

SVD ile Kumeleme

Tekil Deger Ayristirma (Singular Value Decomposition -SVD-) ile bir veri madenciligi ornegi gorecegiz. Ornek olarak [1] adresinde tarif edilen / paylasilan zaman serisini kullandik. Serinin tumunu kullanilmadi, ilk 10 noktasini aldik, ve grafige bakinca iki tane ana seri turu oldugunu goruyoruz.

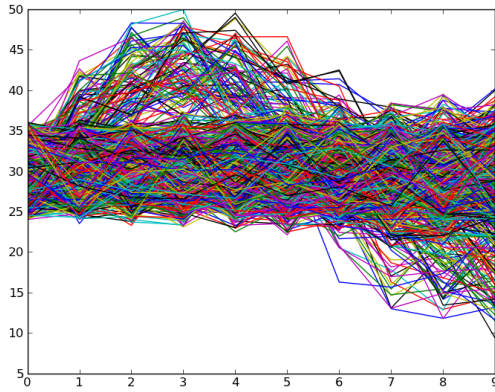
```
import numpy as np
from pylab import *

data = np.genfromtxt("synthetic_control.data", dtype=float)

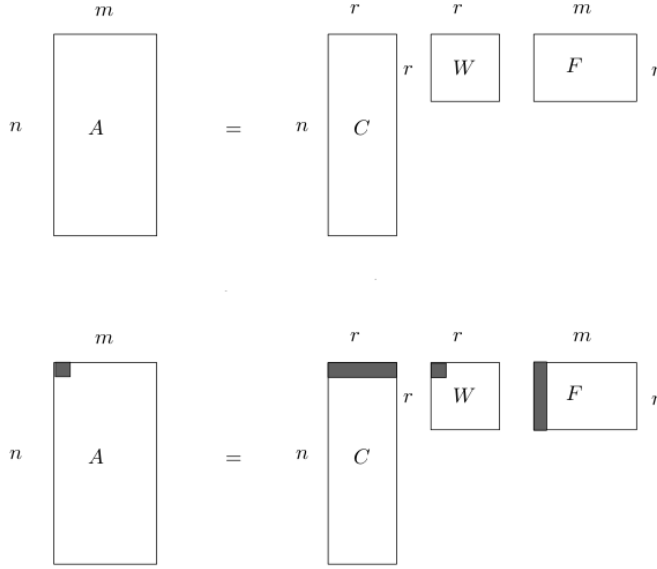
print data.shape

for t in data[:,0:10]:
    plot(t); hold(True)

show()
```



Peki bu serileri nasil otomatik olarak kumeleyerek bulurduk / birbirinden ayirtterdik? *Lineer Cebir Ders 29*'da SVD'nin matematigini isledik. SVD bir matris A uzerinde ayristirma yapar, ve A herhangi boyutta, turde bir matris olabilir.



Ayrıştırma $m \times n$ boyutlu matrisi $A = CWF$ olarak ayristirir, burada C , ana matris ile ayni miktarda satira sahiptir, F ayni miktarda kolona sahiptir. Ayrıştırma sonrasi A 'nin kertes (rank) r ortaya cikar, eger tum A kolonlari birbirinden bagimsiz ise, o zaman $r = m$ olacaktir, ama kolonlari bazilari mesela ayni olcumu degisik katlarda tekrarliyor ise, o zaman matriste tekillik vardir, ve bu durumda $r < m$ olur, ve ortadaki W matrisi $r \times r$ oldugu icin beklenenden daha ufak boyutlarda olabilir.

Ayrica SVD, W caprazindaki ozdegerleri buyukluk sirasina gore dizer, ve her ozdegere tekabul eden ozvektorler de ona gore siraya dizilmis olacaktir, ve SVD tamamlaninca mesela “en buyuk 10” ozdegere ait olan CWF degerlerini alip, digerlerini atmayi da secebiliriz, yani kerte uzerinden yapilan “eleme” ustune bir eleme de kendimiz yapabiliriz. Bu elemeyi yapabilmemizin mantigi soyle; kucuk ozdegerlerin carptigi ozvektorlerin nihai toplama daha az etki ettigi soylenebilir, ve bu “gurultuyu” elemek sonucu degistirmeyecektir. Ayrica bu elemeyi yaparak bir tur boyut azaltma (dimensionality reduction) islemini de ayni zamanda basarmis oluruz.

