Homework2

天津大學

The PCA and KPCA transformations of data

电　　话 13312055621

专　　业 水利水电工程

课程名称 机器学习算法与应用

班　　级 8 班

学　　号 2018205075

姓　　名 董志明

2018 年 10 月 28 日

Abstract

It should be an explicit summary of the report that states the problem, the methods used, the major experiments, discussions and conclusions.

1. Introduction

降维是对数据高维度特征的一种预处理方法。降维是将高维度的数据保留下最重要的一些特征，去除噪声和不重要的特征，从而实现提升数据处理速度的目的。在实际的生产和应用中，降维在一定的信息损失范围内，可以为我们节省大量的时间和成本。降维也成为了应用非常广泛的数据预处理方法。

PCA(principal Component Analysis)，即主成分分析方法，是一种使用最广泛的数据压缩算法。在 PCA 中，数据从原来的坐标系转换到新的坐标系，由数据本身决定。转换坐标系时，以方差最大的方向作为坐标轴方向，因为数据的最大方差给出了数据的最重要的信息。第一个新坐标轴选择的是原始数据中方差最大的方法，第二个新坐标轴选择的是与第一个新坐标轴正交且方差次大的方向。重复该过程，重复次数为原始数据的特征维数。

通过这种方式获得的新的坐标系，我们发现，大部分方差都包含在前面几个坐标轴中，后面的坐标轴所含的方差几乎为 0,。于是，我们可以忽略余下的坐标轴，只保留前面的几个含有绝不部分方差的坐标轴。事实上，这样也就相当于只保留包含绝大部分方差的维度特征，而忽略包含方差几乎为 0 的特征维度，也就实现了对数据特征的降维处理。

2. PCA and KPCA

2.1 Theory of PCA and KPCA

In this section, the background, principle and derivation formula about PCA and KPCA should be described soundly.

PCA 所解决的问题：给定一个由 n 维向量组成的，包含 N 个数据的数据集 X，目标是将其压缩成 p 维(p < n)，同时使得数据集中的主要特征被保留。

假设数据被投影到的方向向量为 v，整个数据集在 v 上投影为:

$$v^T x_1, \dots \dots v^T x_N$$

投影的方差为：

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(v^T x_1)^2$$

将平方写成本身乘以其转置的形式，展开整理得：

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(v^T x_1)^2 = v^T\left(\frac{1}{N}\sum_{i=1}^{N} x_i x_i^T\right)v = v^T C v$$

其中：

$$C = \frac{1}{N}\sum_{i=1}^{N} x_i x_i^T$$

假设：

$$X^T = [x_1, \dots \dots, x_N]$$

从而可知：

$$C = \frac{1}{N} X^T X$$

PCA 的目标是降维，所以我们的目标是找到使得σ^2最大的 v。使用拉格朗日乘子的方法，可以得到下式：

$$f(v,\lambda) = v^T C v - \lambda(c-1)$$

$$\frac{\partial f}{\partial v} = 2Cv - 2\lambda v = 0 => Cv = \lambda v$$

$$\frac{\partial f}{\partial \lambda} = v^T v - 1 = 0 => v^T v = 1$$

从而可以求出 C 矩阵的特征向量和特征值。

最后，计算主成分的贡献率和累计贡献率。

贡献率：

$$\frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}$$

累积贡献率：

$$\frac{\sum_{i=1}^{n} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$

一般取累计贡献率达 85—95%的特征值，即为所对应的第一、第二、…、第 k（k≤n）个主成分。

Kpca：基于 kernel 的 pca。

$$C = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T = \frac{1}{N}\left[\phi(x_1), \cdots, \phi(x_N)\right]\begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}$$

其中：

$$X^T = \left[\phi(x_1), \cdots, \phi(x_N)\right]$$

所以 C 可以被表示为：

$$C = \frac{1}{N} X^T X$$

$$K = \begin{bmatrix} \kappa(x_1, x_1) & \cdots & \kappa(x_1, x_N) \\ \vdots & \ddots & \vdots \\ \kappa(x_N, x_1) & \cdots & \kappa(x_N, x_N) \end{bmatrix}$$

$$= \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}\left[\phi(x_1), \cdots, \phi(x_N)\right] = XX^T$$

