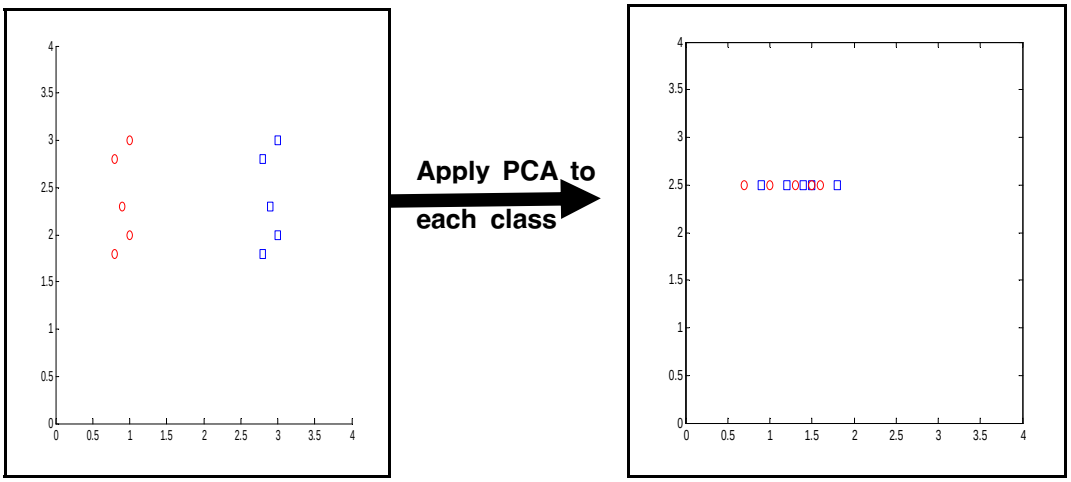
Question 1.

In the Parametric Method section of the course, we learned how to draw a separation hyperplane between two classes by obtaining w0, the argmax of the cost function $J(w) = w^T S_B w \,/\, w^T S_w w$. The solution was found to be $w_0 = S_w^{-1}(m_1 - m_2)$, where m_1 and m_2 are the sample means of each class, respectively. Some students raised the question: can one simply use $J(w) = w^T S_B w$ instead (i.e. setting S_w as the identity matrix in the solution w_0? Investigate this question by numerical experimentation.

Solution 1

The most famous example of dimensionality reduction is PCA(Principle Components Analysis). This technique searches for directions in the data that have largest variance and subsequently project the data onto it. However, the directions of maximum variance may be useless for classification. PCA is an unsupervised technique and as such does not include label information of the data. For example, if we image cigar like clusters in 2-dimensions, one cigar has x = 1 and the other x = 2. The cigars are positioned in parallel and very closely together like the below pictures.



For classification, this would be a terrible projection, because all labels get evenly mixed and we destroy useful information. A much more useful projection is orthogonal to the cigars, i.e. in the direction of least overall variance, which would perfectly separate the data cases. So the question is, how do we utilize the label information in finding informative projection? To that purpose Fisher linear discriminant analysis considers maximizing the following objective

$$J(w) = w^T S_B w \,/\, w^T S_w w$$

We can get the solution, $w_0 = Sw^{-1}(m_1 - m_2)$, by the generalized eigenvalue problem.

The m_1 and m_2 represent the sample means of each class. Some students raised the question: can one simply use $J(w) = w^T S_B w$ instead (i.e. setting S_w as the identity matrix in the solution w_0?. It is obvious the given solution, $w_0 = m1\text{-}m2$, depends on only the sample means of each class and it doesn't consider the variances of the classes. In other words, the scatter factors, the spread of data around the mean, for each class aren't considered. The following case is the counterexample of the given solution.

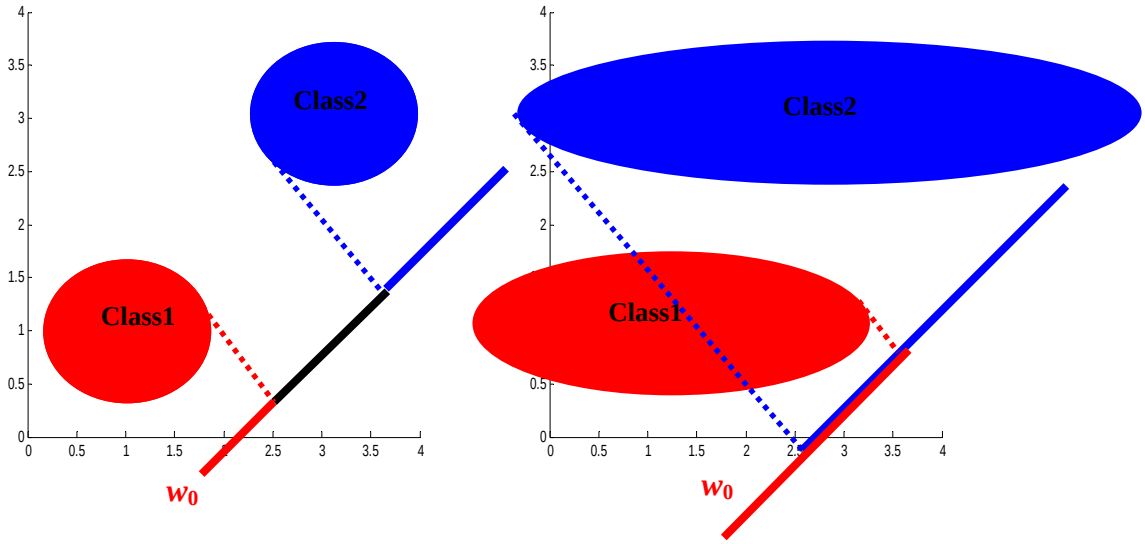


Fig.1-1. The counterexample of the given solution.

In the above pictures, both are the same means of each class but different variances, so they have the same hyperplane by the given solution. In result, the left case is well classified, but the right case is mixed up. Therefore, the given solution isn't guaranteed to find a projection vector used to classify.

Here were present another counterexample using real training samples by matlab simulation.

