**模式识别实验**

基于IRIS数据集的PCA、KPCA和LDA特征变换

3115315017

高榕

# 一．实验要求

1. 学习PCA，KPCA，LDA变换的基本概念；
2. 利用PCA，KPCA，LDA三种变换对Iris数据进行降维并对分类；

# 二．实验原理

## PCA变换

1. 特征选择,即从个度量值集合中，按某一准则选出供分类用的子集，作为降维（维, ）的分类特征。特征提取即使通过某种变换(.)，,而产生个特征。其目的是：在尽可能保留识别信息的前提下，降低特征空间的维数，以达到有效分类。变换就是一种有效的特征变换方法。
2. 变换又称为变换，特征向量变换或主分量方法，将其用于统计特征提取，从而形成了子空间法模式识别的基础。变换数学定义：存在矩阵：  
    

对于一般的线性变换Y=TX，如果变换矩阵T是正交矩阵，并且它是由原始数据矩阵X的斜方差矩阵S的特征向量所组成，则此式的变换称为K-L变换。

1. 变换的具体过程如下：

第一步，根据原始数据矩阵X，求出它的协方差矩阵S，X的协方差矩阵为：

式中：， ；（即为第个波段的均值）；；S是一个实对称矩阵。  
第二步，求S矩阵的特征值λ和特征向量U，并组成变换矩阵T。 用eig函数求出协方差矩阵S的各个特征值及各特征值对应的单位特征向量（经归一化）： ，按排列。  
以各特征方量为列构成矩阵，即

U矩阵的转置矩阵即为所求的K-L变换的变换矩阵T。  
有了变换矩阵T，将其代入Y=TX，则：

  
式中Y矩阵的行向量 为第j主成分。

第三步,用基于单个神经元的线性分类器对Y进行分类。

## KPCA变换

在Kernel PCA分析之中，我们假设原有数据有更高的维数，我们可以在更高维的空间（Hilbert Space）中做PCA分析（即在更高维空间里，把原始数据向不同的方向投影）。通过PCA，我们得到了U矩阵。这里将介绍如何仅利用内积的概念来计算传统的PCA。

首先我们证明U可以由,,…, 展开：













这里定义。因为是一个标量，所以也是一个标量，因此是可以由张成的。

进而我们显示PCA投影可以用内积运算表示，例如我们把向任意一个主成分分量进行投影，得到的是，也就是。







当我们定义时，上式可以写成。进一步，我们得到解为：



K矩阵包含特征值和，我们可以通过计算得到，注意特征值分解时，只代表一个方向。它的长度一般为1，但在此处不为1。

投影时，只需要使用U矩阵，假设我们得到的新数据为t，那么t在方向上的投影是：



对于高维空间的数据，，我们可以用kernel trick，用带入上式：



在我们的分析中，协方差矩阵的定义需要centered data。在高维空间中，显示的将居中并不简单，因为我们并不知道的显示表达。但从上面可以看出，所有的计算只和K矩阵有关。具体计算如下：

令，居中







不难看出，



其中为的矩阵，其中每一个元素都是。

对于新数据，我们同样可以：



## LDA变换

线性判别式分析（Linear Discriminant Analysis），简称为LDA。也称为Fisher线性判别（Fisher Linear Discriminant，FLD），是模式识别的经典算法，在1996年由Belhumeur引入模式识别和人工智能领域。

基本思想是将高维的模式样本投影到最佳鉴别矢量空间，以达到抽取分类信息和压缩特征空间维数的效果，投影后保证模式样本在新的子空间有最大的类间距离和最小的类内距离，即模式在该空间中有最佳的可分离性。

LDA与前面介绍过的PCA都是常用的降维技术。PCA主要是从特征的协方差角度，去找到比较好的投影方式。LDA更多的是考虑了标注，即希望投影后不同类别之间数据点的距离更大，同一类别的数据点更紧凑。

用原来的特征表示的数据记作，提取的特征表示的数据记作，W是的矩阵。沿用Fisher判别别函数中的记号，假设共有k类：

类间散度矩阵： 

类内总散度矩阵：

用提取的m个特征表示的数据的类内、类间散度矩阵记作：





我们希望求取W，是的类内距离小，类间距离大，即：



求解后，矩阵的前m个特征值对应的特征向量即可组成W。

对于k类问题，选出的特征的个数做多只有k-1个，这是因为的秩最多为k-1。因此，对应非零特征根的特征向量做多只有k-1，那些非零特征根对应的特征向量对判据J的值没有任何影响。