假设 K 的特征值和特征向量可以表示成：

$$(XX^T)u = \beta u.$$

两侧同时乘以 X 的转置：

$$X^T(XX^T)u = \beta X^T u \Rightarrow C(X^T u) = \beta(X^T u)$$

这意味着 $X^T u$ 是 C 的特征向量，但是注意到此时 $X^T u$ 的范式不为 1，进行归一化处理：

$$v = \frac{1}{\|X^T u\|} X^T u = \frac{1}{\sqrt{u^T X X^T u}} X^T u$$
$$= \frac{1}{\sqrt{u^T (\beta u)}} X^T u = \frac{1}{\sqrt{\beta}} X^T u,$$

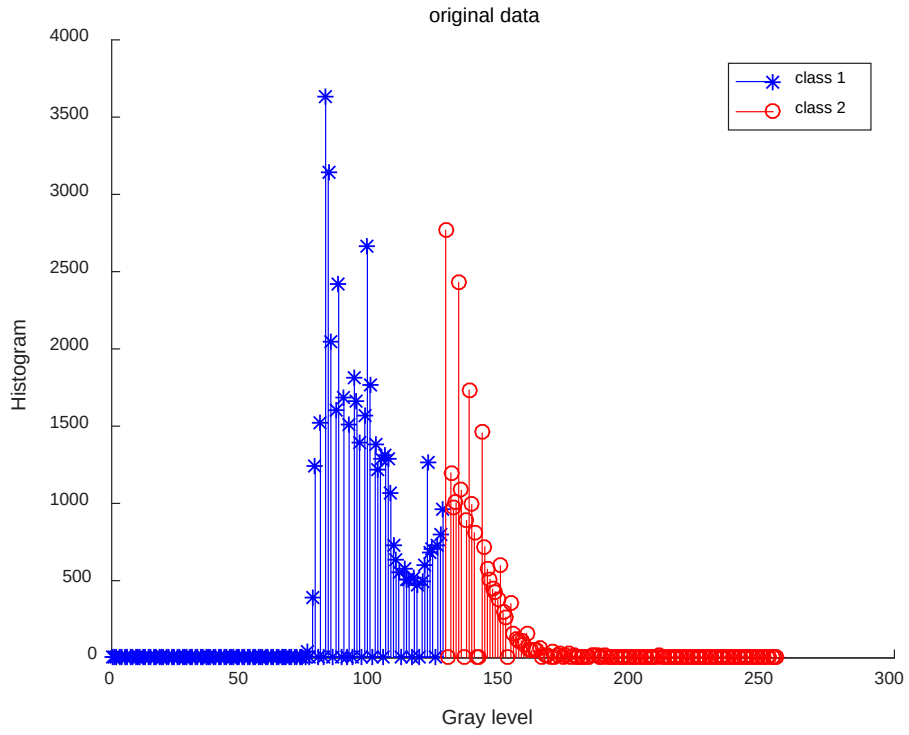
当新的样本来的时候，将样本映射到新的向量上：