Example)

*** Training samples**

- **Class 1 = { (1,2), (2,3), (3,3), (4,5), (5,5) }**
- **Class 2 = { (1,0), (2,1), (3,1), (3,2), (5,3), (6,5) }**

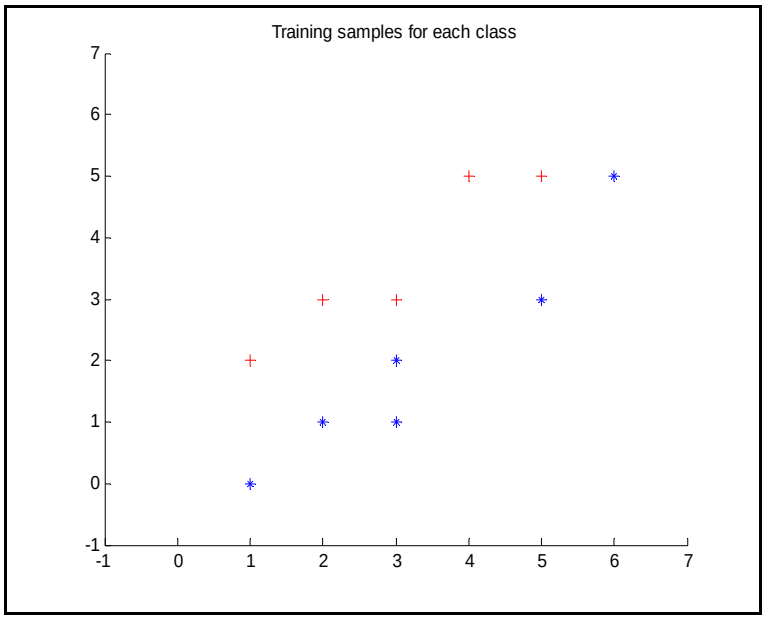


Fig.1-2. Training samples for class1 and class2

- ***S1*** **= cov(Class1) = [2.5 2.0 ; 2.0 1.8]**
- ***S2*** **= cov(Class2) = [3.5 3.2 ; 3.2 3.2]**
- ***Sw*** **=** ***S1*** **+** ***S2*** **= [6.0 5.2 ; 5.2 5.0]**

- For case 1 : $w_0 = Sw^{-1}(m_1 - m_2)$ **= [-3.5752 4.0382]**

- For case 2 : $w_0 = (m_1 - m_2)$ **= [-0.3333 1.6000]**

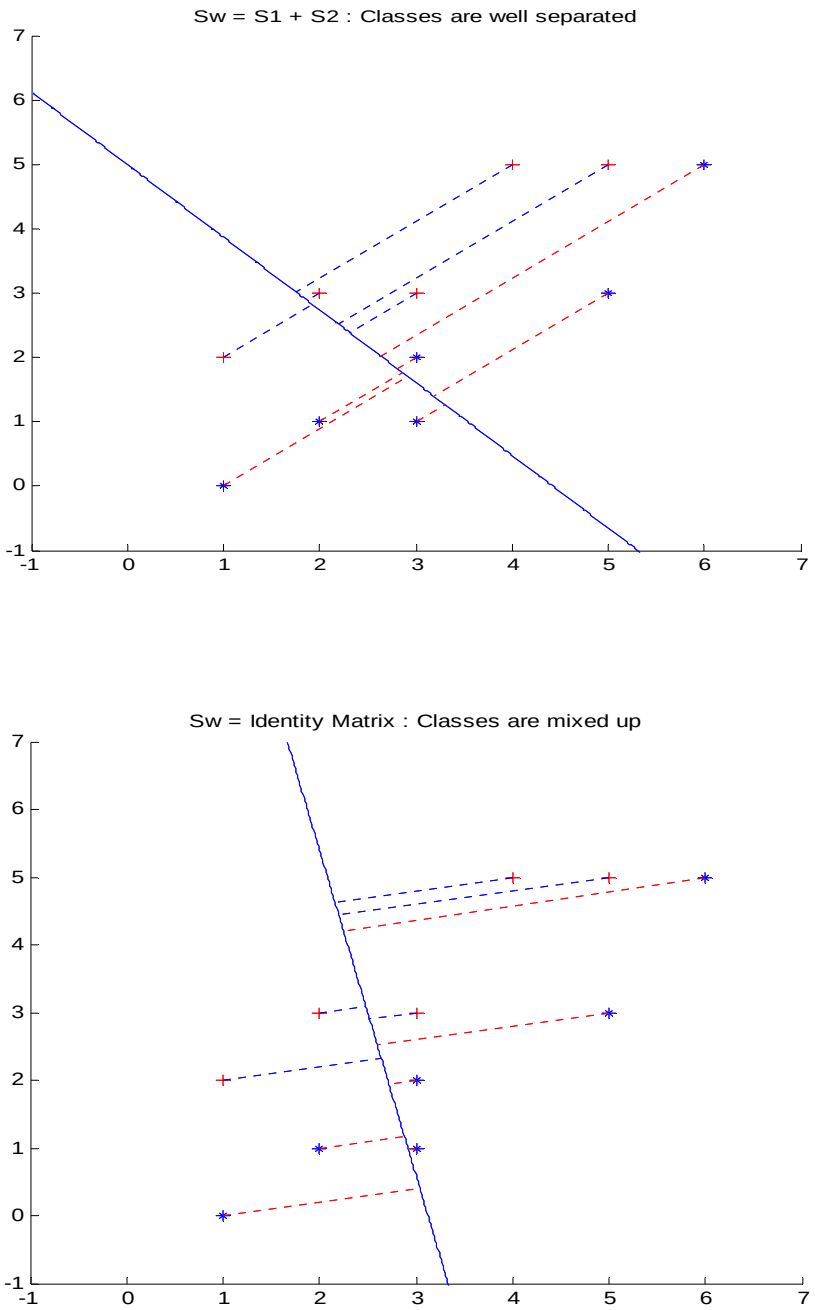


Fig.1-3. Classification example using the Fisher linear discriminant(FLD)

As you can see the result of two cases, classes are well separated in case 1 but, classes are mixed up in case2. In conclusion, if we get a projection vector using $w_0 = Sw^{-1}(m_1 - m_2)$, we are guaranteed that the classes are well separated, but $w_0 = (m_1 - m_2)$ isn't guaranteed to find a projection.

Question 2.

Obtain a set of training data. Divide the training data into two sets. Use the first set as training data and the second set as test data.

a) Experiment with designing a classifier using the neural network approach.

b) Experiment with designing a classifier using the support vector machine approach.

c) Compare the two approaches.

