ECE 662

Hw2

4/1/2008

1. Fisher's Linear Discriminant Analysis

We would like to find a projection from R^n to R^1, such that we can maximize the separation between 2 classes.

We define the cost function,

$$J(w) = \frac{w^T S_B w}{w^T S_w w}$$

where

$$m_1 = \frac{1}{d_1} \sum_{wc \in c_1} \bar{x}, \; m_2 = \frac{1}{d_2} \sum_{wc \in c_2} \bar{x}$$

$$S_B = (m_1 - m_2)(m_1 - m_2)^T$$

$$S_w = \sum_{i=1}^{d_1} (x_i - m_1)(x_i - m_1)^T + \sum_{i=1}^{d_2} (x_i - m_2)(x_i - m_2)^T$$

The solution was found to be $w_{fisher} = S_w^{-1}(m_1 - m_2)$

If we choose the cost function as

$$J(w) = w^T S_B w$$

The solution is $w_{meandiff} = (m_1 - m_2)$

I'd like to investigate this problem with 2 Gaussian distributions with dimension 2.

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mu_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$

Each class contains 1000 data points.

The distribution of 2 data

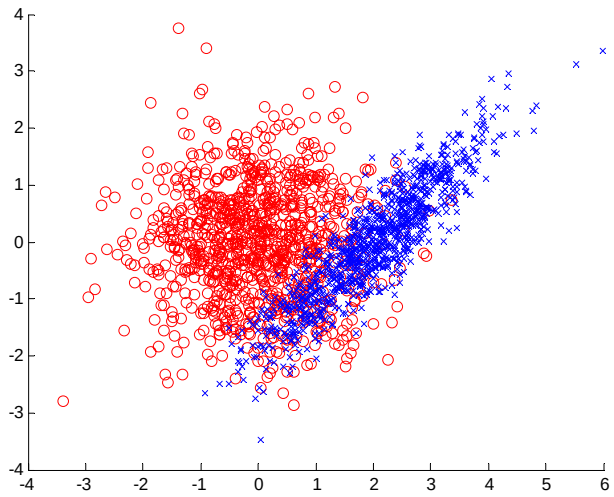


Figure 1: Scatter plot of 2 data with different mean can covariance. Class 1 is labeled with red circle. Class 2 is labeled with blue cross.

In order to compare 2 projections fairly, w_{fisher} and $w_{meandiff}$ are normalized with norm equal to 1.

$$w_{fisher} = \begin{bmatrix} -0.9114 \\ 0.4116 \end{bmatrix},\ w_{meandiff} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

The projected data would be $w\vec{x}$, which is in R^1.

The histogram of the projected data when I use w_{fisher}.

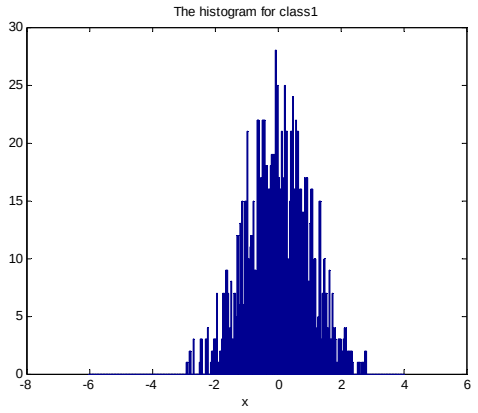
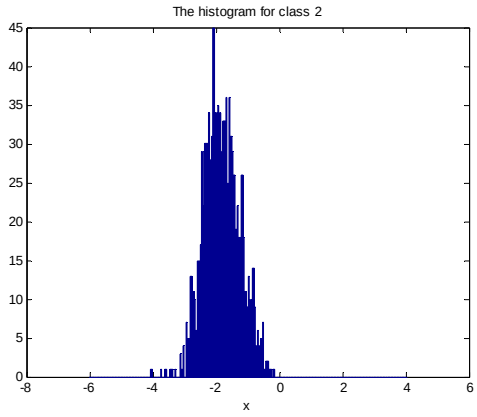
Class1	Class2
	

Figure 2: The histograms of 2 projected data.

