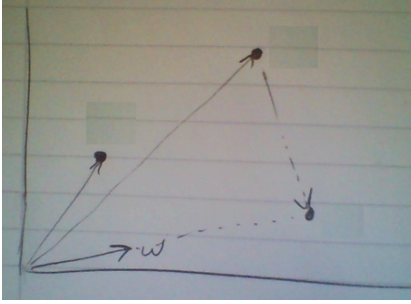
```
plt.scatter(data.ix[:,0],data.ix[:,1])
plt.plot(data.ix[1,0],data.ix[1,1], 'rd')
plt.plot(data.ix[4,0],data.ix[4,1], 'rd')
plt.savefig('pca_1.png')
```



PCA ile yapmaya çalıştığımız öyle bir yon bulmak ki, x veri noktalarının tamamının o yöne izdusumu yapılıncaya sonuc olacak, "izdusumu yapılmış" z 'nin varyansı en büyük olsun. Bu bir maksimizasyon problemidir. Fakat ondan önce x nedir, z nedir bunlara yakından bakalım.

Veri x ile tüm veri noktaları kastedilir, fakat PCA probleminde genellikle bir "vektörün diğeri üzerine" yapılan izdusumu, "daha optimal bir w yonu bulma", ve "o

yone dogru izdusum yapmak” kelimeleri kullanilir. Demek ki veri noktalarini bir vektor olarak gormeliyiz. Eger ustte kirmizi ile isaretlenen iki noktayi alirsak (bu noktalar verideki 1. ve 4. siradaki noktalar),



gibi bir goruntuden bahsediyoruz. Hayali bir w kullandik, ve noktalardan biri veri noktası, w uzerine izdusum yapilarak yeni bir vektoru / noktayı ortaya cikartiyor. Genel olarak ifade edersek, bir nokta icin

$$z = w^T x$$

Yapmaya calistigimiz varyansi maksimize etmek demistik. Ozel bir izdusum yonunu referans alirsak, en buyuk $Var(z_1)$ 'i bulacagiz. Not: Bunu soyler soylemez x 'i ve z 'yi rasgele degiskenler olarak gordugumuzu belirtmis oluyoruz. Ardi ardina alet cantasindan temsili numaralari kullaniyoruz - x 'i bir yandan vektor yapiyoruz, bir yandan bir dagilimdan gelen zar atisi gibi goruyoruz, problemi cozmemize ne yardim edecekse onu surekli devreye sokuyoruz. Devam edelim, rasgele degisken deyince, demek ki x, z dagilimlardan gelecektir, bu dagilimların çok boyutlu normal (multivariate normal) oldugunu kabul edebiliriz. Peki eger $x, N(\mu, \Sigma)$ gibi çok boyutlu normal dagilimdan geliyorsa, ki μ ortalama ve Σ kovaryanstir, $w^T x$ nedir?

Bu yeni "seyin" beklentisi ve varyansına bakalım,

$$E[w^T x] = w^T E[x] = w^T \mu$$

$$Var(w^T x) = w^T Var(x) w = w^T \Sigma w$$

Ustteki sonuclarin boyutlarına dikkat: $w^T \mu$ durumunda $1 \times N \cdot N \times 1 = 1 \times 1$, $w^T \Sigma w$ durumunda ise $1 \times N \cdot N \times N \cdot N \times 1 = 1 \times 1$. İki durumda da tek boyutlu skalar degerler elde ettik. Demek ki w^T ile carpim sonrasi bir $N(w^T \mu, w^T \Sigma w)$ dagilimi elde ederiz ve bu dagilim bir tek boyutlu bir dagilimdir. Yani w yonundeki izdusum bize tek boyutlu bir Gaussian dagilimini verecektir. Bu sonuc aslında çok sasirtici olmasa gerek, tum veri noktalarini alip, baslangici orijin noktasında olan vektorlere ceviriip ayni yone isaret edecek sekilde duzenliyoruz, bu vektorleri tekrar nokta olarak dusunursek, tabii ki ayni yonu gosteriyorlar, bilahere ayni cizgi uzerindeki noktalara donusuyorlar. Ayni cizgi uzerinde olmak ne demek? Tek boyuta inmis olmak demek.