Solution 2:

a)

The NN(Neural Network) approach is a pattern classification algorithm which falls into the broad class of nearest neighbor like algorithm. It is called a neural network because it is based on the concept of neurons in the brain. The figure below displays the architecture for a NN that recognizes 2 classes.

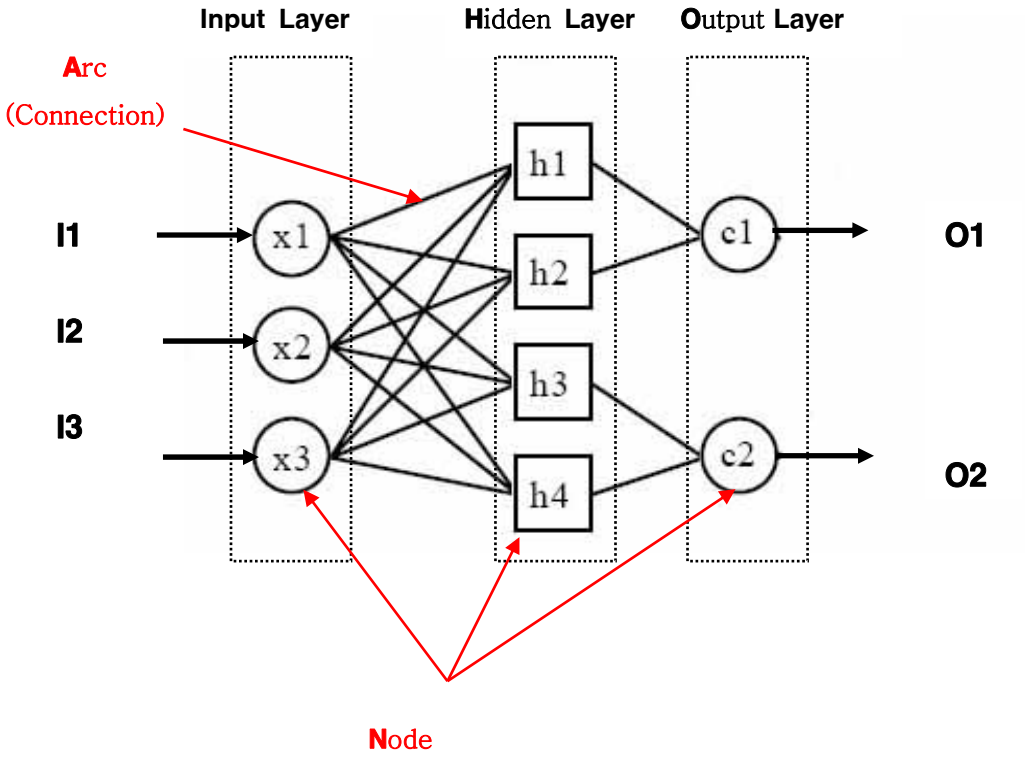


Fig.2-1. The components of neural network classification

The NN has 3 layers of nodes.

Input Layer

- **Input layer has the attributes participating in the classification.**
- **Attribute selection involves the examination of data and the domain expert knowledge.**
- **The inputs are the attribute values for each data tuple.**
- **The number of nodes in input layer is typically defined based on different attribute types and attribute domain.**

Hidden Layer

- **Hidden layer is constructed for the process of learning by computations on their nodes and arcs weights.**
- **A network can have one or more hidden layers.**
- **The number of nodes is determined by experimentation.**
- **Too many nodes lead to over-fitting and too little nodes reduces the classification accuracy.**

Output Layer

- **The result of the classification is the output of a node in output layer.**
- **Weights and activation functions determine the output.**
- **Typically the is one output node for each class.**

The connections (arcs) are from input nodes to hidden layer nodes and from hidden layer nodes to output nodes. If one topology doesn't give good results after some iterations of learning, then change the topology. Each arc is assigned an initial random weight used in training and modified in the learning process.

Training process

- **Run a sample form training set,**
- **The summation of weights and activation functions are applied at each node of hidden and output layers, until output is generated**
  ➔ ***Feed-Forward Process***
- **Compare output with the expected output from the training set.**
- **If output doesn't match, modify arc weights and biases of nodes**
  ➔ ***Back-Propagation Process***
- **Run the next sample using the same process**

Advantage of neural network technique

- Tolerance to noise data

Drawback of neural network technique

- Long learning process

- Getting the number of hidden layers using only experiment

Experimental result : Error rate using the neural network approach

- **Training samples for each class = 100**
- **Test samples = 200 (class1 = 100, class2 = 100)**

# of hidden layers / # of iterations for back-propagation	10	30	50
10	50%	6.5%	7.0%
50	50%	6.5%	6.0%
100	50%	6.5%	6.0%

☞ **The error rate using the Bayesian decision rule : 6.0%**

Compared with Bayesian decision, it produces similar error rate, but it takes long learning process while simulation and it has a drawback to get the best value for the number of hidden layers by experiment. In this case, '50' is the best value for the number of hidden layer.

Simulation results using neural network and Bayesian decision

- **Training samples for each class = 100**
- **Test samples = 200 (class1 = 100, class2 = 100)**

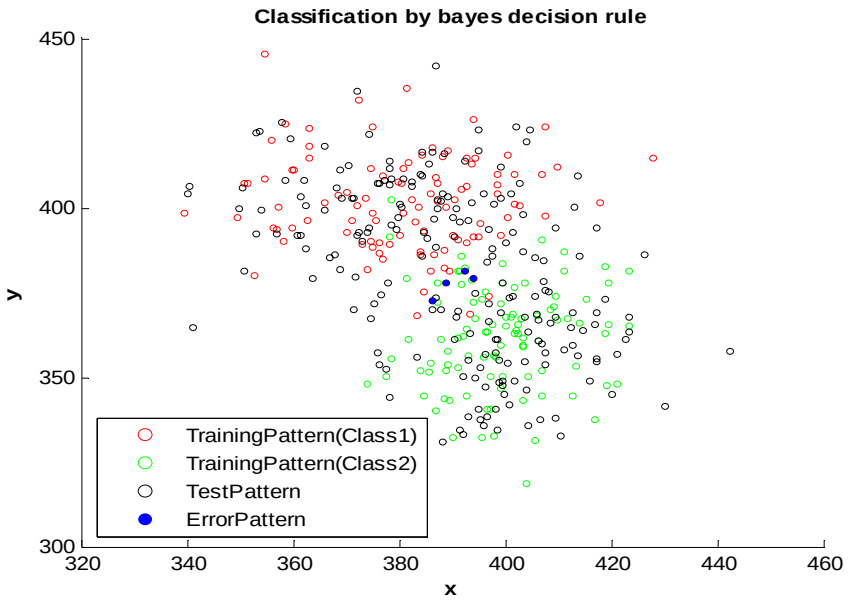


Fig.2-2. The classification using the Bayesian decision(Error rate = 6.0%)