Since it's hard to do the comparison in histogram, I use kernel density estimation to estimate the density for 2 classes.

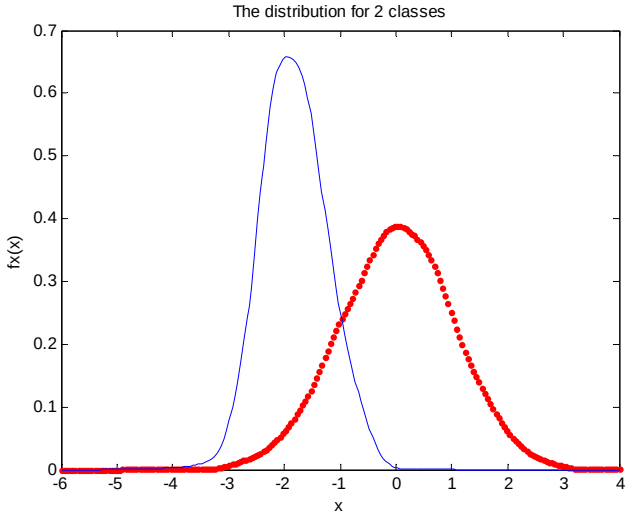


Figure 3: The estimated density for the projected data by using w_{fisher}.

The density of the projected data when I use $w_{meandiff}$ for projection is plotted in Figure 4.

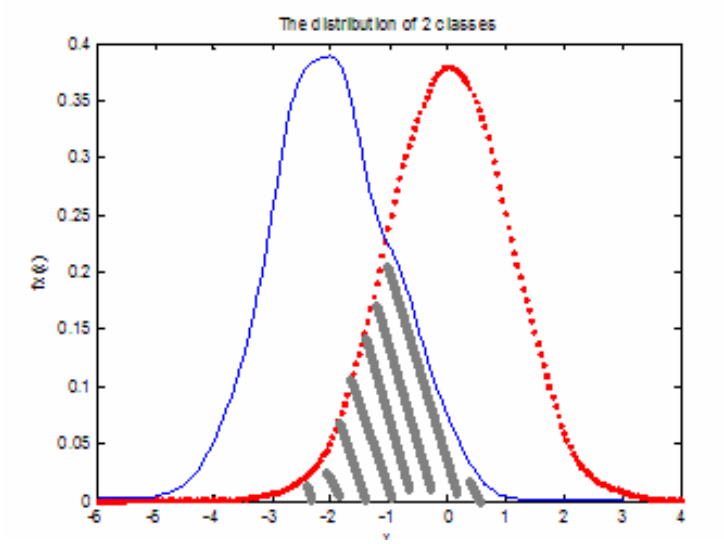


Figure 4: The estimated density for the projected data by using $w_{meandiff}$ .The gray area is the misclassification area, which is called the area of error.

In order to compare which projection performs better, I define the Area of error which is the shaded area in Figure 4. The smaller the Area of error gives the better separation.

Table 1: Comparison between the areas of error for 2 different kind of projections.

	Project onto w_{fisher}	Project onto $w_{meandiff}$
Area of error	0.2569	0.3417

I compute the area by using Simpson's quadrature rule. (a kind of numerical integration)

Furthermore, I design the dataset which shows the S_w really can help us find the best separation hyperplane.