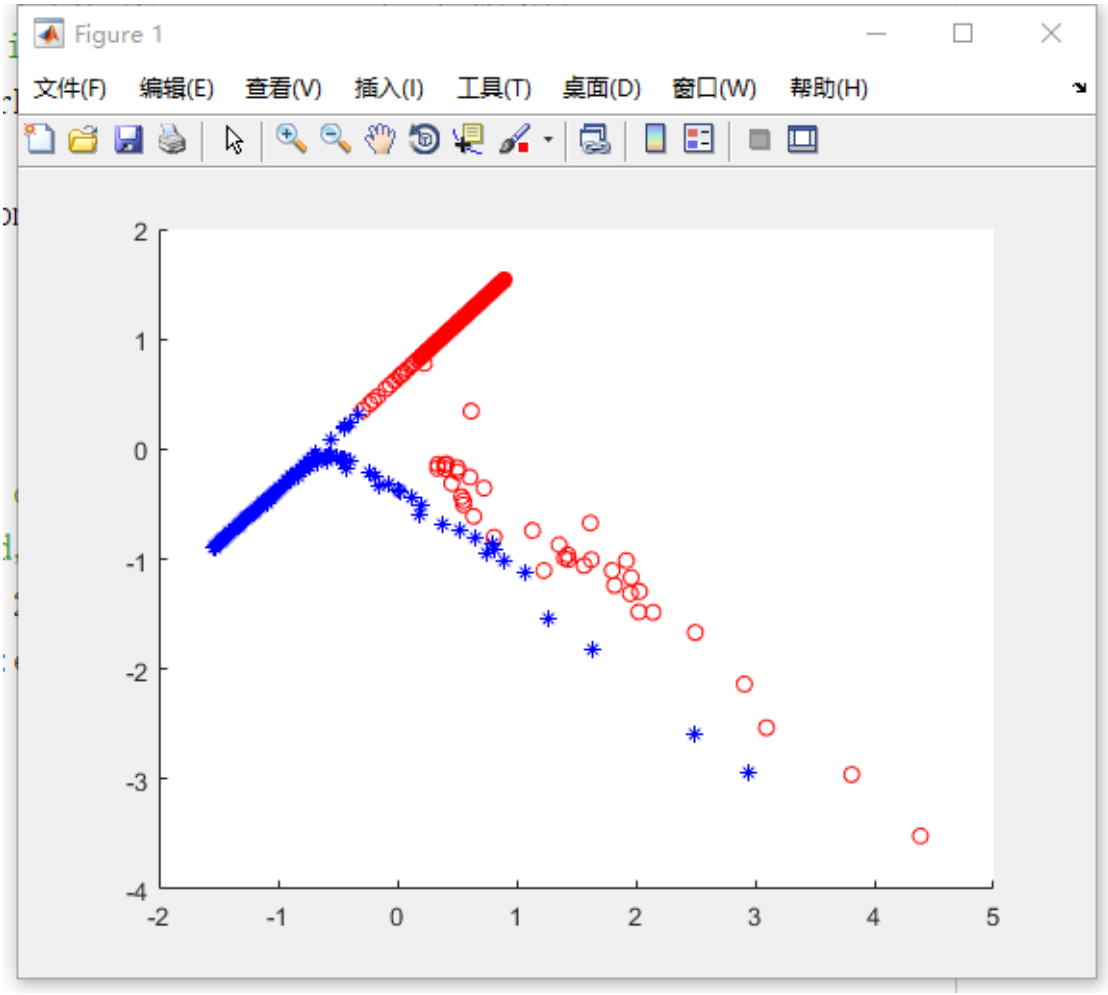
$$v_i^T \phi(x') = \left(\frac{1}{\sqrt{\beta_i}} X^T u_i \right)^T \phi(x') = \frac{1}{\sqrt{\beta_i}} u_i^T X \phi(x')$$
$$= \frac{1}{\sqrt{\beta_i}} u_i^T \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix} \phi(x') = \frac{1}{\sqrt{\beta_i}} u_i^T \begin{bmatrix} \kappa(x_1, x') \\ \vdots \\ \kappa(x_N, x') \end{bmatrix}$$