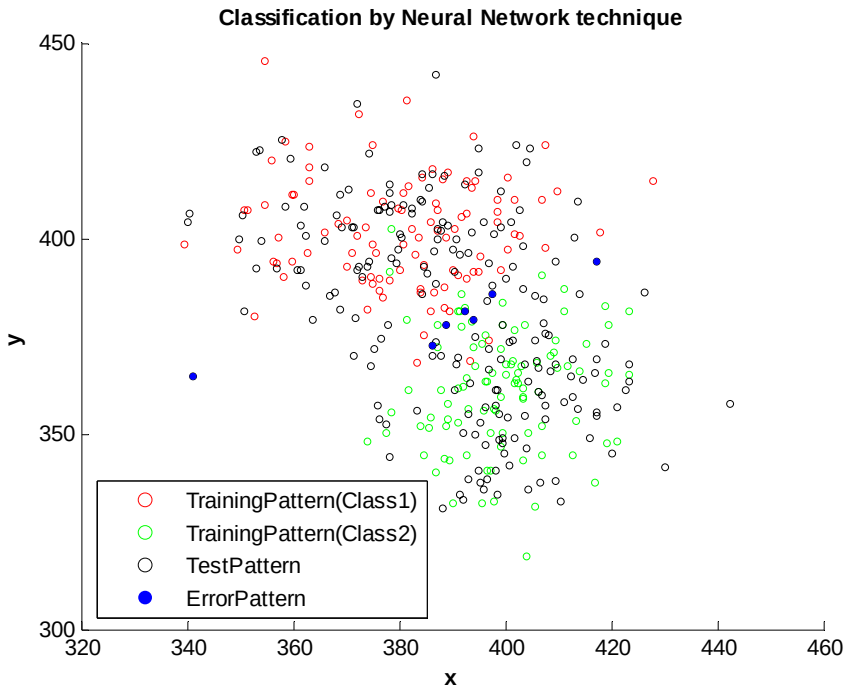


Fig.2-3. The classification using the neural network (Error rate = 6.5%)

b)

SVM (Support Vector Machine) is a kind of supervised learning method used for classification and regression.

For classification, SVM operate by finding a hypersurface in the space of possible inputs. This hypersurface will attempt to split one class samples from another class samples. The split will be chosen to have the largest distance from the hypersurface to the nearest of each class samples. In other words, the objective of SVM is that the hyperplane leaves the maximum margin between the two classes. hence they are also known as maximum margin classifiers. The parameters in maximum margin hypesurface can be calculated by a QP(quadratic program) optimization problem.

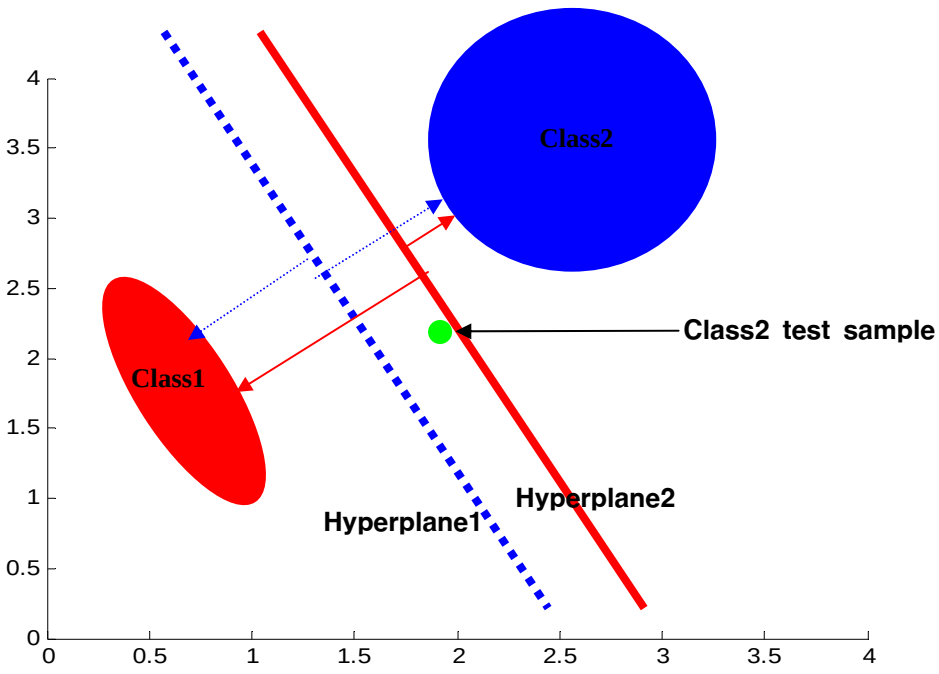


Fig.2-4. The hyperplane with maximum margein

In the above pictures, the red and blue hyperplanes classified well samples into two classes. Compared with the blue dotted hyperplane, the red dashed hyperplance is very closer to the class. If we place a calass2 test sample(green circle) to the above position, the red hyperplane fail in classification. So, it is very important that hyperplane have maximized margin.