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \mu_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$

The distribution of 2 data

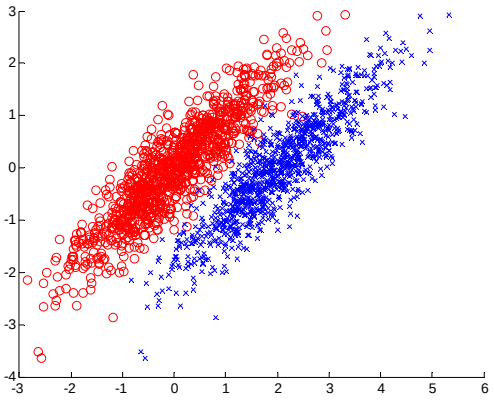


Figure 5: Scatter plot of 2 data with different mean can covariance. Class 1 is labeled with red circle. Class 2 is labeled with blue cross.

$$w_{fisher} = \begin{bmatrix} -0.7433 \\ 0.669 \end{bmatrix},\ w_{meandiff} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

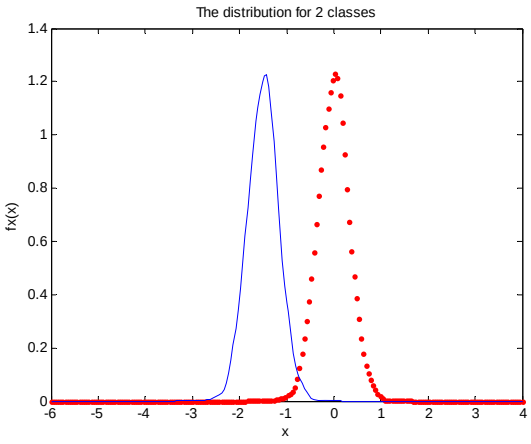
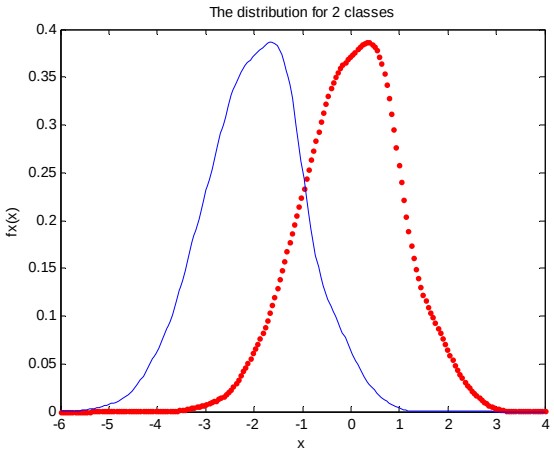
Project onto w_{fisher}	Project onto $w_{meandiff}$
	
Area of error : 0.0275	Area of error : 0.3272

Figure 6: The estimated density for the projected data by using w_{fisher} is on the left panel. The estimated density for the projected data by using $w_{meandiff}$ is on the right panel.

From above 2 examples, I know that $w_{fisher} = S_w^{-1}(m_1 - m_2)$ can give us better separation. This also considers the within class scatter matrix. If the data for one class is thinner in one direction than the other direction, we should include this information to help us to get better separable projection.

2. Neural Network and Support Vector Machine

I download breast cancer wisconsin data set from UCI machine learning repository.
http://archive.ics.uci.edu/ml/
The description of the attributes are as follows:

#	Attribute	Domain
1.	Sample code number	id number
2.	Clump Thickness	1 - 10
3.	Uniformity of Cell Size	1 - 10
4.	Uniformity of Cell Shape	1 - 10
5.	Marginal Adhesion	1 - 10
6.	Single Epithelial Cell Size	1 - 10
7.	Bare Nuclei	1 - 10
8.	Bland Chromatin	1 - 10
9.	Normal Nucleoli	1 - 10
10.	Mitoses	1 - 10
11.	Class:	(2 for benign, 4 for malignant)

Class distribution:

Benign: 458 (65.5%)
Malignant: 241 (34.5%)
I randomly pick half for training and the remaining for testing.

a) In matlab, there is a neural network toolbox.

```
net=newff(minmax([train_input']),[10 10 1],{'tansig' 'tansig'
'tansig'},'traingd');
```

This routine 'newff' can create a feed-forward back propagation network. I use only one hidden layer. The input is a 1x10 vector and the output is a single value which should be between 0 and 1. At each neuron, after summing the input, the output is passed through a hyperbolic tangent sigmoid transfer function.