Bastaki amacimiza donersek, $Var(z_1)$ 'i maksimize etmek ayni anda $Var(w_1^T \Sigma w_1)$ 'i maksimize etmek demektir.

Ufak bir sorun $w_1^T \Sigma w_1$ 'i surekli daha buyuk w_1 'lerle sonsuz kadar buyutebilirsiniz. Bize ek bir kisiltama sarti daha lazim, bu sart $\|w\| = 1$ olabilir, yani w 'nin norm'u 1'den daha buyuk olmasin. Boylece optimizasyon w 'nin buyuklugu uzerinde taklalar atmayacak, sadece yon bulmak ile ilgilenecek, iyi, zaten biz w 'nin yonu ile ilgileniyoruz. Aradigimiz ifadeyi yazalim, ve ek siniri Lagrange ifadesi olarak ekleyelim, ve yeni bir L ortaya cikartalim,

$$L(w_1, \lambda) = w_1^T \Sigma w_1 - \lambda(w_1^T w_1 - 1)$$

Niye eksiden sonraki terim o sekilde eklendi? O terim oyle sekilde secildi ki, $\partial L / \partial \lambda = 0$ alinince $w_1^T w_1 = 1$ geri gelsin / ortaya ciksinsin [2, sf 340], bu Lagrange'in dahice bulusu. Bunu kontrol edebilirsiniz, λ 'ya gore turev alirken w_1 sabit olarak yokolur, parantez icindeki ifadeler kalir ve sifira esitlenince orijinal kisiltama ifadesi geri gelir. Simdi

$$\max_{w_1} L(w_1, \lambda)$$

Turevi w_1 'e gore alirsak, ve sifira esitlersek,

$$2w_1 \Sigma - 2\lambda w_1 = 0$$

$$2w_1 \Sigma = 2\lambda w_1$$

$$\Sigma w_1 = \lambda w_1$$

Ustteki ifade ozdeger, ozvektor ana formulune benzemiyor mu? Evet. Eger w_1 , Σ 'nin ozvektoru ise ve esitligin sagindaki λ ona tekabul eden ozdeger ise, bu esitlik dogru olacaktir.

Peki hangi ozdeger / ozvektor maksimal degeri verir? Unutmayalim, maksimize etmeye calistigimiz sey $w_1^T \Sigma w_1$ idi

Eger $\Sigma w_1 = \lambda w_1$ yerine koyarsak

$$w_1^T \lambda w_1 = \lambda w_1^T w_1 = \lambda$$

Cunku $w_1^T w_1$ 'nin 1 olacagi sartini koymustuk. Neyse, maksimize etmeye calistigimiz deger λ cikti, o zaman en buyuk λ kullanirsak, en maksimal varyansi elde ederiz, bu da en buyuk ozdegerin ta kendisidir.

Demek ki izdusum yapılacak "yon" kovaryans Σ 'nin en büyük özdegerine tekabül eden özvektor olarak seçilirse, temel bileşenlerden en önemlisini hemen bulmuş olacağız.

Cebirin geri kalanı w_2, w_3 için devam eder, bu türetmenin detaylarını [1] ve [2] gibi kaynaklarda bulabilirsiniz. Fakat ulaşılan sonuç en bileşenlerin önem sırasının aynen özdeğerlerin büyüklük sırasına tekabül ediyor olması.

Örnek

Şimdi tüm bunları bir örnek üzerinde görelim. İki boyutlu örnek veriyi üstte yüklemistik. Şimdi veriyi "sıfırda ortalayacağız" yani her kolon için o kolonun ortalama değerini tüm kolondan çıkartacağız. PCA ile işlem yaparken tüm değerlerin sıfır merkezli olması gerekiyor.