Advantages

- **The overfitting problem can be easily controlled**
- **Excellent generalization properties**
- **Objective function has no local minima**
- **Can be used to find non-linear discriminant functions**
- **Complexity of the classifier is characterized by the number of support vectors**

Drawbacks

- **It's not clear how to select a kernel function in a principled manner**
- **Tends to be slower than other methods**

Experimental result

- **Training samples for each class = 100**
- **Test samples = 200(class1 = 100, class2 =100)**
- **Kernel function : Gaussian**

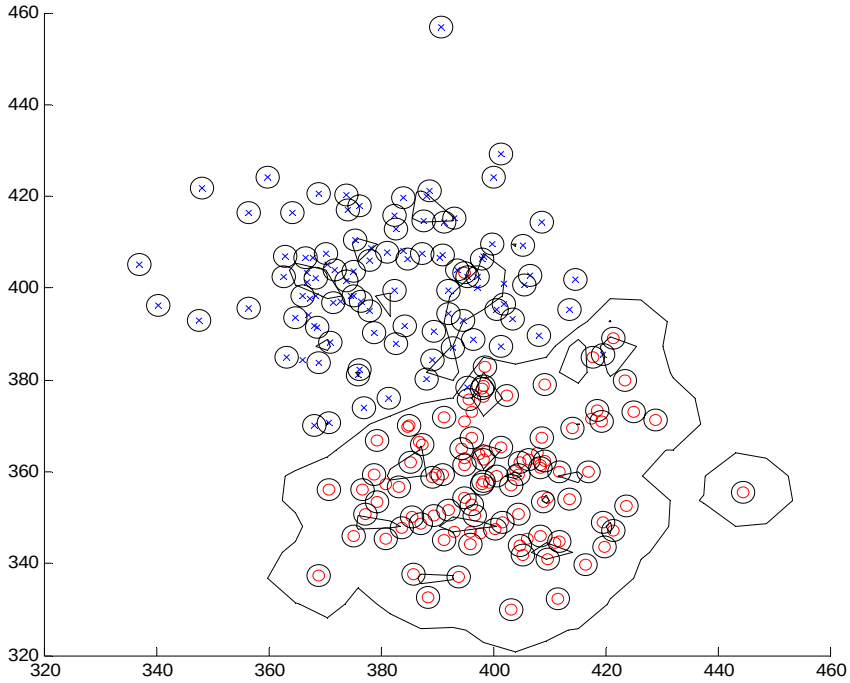


Fig.2-5. non-linear decision boundaries by SVM

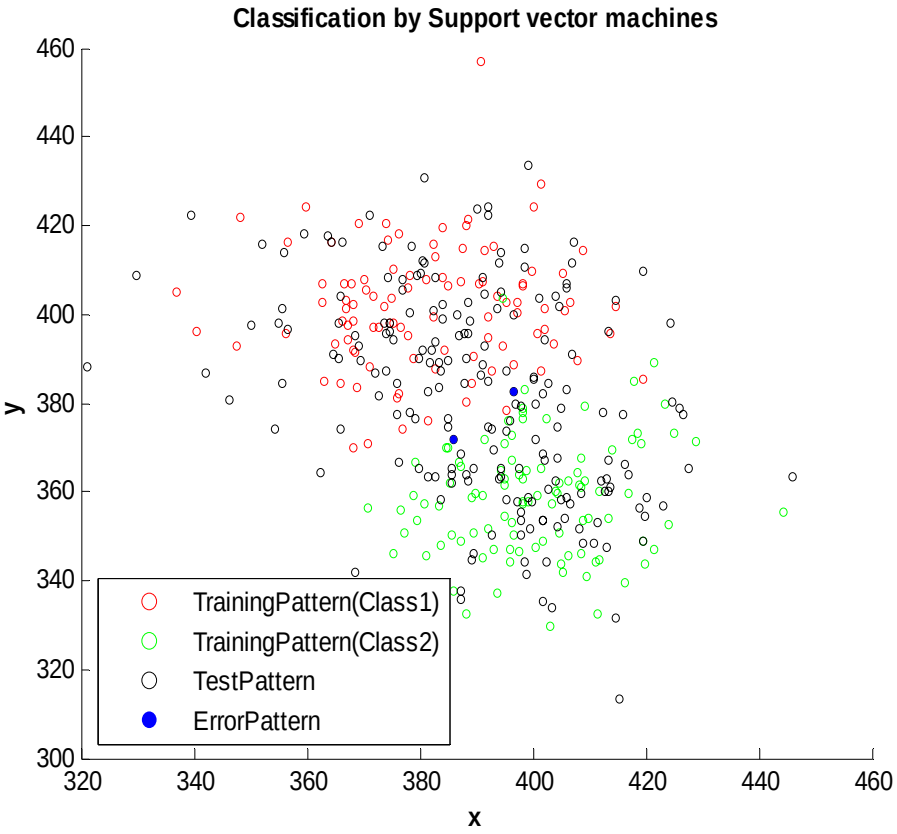


Fig.2-6. The classification result by SVM(Error rate = 5.5%)

In the experiment, I chose the Gaussian function as a kernel function because it performs well under general smoothness assumption.

C) Comparison the SVM(Support Vector Machine) and NN(Neural Network)

The SVM and the NN are trained in terms of deviation performance, the former to find the error bound and the latter to minimize MSE. The SVM has the following advantages over the NN

i) It can obtain the global optimization

ii) The overfitting problem can be easily controlled

So, lots of empirical tests have showed that the performance of SVM is better than NN in classification.

In this test, I have examined the results of classification by experiments with designing classifiers using neural network and support vector machine approaches.

In my experiments, the SVM and the NN approaches perform well in classification even though the NN shows bad results in accordance with the number of hidden layers and the initial value of weights.

Although previous research claims that the SVM method is superior to the NN, but the performance of SVM in not always better than the NN in my experiments (refer to the below table2-1)

Table 2-1. Comparison of SVM and NN on the test using error rate(%)

# of trial / Classifier	1	2	3	4	5
Bayesian	**6.5**	**6.0**	**6.5**	**6.0**	**6.0**
NN	**5.5**	**6.5**	**5.5**	**6.0**	**6.5**
SVM	**5.5**	**6.0**	**5.5**	**6.0**	**6.0**

☞ I used SVM and NN function codes download from the statistical pattern recognition Matlab tool box
(http://cmp.felk.cvut.cz/cmp/software/stprtool/index.html)

Question 3.

Using the same data as for question 2 (perhaps projected to one or two dimensions for better visualization),

a) Design a classifier using the Parzen window technique.

b) Design a classifier using the K-nearest neighbor technique

c) Design a classifier using the nearest neighbor technique.

d) Compare the three approaches.

Solution 3:

It is the general expression of non-parametric density estimation $p(x) = \frac{k/n}{v}$**, we have made two approximations, the Parzen window and the k-nearest neighbor techniques.**

where, ***x*** **is inside some region** ***R***

k **is the number of samples inside** ***R***

n **is the total number of samples**

V **is the volume of** ***R***

a) Parzen window approach

Choose a <u>fixed value for volume *v*</u> and <u>determine the corresponding *k*</u> from the data.