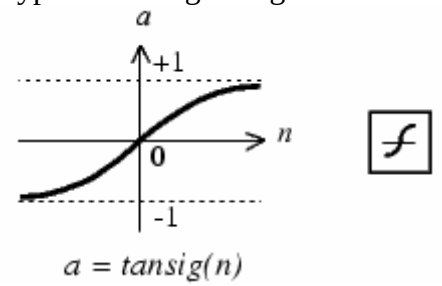


Figure 7: The hyperbolic tangent sigmoid transfer function.

The adjustment of each weight vector is done by using the gradient descent algorithm. In my design, I choose the learning rate η equal to 0.01.

$$w(k+1) = w(k) - \eta\left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_{last}}\right)$$

```
net.trainParam.show = 5;
net.trainParam.lr = 0.01;
net.trainParam.epochs = 100;
net.trainParam.goal = 1e-2;
[net,tr]=train(net,train_input',train_output');
```

In order not to overfit the classifier, the stopping criteria is that either it is trained with 100 epochs or the error between the true output and the predicted output is less than 0.01.

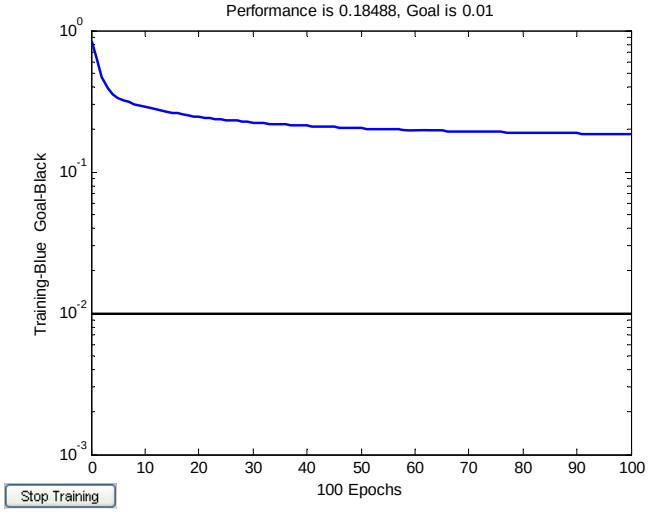


Figure 8: The training error becomes smaller when we keep training.

This is a classification problem. The output of the classifier is between 0 and 1. I set a threshold which is 0.5. If the output is greater than 0.5, I say this sample belongs to class1. If the output is less than 0.5, I say this sample belongs to class2.
I tried the experiment with 5 times. Due to the random initialization of the weight, the classification accuracy is different each time.

Table2: Each run, the weights are randomly initialized. The training accuracy and the testing accuracy are different for each run.

Run	1	2	3	4	5
Training accuracy	0.6923	0.6923	0.6503	0.6993	0.7133
Testing accuracy	0.6434	0.7343	0.6573	0.6573	0.6923

b) I use SVMlight which is downloaded from the website.
http://svmlight.joachims.org/
In the coding, we can define the parameters or choose the kernel function by ourselves.

$$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

subject to

$$y_i(x_i^T w + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

Here, I set C=1. 'Kernel' means what type of kernel function we choose.

```
net = svml('file_SVM00.txt','Kernel',0,'C',1);
net = svmltrain(net,train_input,train_output_);
pred_Ytr = svmlfwd(net,train_input);
```

Because the output of SVM is either 1 or -1, I need to convert the original training output {0,1} to {-1,1}. After the training, we can do the testing by using 'svmlfwd'. The output is the continuous value. I set the threshold equal to 0. When the output is less than 0, I label the output as 0. When the output is greater than 0, I label the output as 1.
The training accuracy is 70.63% and the testing accuracy is 69.93%.

c) Comparison between neural network and support vector machine
Neural Network
At each node, we sum the $x^T w$ of previous layer, and then apply by the sigmoid function and transmit this number to the next node. We use back propagation to update the weight. It learns from examples just like a human being learns new things.
Pros:

- The idea is heuristic. The machine is learning from the error and updating its weight.
- It is good at function approximation.