# 三．实验过程、结果及分析

## 利用PCA变换对Iris数据集聚类

从第二次实验我们知道，单个神经元经过多次迭代后可以将三类数据的两两组合完全分开，本实验中，为了证明K-L变换对分类的有效性，我们在训练单神经元分类器时只迭代1次，并保持每次实验参数相同。通过改变降维的参数和每次参与训练的样本数，可得到如下结果：

**数据集1和数据集2：**

**表1** **特征向量不变时的原始精度**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 100% | 100% | 100% | 100% | 100% |

**表2 特征向量由4维降为3维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 100% | 100% | 100% | 100% | 100% |

**表3 特征向量由4维降为2维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 100% | 100% | 100% | 100% | 100% |

**表4 特征向量由4维降为1维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 99% | 100% | 100% | 100% | 100% |

**数据集1和数据集3：**

**表5 特征向量不变时的原始精度**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 100% | 100% | 100% | 100% | 100% |

**表6 特征向量由4维降为3维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 100% | 100% | 100% | 100% | 100% |

**表7 特征向量由4维降为2维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 100% | 100% | 100% | 100% | 100% |

**表8 特征向量由4维降为1维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 100% | 100% | 100% | 100% | 100% |

**数据集2和数据集3：**

**表9 特征向量不变时的原始精度**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 87% | 95% | 96% | 97% | 98% |

**表10 特征向量由4维降为3维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 92% | 96% | 95% | 96% | 96% |

**表11 特征向量由4维降为2维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 93% | 95% | 95% | 97% | 96% |

**表12 特征向量由4维降为1维**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 每类训练样本数 | 5 | 15 | 25 | 35 | 45 |
| 分类精度 | 87% | 90% | 88% | 90% | 90% |

**结论：**从实验结果可以看出：

1. 数据集1和2以及数据集1和3是比较容易分开的，经过K-L变换降维后，数据依然是可分的，所以K-L变换将特征向量降维后，数据的可分性依然不变；
2. 数据集2和3比较难分，用原始的数据不能成功将其完全分开；但经过K-L变换降维后，分类精度没有太大损失；在训练数据较少时，分类精度还会比原来更高；
3. 从（1）和（2）可以得到，K-L变换可以从原始特征数据中抽出对分类重要的维度，在不损失分类精度的情况下，能大大减少数据的维度，节省了运算时间和存储空间，有时甚至还能提高分类精度；
4. 从实验中可以看出，训练样本数越多，分类精度越高，这符合我们的期望。

## 利用KPCA变换对Iris数据集聚类

本实验中，为了证明KPCA变换对分类的有效性，我们在训练单神经元分类器时只迭代1次，并保持每次实验参数相同。为了控制可调参数的数量，我们用所有样本计算RPCA的变换矩阵，在分类时，每次选取的样本训练数为25，通过改变降维的参数和kernel函数，可得到如下结果（行方向代表将特征向量降为相应的维数，列方向代表将参与分类的数据集）：

**表13 采用Gaussian kernel KPCA降维后在不同数据集上的分类精度**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 3 | 2 | 1 |
| 数据集1和2 | 100% | 100% | 100% |
| 数据集1和3 | 100% | 100% | 100% |
| 数据集2和3 | 95% | 86% | 88% |

**表14 采用Polynomial kernel KPCA降维后在不同数据集上的分类精度**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 3 | 2 | 1 |
| 数据集1和2 | 100% | 100% | 100% |
| 数据集1和3 | 100% | 100% | 100% |
| 数据集2和3 | 95% | 94% | 84% |

**表15 采用PolyPlus kernel KPCA降维后在不同数据集上的分类精度**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 3 | 2 | 1 |
| 数据集1和2 | 100% | 100% | 100% |
| 数据集1和3 | 100% | 100% | 100% |
| 数据集2和3 | 95% | 94% | 84% |