Daha sonra özdeğerlerini, vektörlerini hesaplayabilmek için verinin kovaryansını hesaplayacağız.

```
import numpy.linalg as lin
from pandas import *
data = read_csv("testSet.txt", sep="\t", header=None)
print data[:10]
```

```
means = data.mean()
meanless_data = data - means
cov_mat = np.cov(meanless_data, rowvar=0)
eigs, eigv = lin.eig(cov_mat)
eig_ind = np.argsort(eigs)
print eig_ind
```

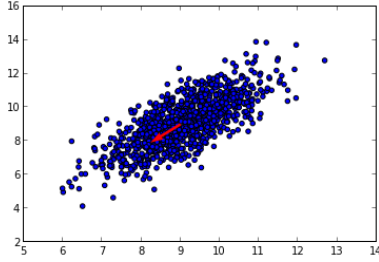
```
      0      1
0  10.235186  11.321997
1  10.122339  11.810993
2   9.190236   8.904943
3   9.306371   9.847394
4   8.330131   8.340352
5  10.152785  10.123532
6  10.408540  10.821986
7   9.003615  10.039206
8   9.534872  10.096991
9   9.498181  10.825446
[0 1]
```

```
print eigs[1], eigv[:,1].T
print eigs[0], eigv[:,0].T
```

```
2.89713495618 [-0.52045195 -0.85389096]
0.366513708669 [-0.85389096  0.52045195]
```

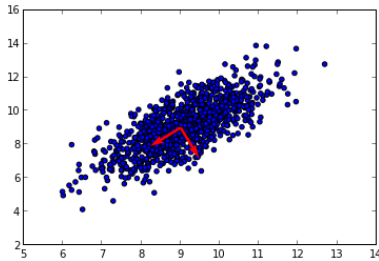
En buyuk olan yonu quiver komutunu kullanarak orijinal veri seti uzerinde gostere-
lim,

```
plt.scatter(data.ix[:,0],data.ix[:,1])  
# merkez 9,9, tahminen secildi  
plt.quiver(9,9,eigv[1,1],eigv[0,1],scale=10,color='r')  
plt.savefig('pca_2.png')
```



Goruldugu gibi bu yon hakikaten dagilimin, veri noktalarinin en cok yayilmis oldugu
yon. Demek ki PCA yontemi dogru sonucu buldu. Her iki yonu de cizersek,

```
plt.scatter(data.ix[:,0],data.ix[:,1])  
plt.quiver(9,9,eigv[1,0],eigv[0,0],scale=10,color='r')  
plt.quiver(9,9,eigv[1,1],eigv[0,1],scale=10,color='r')  
plt.savefig('pca_3.png')
```



Bu ikinci yon birinciye dik olmalıydı, ve o da bulundu. Aslında iki boyut olunca
baska secenek kalmiyor, 1. yon sonrasi ikincisi baska bir sey olamazdi, fakat cok
daha yuksek boyutlarda en cok yayilimin oldugu ikinci yon de dogru sekilde geri
getirilecekti.

SVD ile PCA Hesaplamak

PCA bolumunde anlatilan yontem temel bileşenlerin hesabında özdeğerler ve özvek-
torler kullandı. Alternatif bir yontem Tekil Değer Ayırıştırma (Singular Value De-
composition -SVD-) üzerinden bu hesabi yapmaktır. SVD için Lineer Cebir Ders
29'a bakabilirsiniz. Peki ne zaman klasik PCA ne zaman SVD üzerinden PCA kul-
lanmalı? Bir cevap belki mevcut kutuphanelerde SVD kodlamasının daha iyi olması,
ayırıştırmanın özvektor / değer hesabından daha hızlı işleyebilmesi [6].

Ayrıca birazdan göreceğimiz gibi SVD, kovaryans matrisi üzerinde değil, A 'nin kendisi üzerinde isletilir, bu hem kovaryans hesaplama aşamasını atlamamızı, hem de kovaryans hesabi sırasında ortaya çıkabilecek numerik puruzlerden korunmamızı sağlar (çok ufak değerlerin kovaryans hesabını bozabileceği literatürde bahsedilmektedir).