Ayrıştırmanın Anlamları

Bir ayristirmayi degisik sekillerde gormek mumkundur. Bunlardan onemli birisi cizge bakis acisidir (graph interpretation). Cizge bilindigi gibi dugumler ve onlar arasindaki ayritlardan (edges) olusur. Bir cizge matris formunda temsil edilebilir, satir / kolon kesisimi iki dugum arasindaki ayritin agirligini, ya da varligini (1 ve 0 uzerinden) temsil edecektir. Bu durumda SVD sonucunda elde edilen CWF , bize iki dugum arasi gecisli (bipartite) cizgeyi, uc dugum arasi gecisli (tripartite) cizgeye cevrimis halde geri verir. Ve bu yeni cizgede en fazla r tane gecis noktasi (waystations) olusmustur, ustte bahsettigimiz eleme ile gecisler daha da azaltilabilir.

Simdi, bu gecis noktalarına olan C 'nin “baglanma sekli”, “baglanma kuvveti”, ek kumeleme basamagi tarafindan kullanilabilir. Bu “azaltilmis” gecisin uzerindeki her islem / ona yapilan her referans kumeleme icin bir ipucudur. Bunu gormek icin ornek zaman serilerinin SVD sonrasi elde edilen C (ornekte u) matrisinin ilk iki

kolonunu bile grafiklemek yeterlidir.

```
import scipy.linalg as lin
import numpy as np
from pylab import *

data = np.genfromtxt("synthetic_control.data", dtype=float)

# before norm, and take only 10 data points
data = data[:,0:10]

print data.shape

# show the mean, and std of the first time series
print data[0,:]
print np.mean(data[0,:], axis=0)
print np.std(data[0,:], axis=0)

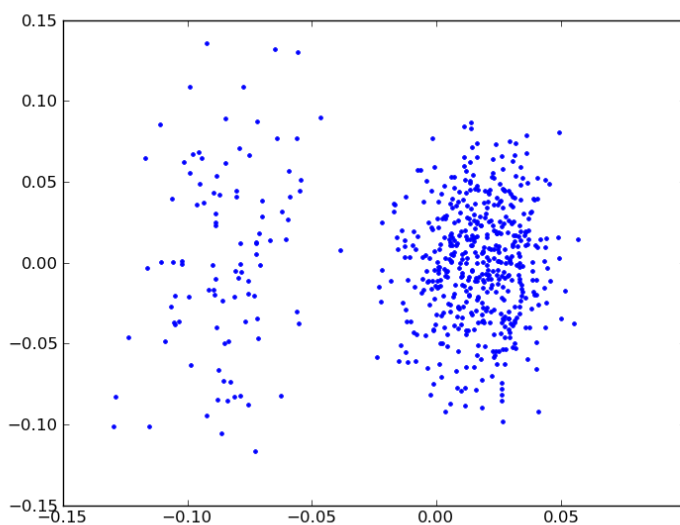
# normalize
data -= np.mean(data, axis=0)
data /= np.std(data, axis=0)

# after norm
print data[0,:]

u,s,v = lin.svd(data, full_matrices=False)
print 'svd'
print u.shape
print s
print v.shape

plot(u[:,0], u[:,1], '. ')

show()
```



Goruldugu gibi net bir sekilde iki tane kume ortaya cikti. Bu kumeler yazinin

basındaki iki ayrı zaman serisi obeklerine tekabül ediyorlar.

O zaman serilerini ayırtetmek için ne yaparız? Üstteki veriler üzerinde kmeans isletebiliriz, ya da kabaca bakıyoruz, dikey olarak -0.025 seviyesinde bir çizgi ayırac olarak görülebilir. Numpy filtreleme tekniği

```
u[:,0] < -0.025
```

bize ana veri üzerinde uygulanabilecek **True** ve **False** değerleri verir, bunları alarak ana veriye filtre olarak uyguluyoruz,

```
data[u[:,0] < -0.025]
```

ve mesela birinci kümeye ait zaman serilerini bulabiliriz.