**结论：**从实验结果可以看出：

1. 对数据集1和2以及数据集1和3，经过KPCA变换降维后，数据依然是可分的；对数据集2和3，经过KPCA变换降维后，分类精度变化不大；
2. 从三个表格的对比可以看出，不同的kernel函数，所得的分类精度是不同的，但对本实验来说，由于数据量小，所以之间的差别是略微的；
3. 从（1）可以得到，KPCA变换可以从原始特征数据中抽出对分类重要的维度，在不损失分类精度的情况下，能大大减少数据的维度，节省了运算时间和存储空间。

## 利用LDA变换对Iris数据集聚类

同前两组实验，我们在训练单神经元分类器时只迭代1次，并保持每次实验参数相同。因为LDA降维后的维数为k-1维（k为类别数目），本实验如果参与分类的数据有两类时，降维后的维度只有1；参与分类的类别数有3类时，降维后的维度为2。实验结果如下：

**表16 采用LDA降维后在不同数据集上的分类精度**

|  |  |  |
| --- | --- | --- |
|  | 降维后的维度 | 分类精度 |
| 数据集1和2 | 1 | 100% |
| 数据集1和3 | 1 | 100% |
| 数据集2和3 | 1 | 98% |
| 数据集1 2 3 | 2 | 100% |

**结论：**从实验结果可以看出：

1. 和前两种方法相比较，LDA的分类精度是最高的，特别是对数据集1和3的分类，虽然降维后的特征向量只有1维，但分类精度却达到了98%；对三个数据集同时分类时，精度也达到了惊人的100%；
2. LDA变换在求取时，用到了类别信息，使得类间距离最大，类内距离最大，所以能够取得比前两种方法更好的结果。

# 四．三种变换的简要分析

1. PCA是选用一组正交基来近似的表示原始数据，以达到数据降维的效果；每一个主成分都是数据在某一个方向上的投影，在不同的方向上这些数据方差Variance的大小由其特征值（eigenvalue）决定。
2. 在KPCA中，我们认为原有数据有更高的维数，我们可以在更高维的空间（Hilbert Space）中做PCA分析（即在更高维空间里，把原始数据向不同的方向投影）。这样做的优点是：对于在通常线性空间难于线性分类的数据点，我们有可能再更高维度上找到合适的高维线性分类平面。
3. PCA是一种unsupervised的映射方法，而LDA是一种supervised映射方法PCA只是将整组数据整体映射到最方便表示这组数据的坐标轴上，映射时没有利用任何数据内部的分类信息。LDA变换在求取时，用到了类别信息，使得类间距离最大，类内距离最大，所以能够取得比前两种方法更好的结果。

# 五．源代码

clear;clc

close all;

% load data

% data1 = xlsread('数据\data1.xls');

% data2 = xlsread('数据\data2.xls');

% data1 = xlsread('数据\data1.xls');

% data2 = xlsread('数据\data3.xls');

data1 = xlsread('数据\data2.xls');

data2 = xlsread('数据\data3.xls');

% classify with PCA

dim = 3; % dim of picked after K-L conversion

for i = 5:10:50

num\_train = i;

% convert data with K-L conversion

[data1, data2] = K\_L(data1, data2, num\_train, dim);

% classification with single-layer perceptron

accu = perceptron\_LMS(data1, data2, num\_train);

fprintf('accurary:%f\n', accu);

end

% classify with RPCA

opts.KernelType = 'PolyPlus'; % 'Gaussian' 'Polynomial' 'PolyPlus'

for i = 3:-1:1

data = [data1; data2];

dim = i;

[eigvector1, eigvalue1, Y] = KPCA(data', dim, opts);

num\_train = 25;

accu = perceptron\_LMS(Y(:,1:50)', Y(:,51:100)', num\_train);

fprintf('accurary:%f\n', accu);

end

% classify with LDA

data = [data1; data2];

N = [length(data1) length(data2)];

reduced\_data = LDA\_my(data, N);

num\_train = 25;

accu = perceptron\_LMS(reduced\_data(1:50,:), reduced\_data(51:100,:), num\_train);

fprintf('accurary:%f\n', accu);

**function [data1, data2] = K\_L(data1, data2, num\_train, dim)**

%

X1 = data1(1:num\_train,:);