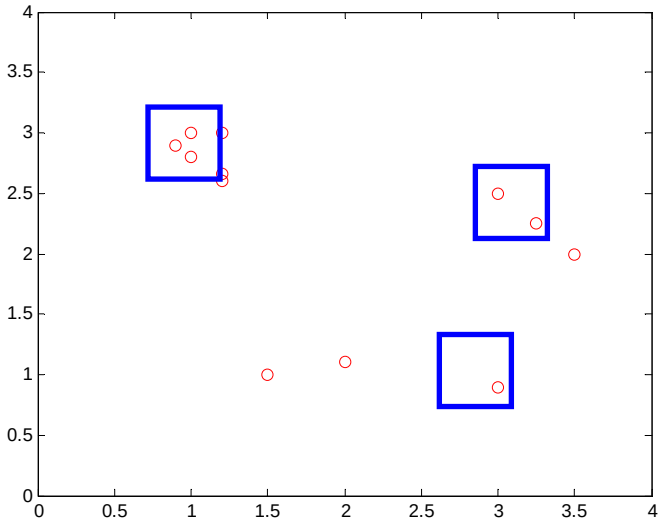


Fig.3-1. The example of the Parzen window technique

How to choose *n* and *V* ? The number of samples is always fixed in experiment, thus the only available parameter is *V*. If *V* is too small, *p(x)* = 0 for most *x*, because most regions will have no samples. Thus have to find compromise for *V*, it is not too small so that it has enough samples but also not too large so the *p(x)* is approximately constant inside *V*. so, the problem of choosing the *V* is crucial in the Parzen window approach.

Experimental results

- **Training samples for each class = 100**
- **Test samples = 200(class1 = 100, class2 =100)**
- **Kernel function : Gaussian**

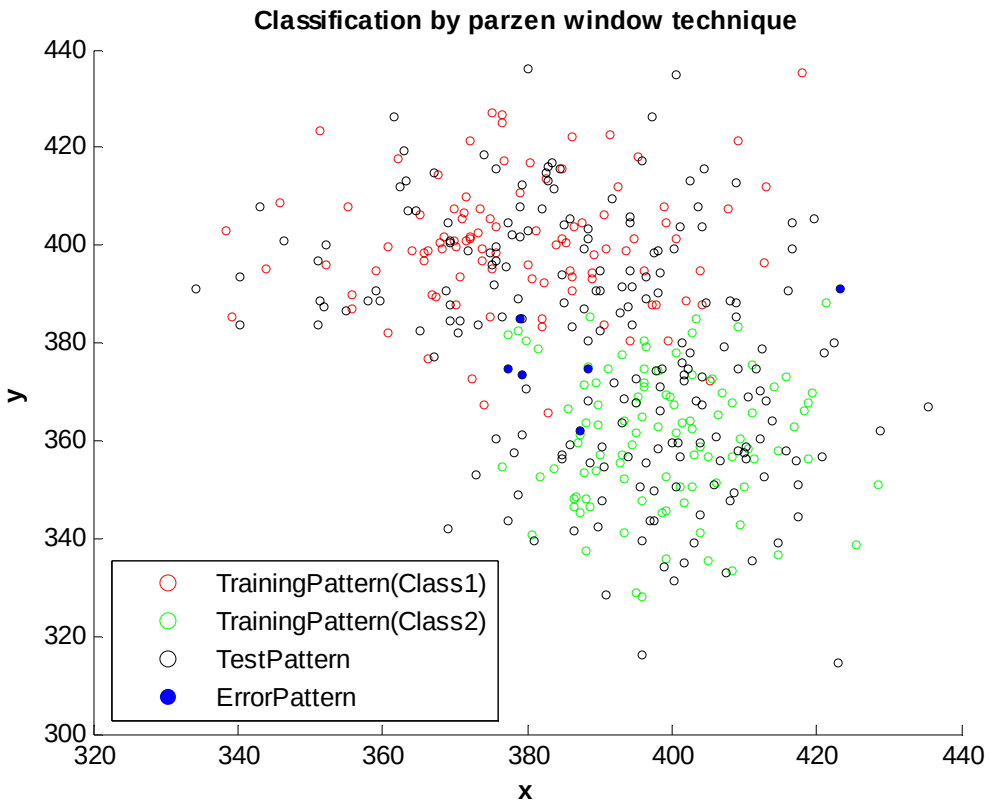


Fig.3-2. The classification using the Parzen window (Error rate = 8%)

Table.3-1. The error rate for each window size(*V*)

Window size(*V*)	0.1	10	100	500	1000	>1000
Error rate(%)	28.5	15.5	8.0	7.5	7.5	50

As you can see the above the table 3-1, it is crucial to choose the best window size as previously mentioned.

2) K-nearest approach

Choose a fixed value for *k* and determine the corresponding volume *V* from the data.

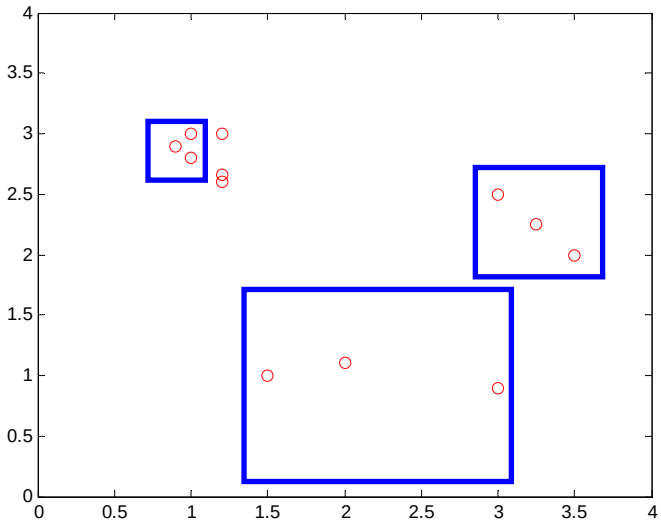


Fig.3-3. The example of the K-nearest neighbor(k = 3)

For that case, it is crucial to find *k*. If *k* is large so that error rate is minimized, then it will lead to over-smoothed boundaries, but *k* is small so that only near by samples are included, then will lead to noisy decision boundaries.