PCA ve SVD bağlantısına gelelim:

Biliyoruz ki SVD bir matrisi şu şekilde ayırıştırır

$$A = USV^T$$

U matrisi $n \times n$ dikgen (orthogonal), V ise $m \times m$ dikgen. S 'in sadece köşegeni üzerinde değerler var ve bu σ_j değerleri A 'nin tekil değerleri (singular values) olarak biliniyor.

Şimdi A yerine AA^T koyalım, yani A 'nin kovaryans matrisinin SVD ayırtmasını yapalım, acaba elimize ne geçecek?

$$\begin{aligned} AA^T &= (USV^T)(USV^T)^T \\ &= (USV^T)(VS^T U^T) \\ &= USS^T U^T \end{aligned}$$

S bir köşegen matrisi, o zaman SS^T matrisi de köşegen, tek farkla köşegen üzerinde artık σ_j^2 değerleri var. Bu normal.

SS^T yerine Λ sembolünü kullanalım, ve denklemi iki taraftan (ve sağdan) U ile çarparsak (unutmayalım U ortanormal bir matris ve $U^T U = I$),

$$AA^T U = U \Lambda U^T U$$

$$AA^T U = U \Lambda$$

Son ifadeye yakından bakalım, U 'nın tek bir kolonuna, u_k diyelim, odaklanacak olursak, üstteki ifadede bu sadece kolona yönelik nasıl bir eşitlik çıkartabilirdik? Şöyle çıkartabilirdik,

$$(AA^T)u_k = \sigma^2 u_k$$

Bu ifade tanıdık geliyor mu? Özdeğer / özvektor klasik yapısına eriştik. Üstteki eşitlik sadece ve sadece eğer u_k , AA^T 'nin özvektörü ve σ^2 onun özdeğeri ise geçerlidir.

Bu esitligi tum U kolonlari icin uygulayabilecegimize gore demek ki U 'nun kolonlarında AA^T 'nin ozvektorleri vardır, ve AA^T 'nin ozdegerleri A 'nin tekil degerlerinin karesidir.

Bu muthis bir bulus. Demek ki AA^T 'nin ozektorlerini hesaplamak icin A uzerinde SVD uygulayarak U 'yu bulmamiz yeterli, kovaryans matrisini hesaplamak bile gerekmiyor! AA^T ozdegerleri uzerinde buyukluk karsilastirmasi icin ise A 'nin tekil degerlerine bakmak yeterli!

Ornek

Ilk bolumdeki ornege donelim, ve ozvektorleri SVD uzerinden hesaplatelim.

```
U,s,Vt = svd(meanless_data.T,full_matrices=False)
print U
[[-0.52045195 -0.85389096]
 [-0.85389096  0.52045195]]

print np.dot(U.T,U)
[[ 1.00000000e+00  3.70255042e-17]
 [ 3.70255042e-17  1.00000000e+00]]
```

Goruldugu gibi ayni ozvektorleri bulduk.

New York Times Yazıları Analizi

Simdi daha ilginç bir ornege bakalim. Bir arastirmaci belli yillar arasindaki NY Times makalelerinde her yazida hangi kelimenin kac kere ciktiginin verisini toplamis [1,2,3], bu veri 4000 kusur kelime, her satir (yazi) icin bir boyut (kolon) olarak kaydedilmis. Bu veri nytimes.csv uzerinde ek bir normalize isleminde sonra, onun uzerinde boyut indirgeme yapabiliriz.

Veri setinde her yazi ayrica ek olarak sanat (arts) ve muzik (music) olarak etiketlenmis, ama biz PCA kullanarak bu etiketlere hic bakmadan, verinin boyutlarini azaltarak acaba verinin "ayrilabilir" hale indirgenip indirgenemedigine bakacagiz. Sonra etiketleri veri ustune koyup sonucun dogrulugunu kontrol edecegiz.