Kontrol etmek için ilk 3 kolonun değerlerini üç boyutta grafikleyelim.

```
from mpl_toolkits.mplot3d import Axes3D
import scipy.linalg as lin
import numpy as np
from pylab import *

data = np.genfromtxt("synthetic_control.data", dtype=float)

data = data[:,0:10]

print data.shape

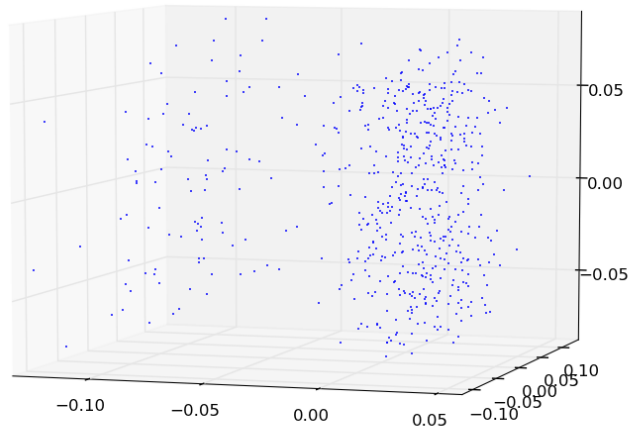
data -= np.mean(data, axis=0)
data /= np.std(data, axis=0)

u,s,v = lin.svd(data)
print 'svd'
print u.shape
print s
print v.shape

fig = plt.figure()
ax = Axes3D(fig)
ax.plot(u[:,0], u[:,1], u[:,2], ',', zs=0, zdir='z', label='zs=0, zdir=z')

show()
```

Yine iki tane küme olduğunu görüyoruz.



Simdi biraz daha degisik bir probleme bakalim, bu sefer bir grup kelimeyi birbirlerine benzerlikleri (ya da uzakligi) üzerinden kumelemeye ugrasacagiz.

Benzerlik, Levenhstein mesafesi adli olcut [2] üzerinden olacak. Matrisimiz her kelimenin her diger kelime ile arasindaki uzakligi veren bir matris olmal, eger 100 kelime var ise, bu matris 100 x 100 boyutlarinda olacak. SVD sonrasi elde edilen u uzerinde kmeans isletecegiz, ve kumeleri bulacagiz. Ayrica her kume icin bir “temsilci” secebilmek icin kmeans’in bize verdigi kume ortasi kordinatinin en yakin oldugu kelimeyi cekip cikartacagiz, ve onu temsilci olarak alacagiz.

```
import numpy as np
import scipy.linalg as lin
import Levenshtein as leven
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import itertools

words = np.array([
    'the', 'be', 'to', 'of', 'and', 'a', 'in', 'that', 'have',
    'I', 'it', 'for', 'not', 'on', 'with', 'he', 'as', 'you',
    'do', 'at', 'this', 'but', 'his', 'by', 'from', 'they', 'we',
    'say', 'her', 'she', 'or', 'an', 'will', 'my', 'one', 'all',
    'would', 'there', 'their', 'what', 'so', 'up', 'out', 'if',
    'about', 'who', 'get', 'which', 'go', 'me', 'when', 'make',
    'can', 'like', 'time', 'no', 'just', 'him', 'know', 'take',
    'people', 'into', 'year', 'your', 'good', 'some', 'could',
    'them', 'see', 'other', 'than', 'then', 'now', 'look',
    'only', 'come', 'its', 'over', 'think', 'also', 'back',
    'after', 'use', 'two', 'how', 'our', 'work', 'first', 'well',
    'way', 'even', 'new', 'want', 'because', 'any', 'these',
    'give', 'day', 'most', 'us'])

print "calculating_distances..."

(dim,) = words.shape
```

```

f = lambda (x,y): leven.distance(x,y)
res=np.fromiter(itertools.imap(f, itertools.product(words, words)),
                dtype=np.uint8)
A = np.reshape(res,(dim,dim))

print "svd..."

u,s,v = lin.svd(A, full_matrices=False)

print u.shape
print s.shape
print s
print v.shape

data = u[:,0:10]
k=KMeans(init='k-means++', k=25, n_init=10)
k.fit(data)
centroids = k.cluster_centers_
labels = k.labels_
print labels

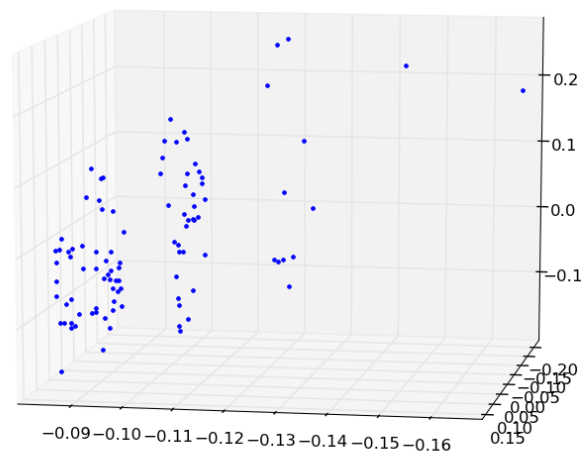
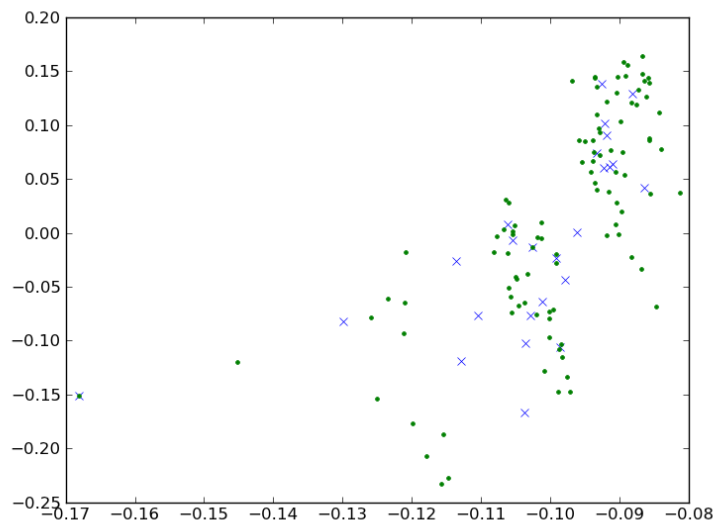
def dist(x,y):
    return np.sqrt(np.sum((x-y)**2, axis=1))

print "clusters, _centroid _points.."
for i,c in enumerate(centroids):
    idx = np.argmin(dist(c,data[labels==i]))
    print words[labels==i][idx]
    print words[labels==i]

plt.plot(centroids[:,0],centroids[:,1], 'x')
plt.hold(True)
plt.plot(u[:,0], u[:,1], ' . ')
plt.show()

from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
ax.plot(u[:,0], u[:,1], u[:,2], ' . ', zs=0,
        zdir='z', label='zs=0, _zdir=z')
plt.show()