从而完成降维。

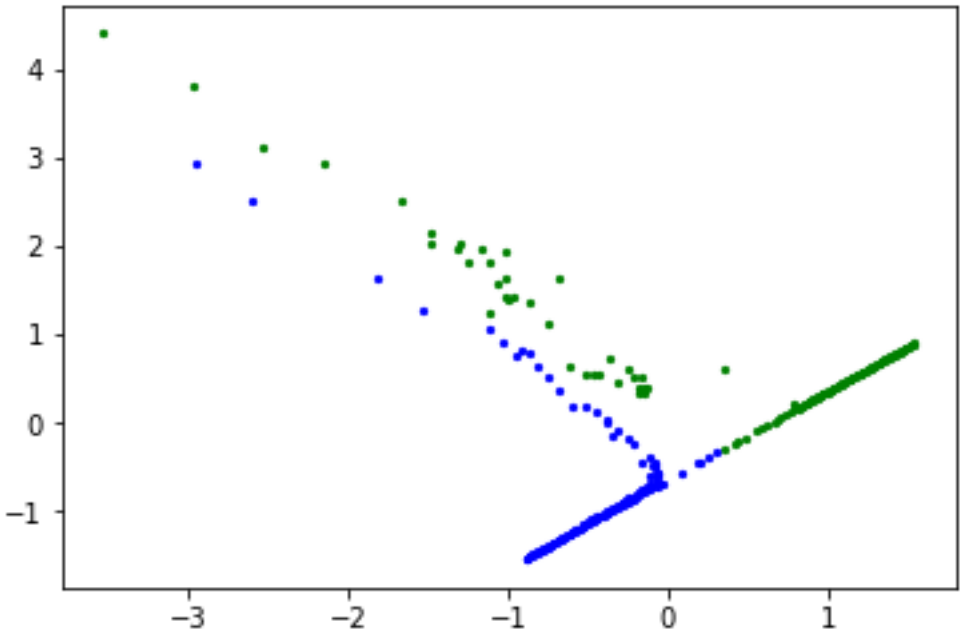
2.2 The PCA and KPCA for "data_PCA.txt"

In this section, an actual project is required to carry out with PCA and KPCA. The steps and results of processing should be given and analyzed in detail. The original date was given as follow, the different data sets are marked as red (1st-128th data in "data_PCA. txt") and blue (129th-256th data in "data_PCA.txt") respectively.