Cons:

- Each time, we might get different neural network due to the random initialization of the weights. This may affect the testing accuracy a lot.
- We need to pick the learning rate training cycles.

Support vector machine

For linear kernel function, we are trying to find the best linear hyperplane which separates 2 classes.

Pros:

- Given the training data, we can generate only one SVM. There is the optimization to get better separation hyperplane.
- After training, it only remembers the support vectors which are on the boundary plane 1 and -1.

Cons:

- When the problem is hard to solve, it may take longer training time.
- Selection of C and the parameters of the Kernel is a tricky issue.

3. Parzen Window and nearest neighbor

a) Parzen window

We assign class *j* to the sample $\vec{x_0}$, when

$$p(w_j \mid \vec{x_0}) \geq p(w_i \mid \vec{x_0})$$

$$\Rightarrow p(\vec{x_0} \mid w_j)\, p(w_j) \geq p(\vec{x_0} \mid w_i)\, p(w_i)$$

$$\Rightarrow p(\vec{x_0}, w_j) \geq p(\vec{x_0}, w_i)$$

$$\frac{1}{N_t V} \sum_{l=1}^{d_j} \phi\left(\frac{x_l{}^j - x_0}{h} \right) \geq \frac{1}{N_t V} \sum_{l=1}^{d_i} \phi\left(\frac{x_l{}^i - x_0}{h} \right)$$

where Nt = total number of samples,

V is the volume.

$$\Rightarrow \sum_{l=1}^{d_j} \phi\left(\frac{x_l{}^j - x_0}{h} \right) \geq \sum_{l=1}^{d_i} \phi\left(\frac{x_l{}^i - x_0}{h} \right)$$

In 2D, we have the normal Gaussian distribution. I assume $\sigma_X = \sigma_Y$. Mean =$[0\ 0]^T$

$$f(x, y) = \frac{1}{2\pi\sigma^2} \exp(-\frac{(x^2 + y^2)}{2\sigma^2})$$

There I choose the normal Gaussian as my window function.

$$\phi\left(\frac{x_l - x_0}{h} \right) = \frac{1}{2\pi h^2} \exp\left(-\frac{|\, x_l - x_0 \,|^2}{2h^2} \right)$$

Table 3: The classification accuracy of Parzen window varies with different *h*.

h	0.1	0.5	1	2
Training accuracy	0.7063	0.7063	0.7063	0.7063
Testing accuracy	0.6014	0.6643	0.6713	0.6993

b) K- nearest neighbor

In order to not have the tie vote, I only use the odd number for *k*. I assign the category to the testing sample based on majority of of *k* nearest neighbors.

Table 4: The classification accuracy of k nearest neighbor varies with different *k*.

k	3	5	7	9	11	13
Training accuracy	0.8462	0.7972	0.7483	0.7343	0.6853	0.6853
Testing accuracy	0.6224	0.6364	0.6014	0.6643	0.7133	0.7063

c) Nearest neighbor

When *k* is 1, it is nearest neighbor method.

Table 5: The classification accuracy of nearest neighbor.

Training accuracy	1
Testing accuracy	0.5944

d) Comparison of these 3 methods

Both Parzen window and k nearest neighbor require us to choose the parameter.

We can notice that the training accuracies of Parzen window classifier are the same. The highest testing accuracy for Parzen window is obtained when we enlarge *h* which means we make the volume larger.

The highest testing accuracy for *k* nearest neighbor is obtained when *k* equal to 11. The optimal *k* might be different for other datasets.

Nearest neighbor can get 100% training accuracy, but this is sort of over fitting. The testing accuracy is the worst among these 3 methods.