Bakmak derken veriyi (en onemli) iki boyuta indirgeyip sonucu grafikleyecegiz. Illa 2 olması gerekmez tabii, 10 boyuta indirgeyip (ki 4000 kusur boyuttan sonra bu hala muthis bir kazanim) geri kalanlar uzerinde mesela bir kumeleme algoritmasi kullanabilirdik.

Ana veriyi yukleyip birkac satirini ve kolonlarini gosterelim.

```
from pandas import *
import numpy.linalg as lin
nyt = read_csv ("nytimes.csv")
labels = nyt['class.labels']
print nyt.ix[:8,102:107]
```

	after	afternoon	afterward	again	against
0	1	0	0	0	0
1	1	1	0	0	0
2	1	0	0	1	2
3	3	0	0	0	0
4	0	1	0	0	0
5	0	0	0	1	2
6	7	0	0	0	1
7	0	0	0	0	0
8	0	0	0	0	0

Yuklemeyi yapip sadece etiketleri aldik ve onlari bir kenara koyduk. Simdi onemli bir normalizasyon islemi gerekiyor - ki bu isleme ters dokuman-frekans agirliklandirmasi (inverse document-frequency weighting -IDF-) ismi veriliyor - her dokumanda aşırı fazla ortaya cikan kelimelerin onemi ozellikle azaltiliyor, ki diger kelimelerin etkisi artabilsin.

IDF kodlamasi alttaki gibidir. Once class.labels kolonunu atariz. Sonra "herhangi bir deger iceren" her hucrenin 1 digerlerinin 0 olmasi icin kullanılan DataFrame üzerinde astype(bools) isletme numarasini kullaniriz, boylece asiri büyük degerler bile sadece 1 olacaktır. Bazi diger islemler sonrasI her satiri kendi icinde tekrar normalize etmek icin o satirdaki tum degerlerin karesinin toplamının karekokunu aliriz ve satirdaki tum degerler bu karekok ile bolunur. Buna oklitsel (euclidian) normalizasyon denebilir.

Not: Oklitsel norm alirken toplamın hemen ardından çok ufak bir 1e-16 degeri eklememize dikkat cekelim, bunu toplamın sıfır olma durumu icin yapıyoruz, ki sonra sıfırla bolerken NaN sonucundan kacinalim.

```
nyt = nyt.drop(['class.labels'],axis=1)
freq = nyt.astype(bool).sum(axis=0)
freq = freq.replace(0,1)
w = np.log(float(nyt.shape[0])/freq)
nyt = nyt.apply(lambda x: x*w,axis=1)
nyt = nyt.apply(lambda x: x / np.sqrt(np.sum(np.square(x))+1e-16), axis=1)
```

```
nyt=nyt.ix[:,1:] # ilk kolonu atladiK
print nyt.ix[:8,102:107]
```

	afterward	again	against	age	agent
0	0	0.000000	0.000000	0.051085	0
1	0	0.000000	0.000000	0.000000	0
2	0	0.021393	0.045869	0.000000	0
3	0	0.000000	0.000000	0.000000	0
4	0	0.000000	0.000000	0.000000	0
5	0	0.024476	0.052480	0.000000	0
6	0	0.000000	0.008536	0.000000	0
7	0	0.000000	0.000000	0.000000	0


```
8          0  0.000000  0.000000  0.000000          0
```

SVD yapalım

```
nyt = nyt - nyt.mean(0)
u,s,v = lin.svd(nyt.T,full_matrices=False)
print s[:10]

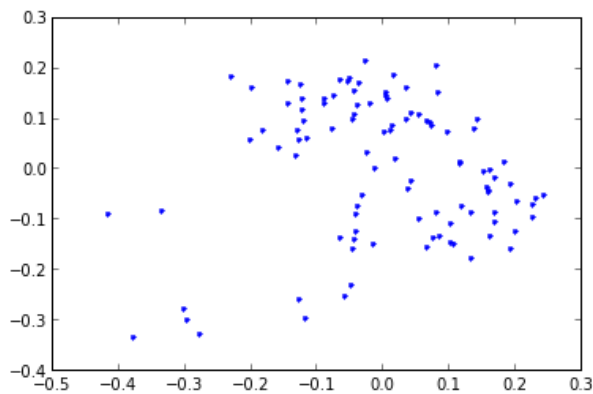
[ 1.41676764  1.37161893  1.31840061  1.24567955  1.20596873  1.18624932
  1.15118771  1.13820504  1.1138296   1.10424634]
```

```
print u.shape
```

```
(4430, 102)
```