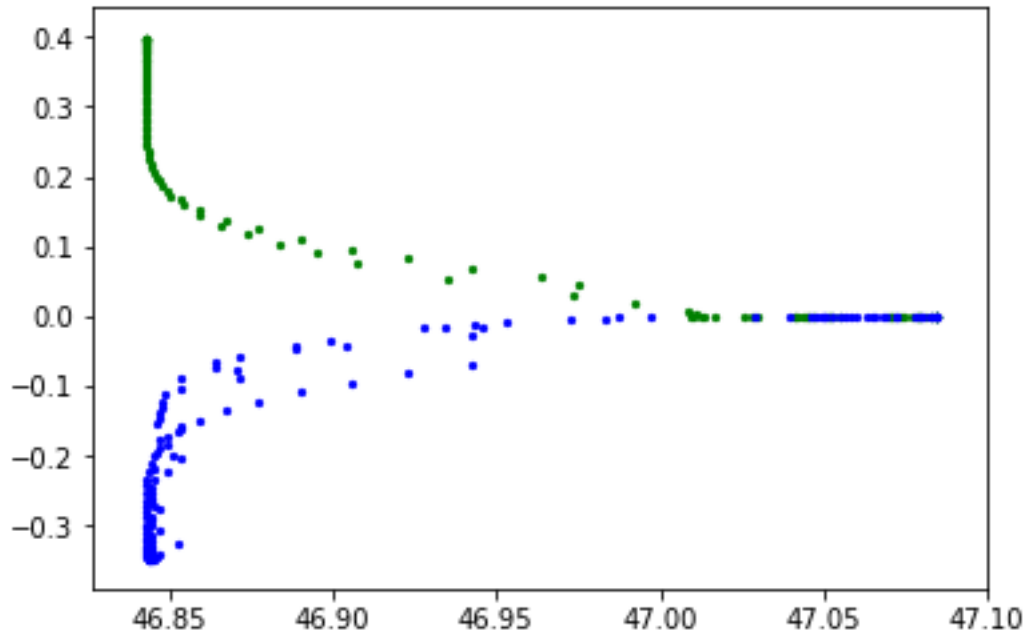




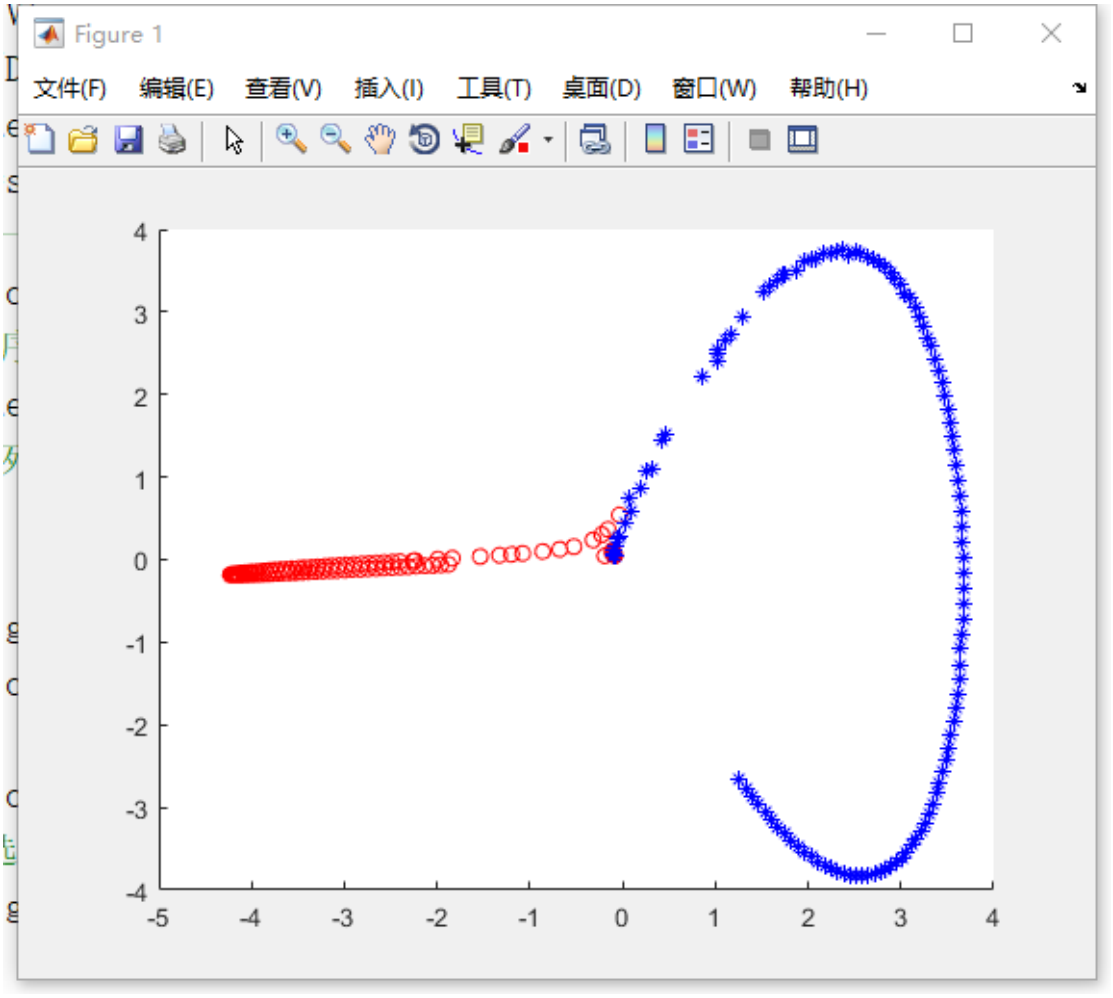
Pca 的 matlab 实现



Pca 的 python 实现



Kpca 的 python 实现



Kpca 的 matlab 实现

由结果可知，kpca 的表现要好于 pca。在 pca 中，由于 pca 是一个降维算法，而作业的目标是从二维降到二维，维度没有变换，所以就相当于旋转了一下坐标系。
在 kpca 中，数据被很好的分成了 2 类，一类近似于一条直线，另外一类是一个圆。但是在用 python 实现的时候，我没有用 sklearn 而是自己写的代码，效果有些差。