Experimental results

- **Training samples for each class = 100**
- **Test samples = 200(class1 = 100, class2 =100)**
- **K = 5**

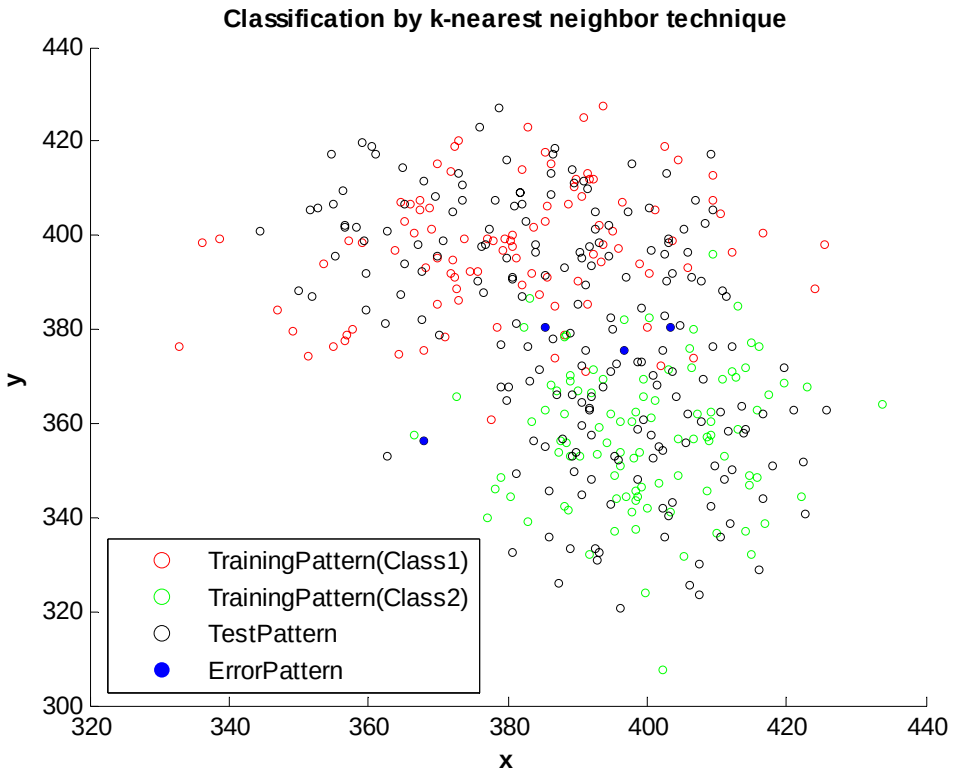


Fig.3-4. The classification using K-nearest neighbor (Error rate = 8%)

Table.3-2. The error rate for each *k* value
(# of training = 1000, # of test = 2000)

k	1	5	10	50	100	500
Error rate(%)	8.65	6.35	5.75	5.70	5.45	6.0

As you can see the above the table 3-2, it is crucial to choose the best *k* as previously mentioned.

c)

In general, the decision boundary is the boundary in input space that our decision as to the class of the input changes aw e cross this boundary. In the nearest neighbor algorithm, the decision boundary is determined by the lines which are the perpendicular bisectors of the closet training points with different training labels. See Fig.3-5. This is called a Voronoi diagram.

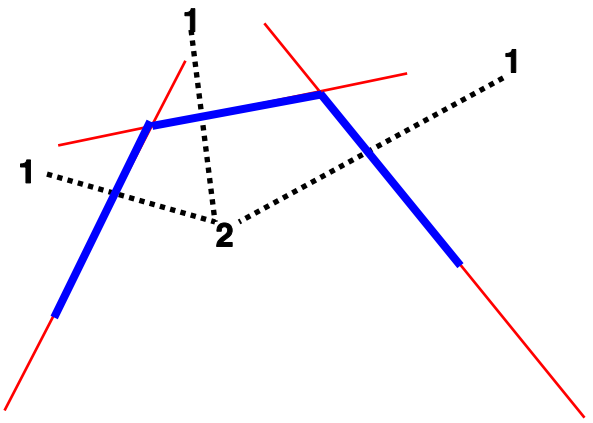


Fig.3-5. The decision boundary for the nearest neighbor classification

(The blue lines represent the decision boundary)

Experimental result

- **Training samples for each class = 100**
- **Test samples = 200(class1 = 100, class2 =100)**

Fig.3-6. The training samples

Fig.3-7. The decision boundary (Voronoi diagram)

☞ I used a Vonoroi function code download from the website.
(http://stuff.mit.edu/afs/sipb.mit.edu/user/arolfe/mat)

d)

Table 3-3. The error rate(%) for each technique

# of trial / Classifier	1	2	3	4	5
Parzen window	**7.5**	**7.5**	**7.5**	**6.5**	**7.5**
K-nearest	**7.5**	**9.5**	**4.5**	**7.5**	**7.5**
Nearest neighbor	**9.0**	**8.5**	**9.0**	**7.5**	**8.0**

The Parzen, KNN, and NN perform well in classification. In the experiment, the Parzen is a little better than the KNN and the NN, but it's not always true. The performance of all methods depends on the number of samples. so, we need a lot of samples for accurate density estimation.

The KNN gives good classification if the number of samples is large enough and has a few advantages, it can be applied to the data from any distribution and it is very simple and intuitive. It is difficult to choose best k in this method. The *k* in the KNN plays a similar role as the window size in the Parzen window.

The Parzen window also gives good classification if the number of samples is large, it is also difficult to choose the appropriate window size.

The parametric techniques forms rarely fit the densities encountered in practive, but the non-parametric techniques can be used with arbitrary distributions and without an assumption about forms of the underlying densities are known.