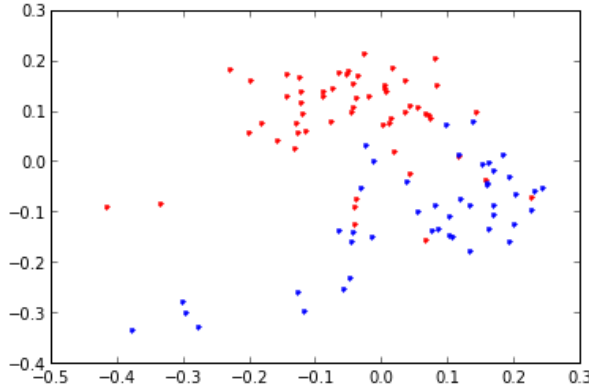
SVD'nin verdiği u icinden iki ozvektoru seciyoruz (en bastakiler, cunku Numpy SVD kodu bu ozvektorleri zaten siralanmis halde dondurur), ve veriyi bu yeni kordinata izdusumluyoruz.

```
proj = np.dot(nyt, u[:, :2])
proj.shape
plt.plot(proj[:,0],proj[:,1],'.')
plt.savefig('pca_4.png')
```



Simdi ayni veriyi bir de etiket bilgisini devreye sokarak cizdirelim. Sanat kirmizi muzik mavi olacak.

```
arts =proj[labels == 'art']
music =proj[labels == 'music']
plt.plot(arts[:,0],arts[:,1],'.r')
plt.plot(music[:,0],music[:,1],'.b')
plt.savefig('pca_5.png')
```



Goruldugu gibi veride ortaya cikan / ozvektorlerin kesfettigi dogal ayirim, hakikaten dogruymus.

Metotun ne yaptigina dikkat, bir suru boyutu bir kenara atmamiza ragmen geri kalan en onemli 2 boyut uzerinden net bir ayirim ortaya cikartabiliyoruz. Bu PCA yonteminin iyi bir is becerdigini gosteriyor, ve kelime sayilarinin makalelerin icerigi hakkında ipucu icerdigini ispatliyor.

Kaynaklar

- [1] Alpaydin, E., Introduction to Machine Learning, 2nd Edition
- [2] Strang, G., Linear Algebra and Its Applications, 4th Edition
- [3] <http://www.stat.columbia.edu/~fwood/Teaching/w4315/Spring2010/PCA/slides.pdf>
- [4] Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View
- [5] <http://www ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T19>
- [6] <http://www.stat.cmu.edu/~cshalizi/490/pca/>
- [6] <http://www.math.nyu.edu/faculty/goodman/teaching/RPME/notes/Section3.pdf>
- [7] Lineer Cebir notlarimizda SVD turetilmesine bakinca ozdeger/vektor mantigina atif yapildigini gorebiliriz ve akla su gelebilir; "ozdeger / vektor rutini isletmek-ten kurtulalim dedik, SVD yapiyoruz, ama onun icinde de ozdeger/vektor hesabi var". Fakat sunu belirtmek gerekir ki SVD numerik hesabini yapmanin tek yontemi ozdeger/vektor yontemi degildir. Mesela Numpy Linalg kutuphanesi icindeki SVD, LAPACK dgesdd rutinini kullanir ve bu rutin ic kodlamasinda QR, ve bir tur bol / istila et (divide and conquer) algoritmasi isletmektedir.