3. Conclution

Appendix

Give the used Codes
Specify your environment and how to execute.
PCA：matlab代码

```
clc
clear
dir = 'data_PCA.txt';
data=pcaRead(dir);
%dataOrigin=data;
[dataOrigin,m,sigma]=zscore(data);
%标准化
%Y1=PCA(dataOrigin,2);
d=2;
Sx=cov(dataOrigin);
[V,D]=eig(Sx);
%v是特征向量，D是由特征值组成的对角矩阵
eigValue=diag(D);
%eigValue是从对角矩阵
[eigValue,IX]=sort(eigValue,'descend');
%ix是按照从高到低的顺序排列得到的索引
eigVector=V(:,IX);
%按照特征值给特征向量排序

norm_eigVector=sqrt(sum(eigVector.^2));
eigVectorRepmat=eigVector./repmat(norm_eigVector,size(eigVector,1),1);
%将矩阵A复制m*n块，即把A视为B的元素，B 由 m*n 个A平铺而成。
%B 的维数是 [size(A,1)*m,size(A,2)*n]
```

```
eigVectorRepmatd=eigVectorRepmat(:,1:d);
%d为要选的维数
Y1=dataOrigin*eigVectorRepmatd;


figure;
hold on;
%plot(dataOrigin(1:end,1),dataOrigin(1:end,2),'ro');
%plot(Y1(1:end,1),Y1(1:end,2),'b*');
plot(Y1(1:128,1),Y1(1:128,2),'ro');
plot(Y1(129:end,1),Y1(129:end,2),'b*');
```

结果：

Pca：python实现

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 19 22:24:06 2018

@author: 11854
"""

import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
#读取数据
filename='data_PCA.txt'

data=[]

data_mean=np.zeros((265,2))
data_mapped=np.zeros((265,2))
c=np.zeros((265,2))
with open(filename,'r') as file_to_read:
    while True:
        lines=file_to_read.readline()
        if not lines:
            break
            pass
        p_tmp=[float(i) for i in lines.split(',')]
        data.append(p_tmp)
        pass
    data=np.array(data)
    pass
```

```
#修改数据使得u=0
#data_mean=data-np.mean(data,axis=0)
data_mean=preprocessing.scale(data)
#计算c矩阵
c=data_mean.T.dot(data_mean)
#c=np.cov(data_mean.T)

#求c的特征值和特征向量
eigenvalue,featurevector=np.linalg.eig(c)
#featurevector=featurevector.T
#featurevector的每一行为一个特征向量
#featurevector=[[-0.7071067812,-0.7071067812],[0.7071067812,-0.7071067812]]

#按照特征值大小给特征向量排序
eigenvalue_sort=np.argsort(-eigenvalue)
#按照降序排列，返回索引
featurevector_sort=featurevector[eigenvalue_sort]

#映射到新的空间
data_mapped=data_mean.dot(featurevector_sort)