Appendix

```
%question 1
%Comparison between w for Fisher's Linear Discrimant and w for (m1-m2)

n=2;%I choose dimension=2
N=1000;% N points for class 1, and N points for class 2

mu1=[0 0 ];
cov1=[1 0.9 ;0.9 1];
mu2=[2 0 ];
cov2=[1 0.9 ;0.9 1];

[x1,x2]=gen_multi_gauss(mu1,cov1,mu2,cov2,N);

figure(1);
scatter(x1(:,1),x1(:,2),'or')
hold on;
scatter(x2(:,1),x2(:,2),'xb')
hold off;

m1=mean(x1);
m2=mean(x2);
S1=(x1-ones(N,1)*m1)'*(x1-ones(N,1)*m1);
S2=(x2-ones(N,1)*m2)'*(x2-ones(N,1)*m2);
w=pinv(S1+S2)*(m1-m2)';
w_fisher=w/norm(w);
x1_fisher=x1*w_fisher;
x2_fisher=x2*w_fisher;

x_axis=linspace(-6,4,200);

figure(2);
hist(x1_fisher,x_axis)
figure(3);
hist(x2_fisher,x_axis)

figure(4);
y1_fisher = ksdensity(x1_fisher,x_axis);
y2_fisher = ksdensity(x2_fisher,x_axis);
plot(x_axis,y1_fisher,'r.',x_axis,y2_fisher,'b-')

error_area=simpson_quad(x_axis,min([y1_fisher;y2_fisher]))

w=(m1-m2)';
w_meandiff=w/norm(w);
x1_meandiff=x1*w_meandiff;
x2_meandiff=x2*w_meandiff;

x_axis=linspace(-6,4,200);

figure(5);
hist(x1_meandiff,x_axis)
figure(6);
```

```
hist(x2_meandiff,x_axis)

figure(7);
y1_meandiff = ksdensity(x1_meandiff,x_axis);
y2_meandiff = ksdensity(x2_meandiff,x_axis);
plot(x_axis,y1_meandiff,'r.',x_axis,y2_meandiff,'b-')

error_area=simpson_quad(x_axis,min([y1_meandiff;y2_meandiff]))

function [ output_integration ] = simpson_quad( x,f )
%simpson_quad for calculating the area under the curve
%Assume h is equal between any 2 points
integral=f(1)+f(end);

for i=2:size(f,2)-1
    if mod(i,2)==0
        integral=integral+4*f(i);
    else
        integral=integral+2*f(i);
    end
end

output_integration=integral*(x(2)-x(1))/3;


%question 2
load('dataset.mat');

train_input=traindata(:,2:end);
test_input=testdata(:,2:end);
train_output=traindata(:,1);
test_output=testdata(:,1);

%Use Matlab Neural Network Toolbox
net=newff(minmax([train_input']),[10 10 1],{'tansig' 'tansig'
'tansig'},'traingd');
net.trainParam.show = 5;
net.trainParam.lr = 0.01;
net.trainParam.epochs = 100;
net.trainParam.goal = 1e-2;
[net,tr]=train(net,train_input',train_output');

%Evaluate the training error
out_tr= sim(net,train_input');
pred_train=(sign(out_tr'-0.5)+1)/2;
tr_err=sum(abs(pred_train-train_output))/size(train_output,1);
tr_acc=1-tr_err;

%Evaluate the testing error
out_te= sim(net,test_input');
pred_test=(sign(out_te'-0.5)+1)/2;
te_err=sum(abs(pred_test-test_output))/size(test_output,1);
te_acc=1-te_err;
result=[tr_acc te_acc]'
```

```
%(b)Use SVM light
net = svml(sprintf('file_SVM00.txt'),'Kernel',0,'C',1);
%net  =  svml(sprintf('file_SVM00.txt'),'Kernel',2,'KernelParam',1,'C',1);

train_output_=sign(train_output-0.5);
net = svmltrain(net,train_input,train_output_);
pred_Ytr = svmlfwd(net,train_input);

greater_index = pred_Ytr>0;
pred_Ytr(greater_index)=1;
pred_Ytr(~greater_index)=-1;
%Calculate correction rate
err_rate=sum(abs(train_output_-pred_Ytr)/2)/size(pred_Ytr,1);