X2 = data2(1:num\_train,:);

X\_train = [X1; X2];

cov\_x = cov(X\_train,1);

[eigen\_vector,eigen\_value] = eig(cov\_x);

e\_value = diag(eigen\_value);

[~,index] = sort(e\_value,'descend');

e\_vector = eigen\_vector(:,index);

U = e\_vector(:, 1:dim);

data1 = data1 \* U;

data2 = data2 \* U;

**function [eigvector, eigvalue,Y] = KPCA(X,r,opts)**

%

% Kernel Principal Component Analysis

% [eigvector, eigvalue,Y] = KPCA(X,r,opts)

% Input:

% X: d\*N data matrix;Each column vector of X is a sample vector.

% r: Dimensionality of reduced space (default: d)

% opts: Struct value in Matlab. The fields in options that can be set:

% KernelType - Choices are:

% 'Gaussian' - exp{-gamma(|x-y|^2)}

% 'Polynomial' - (x'\*y)^d

% 'PolyPlus' - (x'\*y+1)^d

% gamma - parameter for Gaussian kernel

% d - parameter for polynomial kernel

%

% Output:

% eigvector: N\*r matrix;Each column is an embedding function, for a new

% data point (column vector) x, y = eigvector'\*K(x,:)'

% will be the embedding result of x.

% K(x,:) = [K(x1,x),K(x2,x),...K(xN,x)]

% eigvalue: The sorted eigvalue of KPCA eigen-problem.

% Y : Data matrix after the nonlinear transform

if nargin<1

error('Not enough input arguments.')

end

[d,N]=size(X);

if nargin<2

r=d;

end

%% Ensure r is not bigger than d

if r>d

r=d;

end;

% Construct the Kernel matrix K

K =ConstructKernelMatrix(X,[],opts);

% Centering kernel matrix

One\_N=ones(N)./N;

Kc = K - One\_N\*K - K\*One\_N + One\_N\*K\*One\_N;

clear One\_N;

% Solve the eigenvalue problem N\*lamda\*alpha = K\*alpha

if N>1000 && r

% using eigs to speed up!

opts.disp=0;

[eigvector, eigvalue] = eigs(Kc,r,'la',opts);

eigvalue = diag(eigvalue);

else

[eigvector, eigvalue] = eig(Kc);

eigvalue = diag(eigvalue);

[junk, index] = sort(-eigvalue);

eigvalue = eigvalue(index);

eigvector = eigvector(:,index);

end

if r < length(eigvalue)

eigvalue = eigvalue(1:r);

eigvector = eigvector(:, 1:r);

end

% Only reserve the eigenvector with nonzero eigenvalues

maxEigValue = max(abs(eigvalue));

eigIdx = find(abs(eigvalue)/maxEigValue < 1e-6);

eigvalue (eigIdx) = [];

eigvector (:,eigIdx) = [];

% Normalizing eigenvector

for i=1:length(eigvalue)

eigvector(:,i)=eigvector(:,i)/sqrt(eigvalue(i));

end;

if nargout >= 3

% Projecting the data in lower dimensions

Y = eigvector'\*K;

end

function K=ConstructKernelMatrix(X,Y,opts)

%

if nargin<1

error('Not enough input arguments.')

end

if (~exist('opts','var'))

opts = [];

else

if ~isstruct(opts)

error('parameter error!');

end

end

N=size(X,2);

if isempty(Y)

K=zeros(N,N);

else

M=size(Y,2);

if size(X,1)~=size(Y,1)

error('Matrixes X and Y should have the same row dimensionality!');

end;

K=zeros(N,M);

end;

%=================================================

if ~isfield(opts,'KernelType')

opts.KernelType = 'Gaussian';

end

switch lower(opts.KernelType)

case {lower('Gaussian')} % exp{-gamma(|x-y|^2)}

if ~isfield(opts,'gamma')

opts.gamma = 0.5;

end

case {lower('Polynomial')} % (x'\*y)^d

if ~isfield(opts,'d')

opts.d = 1;

end

case {lower('PolyPlus')} % (x'\*y+1)^d

if ~isfield(opts,'d')

opts.d = 1;

end

otherwise

error('KernelType does not exist!');

end

switch lower(opts.KernelType)

case {lower('Gaussian')}

if isempty(Y)

for i=1:N

for j=i:N

dist = sum(((X(:,i) - X(:,j)).^2));

temp=exp(-opts.gamma\*dist);