```



Sonuc

this

['this' 'think']

by

['a' 'I' 'as' 'by' 'my' 'up' 'us']

back

['say' 'all' 'can' 'also' 'back' 'want' 'day']

your

['out' 'about' 'your' 'our']

there

['there' 'their' 'these']

will

['with' 'will' 'which']
 for
 ['of' 'for' 'not' 'you' 'or']
 could
 ['would' 'people' 'could']
 get
 ['at' 'but' 'her' 'get' 'year']
 be
 ['be' 'he' 'we' 'she' 'me' 'see' 'use' 'new']
 like
 ['like' 'time' 'give']
 do
 ['to' 'do' 'so' 'go' 'no' 'two' 'how']
 its
 ['in' 'it' 'his' 'if' 'him' 'into' 'its']
 who
 ['who']
 one
 ['on' 'one' 'only' 'even']
 some
 ['some' 'come']
 when
 ['when' 'well']
 just
 ['just' 'first' 'most']
 what
 ['that' 'what' 'way']
 they
 ['the' 'they' 'them' 'than' 'then']
 good
 ['from' 'know' 'good' 'now' 'look' 'work']
 have
 ['have' 'make' 'take']
 over
 ['other' 'over' 'after']
 because
 ['because']
 any
 ['and' 'an' 'any']

Bu teknigin uygulanabilecegi daha pek cok alan var. Mesela her dokumanin icindeki belli kelimelerin sayilari kolonlarda (her kolon ozel bir kelimeye tekabul edecek sekilde), ve dokumanlari kendisi satirlarda olacak sekilde bir matrisimiz olsaydi, SVD bu matris uzerinde de bir kumeleme icin kullanilabilirdi. Bu ornekte “kac tane kelime oldugu” gibi bir olcut vardir (daha once kelimelerin birbirine uzakligini kullandik), ama teknik yine de ise yarar.

- [1] http://kdd.ics.uci.edu/databases/synthetic_control/synthetic_control.data.html
- [2] <http://sayilarvekuramlar.blogspot.de/2012/07/kelime-benzerligi-levenshtein-mesafesi.html>
- [3] Skillicorn, D., *Understanding Complex Datasets Data Mining with Matrix Decompositions*