#画图
print('pca变换前')
plt.scatter(data[:,0], data[:,1], s=5)
plt.show()
print('pca变换后')
#plt.scatter(data_mapped[:,0], data_mapped[:,1], s=5)
plt.scatter(data_mapped[0:127,0], data_mapped[0:127,1], s=5,c='g')
plt.scatter(data_mapped[128:255,0], data_mapped[128:255,1], s=5,c='b')
plt.show()
```

KPCA：python实现

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 21 15:32:30 2018

@author: 11854
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
```

```
import math
#读取数据
filename='data_PCA.txt'

data=[]
featurevector_sort_normal=np.zeros((256,2))
data_mean=np.zeros((256,2))
data_mapped=np.zeros((256,2))
distance_maxtrix_sigma=np.zeros((256,256))
c=np.zeros((256,2))
with open(filename,'r') as file_to_read:
    while True:
        lines=file_to_read.readline()
        if not lines:
            break
            pass
        p_tmp=[float(i) for i in lines.split(',')]
        data.append(p_tmp)
        pass
    data=np.array(data)
    pass



#修改数据使得u=0
data_mean=preprocessing.scale(data)
#data_mean=data-np.mean(data,axis=0)
#data_mean=data_mean.T

#计算k矩阵
#K=data_mean.T.dot(data_mean)
#K=kernel(data_mean)
nun_of_data=data_mean.shape[0]
distance_maxtrix=np.zeros([nun_of_data,nun_of_data])
K=np.zeros([nun_of_data,nun_of_data])
#k是kernel矩阵
x=np.sum(np.square(data_mean),axis=1)
distance_maxtrix=np.add(x.reshape(1,len(x)),x.reshape(len(x),1))+np.multiply(-
2,np.dot(data_mean,data_mean.T))
distance_maxtrix_sigma=np.copy(distance_maxtrix)
distance_maxtrix_sigma[distance_maxtrix_sigma==0]=float('inf')
sigma=5*np.mean(distance_maxtrix_sigma.min(axis=0))
sigma_kernel=1/np.square(sigma)/2
```

```
#算距离矩阵
K0=np.exp(-distance_maxtrix*sigma_kernel)
oneN=np.ones_like(K0)
K=K0-oneN.dot(K0)-K0.dot(oneN)+(oneN.dot(K0)).dot(oneN)
#计算K的特征值特征矩阵
eigenvalue,featurevector=np.linalg.eig(K)
featurevector=featurevector.real
eigenvalue=eigenvalue.real
eigenvalue_index=np.argsort(-eigenvalue)
eigenvalue_index=eigenvalue_index[0:2]
#按照降序排列，返回索引
eigenvalue_sort=eigenvalue[eigenvalue_index[0:2]]
featurevector_sort=featurevector[:,eigenvalue_index[0:2]]

for i in range(len(eigenvalue_index)):
    featurevector_sort_normal[:,i]=featurevector_sort[:,i]/np.sqrt(eigenvalue_sort[i])



data_mapped=K.dot(featurevector_sort_normal)
print('pca变换后')
plt.scatter(data_mapped[0:127,0], data_mapped[0:127,1], s=5,c='g')
plt.scatter(data_mapped[128:255,0], data_mapped[128:255,1], s=5,c='b')
plt.show()
```

KPCA：matlab代码

```
clc
clear
dir = 'data_PCA.txt';
data=pcaRead(dir);
[data,m,sigma]=zscore(data);
N=size(data,1);
disp('开始计算');


%N为样本的个数
K0=kernel(data);
%K0=data*data';
oneN=ones(N,N)/N;
K=K0-oneN*K0-K0*oneN+oneN*K0*oneN;
%K=K0;
```

```
%K即为书上的K矩阵

%获取特征值和特征向量
[V,D]=eig(K/N);
V=real(V);
D=real(D);
eigValue=diag(D);
[~,IX]=sort(eigValue,'descend');
%忽略第一个参数
eigVector=V(:,IX);
%重新排序V特征向量组成的矩阵
eigValue=eigValue(IX);
%重新排列特征值

%标准化
norm_eigVector=sqrt(sum(eigVector.^2));
eigVector=eigVector./repmat(norm_eigVector,size(eigVector,1),1);

eigVector=eigVector(:,1:2);
%d为要选的维数
Y3=K*eigVector;
figure;
hold on;
%plot(Y3(1:end,1),Y3(1:end,2),'b*');
plot(Y3(1:128,1),Y3(1:128,2),'ro');
plot(Y3(129:end,1),Y3(129:end,2),'b*');
drawnow;
```

kernel函数：

```
function K=kernel(X)
% Gaussian Kernel
% Input:
% X: 输入的数据矩阵，每一行是一个数据，行数是数据总数
% sigma_kernel: 径向基核函数的参数
% Output:
% K: X的kernel矩阵，大小为N*N

N = size(X,1); %行数，即为数据个数
DIST=zeros(N,N);
for i=1:N
    for j=1:N
        diff=X(i,:)-X(j,:);
        DIST(i,j)=sqrt(diff*diff');
```

```
        end
    end
    DIST(DIST==0)=inf;
    %inf为无穷大
    DIST1=min(DIST);
    sigma_kernel=5*mean(DIST1);
    K = zeros(N);
    for i = 1:N
        for j = i:N
            diff_ij = X(i,:)-X(j,:);
            K(i,j) = exp(-diff_ij*diff_ij'/2/sigma_kernel^2);
            %得到一个数
            if i~=j
                K(j,i) = K(i,j);
            end
        end
    end
```