K(i,j)=temp;

if i~=j

K(j,i)=temp;

end;

end

end

else

for i=1:N

for j=1:M

dist = sum(((X(:,i) - Y(:,j)).^2));

K(i,j)=exp(-opts.gamma\*dist);

end

end

end

case {lower('Polynomial')}

if isempty(Y)

for i=1:N

for j=i:N

temp=(X(:,i)'\*X(:,j))^opts.d;

K(i,j)=temp;

if i~=j

K(j,i)=temp;

end;

end

end

else

for i=1:N

for j=1:M

K(i,j)=(X(:,i)'\*Y(:,j))^opts.d;

end

end

end

case {lower('PolyPlus')}

if isempty(Y)

for i=1:N

for j=i:N

temp=(X(:,i)'\*X(:,j)+1)^opts.d;

K(i,j)=temp;

if i~=j

K(j,i)=temp;

end;

end

end

else

for i=1:N

for j=1:M

K(i,j)=(X(:,i)'\*Y(:,j)+1)^opts.d;

end

end

end

otherwise

error('KernelType does not exist!');

end

**function reduced\_data = LDA\_my(data, N)**

% data=[2.95 6.63; 2.53 7.79; 3.57 5.65;3.16 5.47;2.58 4.46; 2.16 6.22; 3.27 3.52];

% N=[4 3];

C = length(N);

dim = size(data,2);

pos = zeros(C,2);

% 计算每类样本在data中的起始、终止行数

for i=1:C

START=1;

if i>1

START=START+sum(N(1:i-1));

end

END=sum(N(1:i));

pos(i,:)=[START END];

end

% 每类样本均值

UI=[];

for i=1:C

if pos(i,1)==pos(i,2)

% pos(i,1)==pos(i,2)时，mean函数不能工作

UI=[UI;data(pos(i,1),:)];

else

UI=[UI;mean(data(pos(i,1):pos(i,2),:))];

end

end

% 总体均值

U=mean(data);

% 类间散度矩阵

SB=zeros(dim,dim);

for i=1:C

SB=SB+N(i)\*(UI(i,:)-U)'\*(UI(i,:)-U);

end

% 类内散度矩阵

SW=zeros(dim,dim);

for i=1:C

for j=pos(i,1):pos(i,2)

SW=SW+(data(j,:)-UI(i,:))'\*(data(j,:)-UI(i,:));

end

end

SW = SW/C;

SB = SB/C;

% 计算特征值与特征向量

matrix = pinv(SW)\*SB;

[V,D] = eig(matrix);

e\_value = diag(D);

[~,index] = sort(e\_value,'descend');

e\_vector = V(:,index);

W = e\_vector(:, 1:C-1);

% 根据新的特征向量，将数据映射到新空间

reduced\_data = data\*W;

**function [accu] = perceptron\_LMS(data1, data2, num\_train)**

% single-layer perceptron with LMS

N1 = size(data1,1);

N2 = size(data2,1);

N = N1 + N2;

data = [data1; data2];

% trainIdx1 = randsample(N1, num\_train);

% trainIdx2 = randsample(N2, num\_train);

data1\_train = data1(1:num\_train,:);

data2\_train = data2(1:num\_train,:);

data\_train = [data1\_train; data2\_train];

dim = size(data, 2);

R = zeros(dim+1, dim+1);

P = zeros(1,dim+1);

for i = 1:num\_train

R = R + [data\_train(i,:) 1]'\*[data\_train(i,:) 1];

P = P + 1\*[data\_train(i,:) 1];

end

for i = (num\_train+1):(2\*num\_train)

R = R + [data\_train(i,:) 1]'\*[data\_train(i,:) 1];

P = P + (-1)\*[data\_train(i,:) 1];

end

R = R./(2\*num\_train);

P = P./(2\*num\_train);

W\_opt = P/R;

data(:,end+1) = ones(N, 1);

score = data \* W\_opt';

accu = (sum(score(1:N1)>0) + sum(score(N1+1:N1+N2)<0))/N;