correct_rate_training=1-err_rate

test_output_=sign(test_output-0.5);
pred_Yte = svmlfwd(net,test_input);

greater_index = pred_Yte>0;
pred_Yte(greater_index)=1;
pred_Yte(~greater_index)=-1;
%Calculate correction rate
err_rate=sum(abs(test_output_-pred_Yte)/2)/size(pred_Yte,1);
correct_rate_testing=1-err_rate

%question 3
%Use Parzen Window, K-nearest neighbor, nearest neighbor

load('dataset.mat');

train_input=traindata(:,2:end);
test_input=testdata(:,2:end);
train_output=traindata(:,1);
test_output=testdata(:,1);

[COEFF, latent] = pcacov(train_input);
train_input2D=train_input*COEFF;
train_input2D=train_input2D(:,1:2);
test_input2D=test_input*COEFF;
test_input2D=test_input2D(:,1:2);

x1=train_input2D(train_output==0,:);
x2=train_input2D(train_output==1,:);

figure(1);
scatter(x1(:,1),x1(:,2),'or')
hold on;
scatter(x2(:,1),x2(:,2),'xb')
hold off;

%(a)Parzen Window method
pred_train=zeros(size(train_output,1),1);
h=2;
```

```
for i=1:size(train_input2D,1)
    P1=0;
    for j=1:size(x1,1)
        P1=P1+window_fx(x1(j,:),train_input2D(i,:),h);
    end
    P2=0;
    for j=1:size(x2,1)
        P2=P2+window_fx(x2(j,:),train_input2D(i,:),h);
    end
    if P2>P1
        pred_test(i)=1;
    end
end
tr_err=sum(abs(pred_train-train_output))/size(train_output,1);
tr_acc=1-tr_err

pred_test=zeros(size(test_output,1),1);
for i=1:size(test_input2D,1)
    P1=0;
    for j=1:size(x1,1)
        P1=P1+window_fx(x1(j,:),test_input2D(i,:),h);
    end
    P2=0;
    for j=1:size(x2,1)
        P2=P2+window_fx(x2(j,:),test_input2D(i,:),h);
    end
    if P2>P1
        pred_test(i)=1;
    end
end
te_err=sum(abs(pred_test-test_output))/size(test_output,1);
te_acc=1-te_err

%(b)K nearest nighbor, when k=1 => nearest neighbor
pred_train=zeros(size(train_output,1),1);
k=13;
for i=1:size(train_input2D,1)
    dist_to_train=zeros(size(train_input2D,1),1);
    for j=1:size(train_input2D,1)
        dist_to_train(j)=norm(train_input2D(i,:)-train_input2D(j,:));
    end
    [B,idx] = sort(dist_to_train,'ascend');
    pred=0;
    for j=1:k
        pred=pred+train_output(idx(j));
    end
    if pred>k*0.5
        pred_train(i)=1;
    end
end
tr_err=sum(abs(pred_train-train_output))/size(train_output,1);
tr_acc=1-tr_err

pred_test=zeros(size(test_output,1),1);
for i=1:size(test_input2D,1)
    dist_to_train=zeros(size(train_input2D,1),1);
```

```
    for j=1:size(train_input2D,1)
        dist_to_train(j)=norm(test_input2D(i,:)-train_input2D(j,:));
    end
    [B,idx] = sort(dist_to_train,'ascend');
    pred=0;
    for j=1:k
        pred=pred+train_output(idx(j));
    end
    if pred>k*0.5
        pred_test(i)=1;
    end
end
te_err=sum(abs(pred_test-test_output))/size(test_output,1);
te_acc=1-te_err

function output = window_fx( x,y,h)
%Assume window function is 2D normal Gaussian distribution
output=(1/(2*pi*h^2))*exp(-(norm(x-y)^2)/(2*(h^2)));
```