Matlabe Code

```
%----------------------------------------
%  Question1 : Fisher Linear Discriminant
%  Designed by JT GONG
%----------------------------------------

clear all;
Range = 1000;

Class1 = [1 2; 2 3; 3 3; 4 5; 5 5];
Class2 = [1 0; 2 1; 3 1; 3 2; 5 3; 6 5];

[x1 y1] = size(Class1);
[x2 y2] = size(Class2);

u1      = sum(Class1)./x1;
u2      = sum(Class2)./x2;

%-----------------------------------
%  Sw = S1 + S2
%-----------------------------------
s1      =  cov(Class1);   %scatter matrix for class1
s2      =  cov(Class2);   %scatter matrix for class2

figure(1)
hold on
axis([-1 7 -1 7])
plot(Class1(:,1),Class1(:,2),'r+', Class2(:,1),Class2(:,2),'b*')
title('Training samples for each class')
hold off

sw      = s1 + s2;        %Within the class scatter

w1       = inv(sw)*(u1-u2)';
```

```
for i = 1:Range
    x(i) = -2 + (i-1)*0.01;
end
y       = w1(2)/w1(1) * x + 5 ;

for i = 1 : x1
    for j = 1:Range
    dist(j,i) = sum( (Class1(i,:) - [x(j) y(j)] ).^2 );
    end
end
[Class1value Class1index] = min(dist);

for i = 1 : x2
    for j = 1:Range
    dist(j,i) = sum( (Class2(i,:) - [x(j) y(j)] ).^2 );
    end
end
[Class2value Class2index] = min(dist);

figure(2)
hold on
axis([-1 7 -1 7])
plot(Class1(:,1),Class1(:,2),'r+', Class2(:,1),Class2(:,2),'b*')
plot(x,y)

for i = 1 : x1
    A = zeros(2,2);
    A(1,:) = Class1(i,:);
    A(2,:) = [x(Class1index(i)) y(Class1index(i))];

    plot(A(:,1),A(:,2),'b:')
end

for i = 1 : x2
    A = zeros(2,2);
    A(1,:) = Class2(i,:);
```

```
    A(2,:) = [x(Class2index(i)) y(Class2index(i))];

    plot(A(:,1),A(:,2),'r:')
end

title('Sw = S1 + S2 : Classes are well separated')

hold off

%-----------------------------------
%   Sw = Indentity Matrix
%-----------------------------------

w2       = (u1-u2); %sw is a identity matrix

y       = w2(2)/w2(1) * x + 15 ;

for i = 1 : x1
    for j = 1:Range
    dist(j,i) = sum( (Class1(i,:) - [x(j) y(j)] ).^2 );
    end
end
[Class1value Class1index] = min(dist);

for i = 1 : x2
    for j = 1:Range
    dist(j,i) = sum( (Class2(i,:) - [x(j) y(j)] ).^2 );
    end
end
[Class2value Class2index] = min(dist);


figure(3)
hold on
axis([-1 7 -1 7])
plot(Class1(:,1),Class1(:,2),'r+', Class2(:,1),Class2(:,2),'b*')
```

```
plot(x,y)
for i = 1 : x1
    A = zeros(2,2);
    A(1,:) = Class1(i,:);
    A(2,:) = [x(Class1index(i)) y(Class1index(i))];

    plot(A(:,1),A(:,2),'b:')
end

for i = 1 : x2
    A = zeros(2,2);
    A(1,:) = Class2(i,:);
    A(2,:) = [x(Class2index(i)) y(Class2index(i))];

    plot(A(:,1),A(:,2),'r:')
end
title('Sw = Identity Matrix : Classes are mixed up')
hold off



%-------------------------------------------------
%   HW2. Q2
%   a) Classifier uisng the Neural Network
%   b) Classifier using the Suppor Vector Machine
%-------------------------------------------------
%   JT Gong (03/29/2008)
%-------------------------------------------------
%-------------------------------------------------
%   HW2. Q2
%   a) Classifier uisng the Neural Network
%   b) Classifier using the Suppor Vector Machine
%-------------------------------------------------
%   JT Gong (03/29/2008)
%-------------------------------------------------
close all;
clear all;
```

```
%-- Initial Value Setup --%
Dim2 = 2;
u2_1 = [ 380   400 ];
u2_2 = [ 400   360 ];
covar2_1   = [ 300 20 ; 20 200 ];
covar2_2   = [ 150 15 ; 15 250 ];
outputlayer = 2; % the number of output layer

%-- Parameter Set up ----%
N = input('Enter the number of total generating data(training & test) =>   ');
NT= input('Enter the number of training data => ');
%The number of test data = N(total generation data) - NT(training data)

%- Generating training and test patterns using random gaussian generation
model.Mu = [u2_1' u2_2'];
model.C = zeros(Dim2,Dim2,2);
model.C(:,:,1) = covar2_1;
model.C(:,:,2) = covar2_2;
model.P = [0.5, 0.5];

XClass1 = mvnrnd(u2_1,covar2_1,N);
XClass2 = mvnrnd(u2_2,covar2_2,N);

TrainPattern     = [ XClass1(1:NT,:)' XClass2(1:NT,:)' ];
TrainLabel       = [ 1*ones(1,NT), 2*ones(1,NT) ];
TestPattern      = [ XClass1(NT+1:N,:)' XClass2(NT+1:N,:)'];
TrueClass        = [ 1*ones(1,N-NT), 2*ones(1,N-NT) ];


%-------------------------------------------------------------------
%   Classifier using the Bayesian Decision Rule
%-------------------------------------------------------------------

%-- Bayesian decision rule : function call --%
PredictedClass_baydec = baydec( TestPattern, model );
```

```
%-- The accuracy of the results ---%
Error_Baydec = accuracy(PredictedClass_baydec, TrueClass)

% Trianing pattern setup
trn.X = TrainPattern;
trn.nanme = 'train';
trn.y = TrainLabel;
trn.dim = Dim2;
trn.num_data = NT;

%-----------------------------------------------------------------
%    Classifier using the Neural network technique
%-----------------------------------------------------------------

hiddenlayer = input('Enter the number of hidden layer => '); % choose # of hidden layer
num_iter    = input('Enter the number of iteration ==> ');   % choose # of iteration for back-
propagation
nn_layer = [hiddenlayer, outputlayer];
net = newff(minmax(TrainPattern), nn_layer, {'logsig', 'logsig'}, 'trainbfg');
mu = [u2_1;u2_2];
sigma = [covar2_1;covar2_2];

net.performFcn = 'mse';
net.trainParam.epochs = 5;
net.trainParam.show = NaN;
net = init(net);
net = nnt_classify(net,trn,mu,sigma,num_iter);
out = sim(net,TestPattern);
[maxVal,PredictedClass_nn] = max(out);
Error_NeuralNetwork = accuracy(PredictedClass_nn,TrueClass)

%-----------------------------------------------------------------
%% classify using the Support vector machine
%-----------------------------------------------------------------
```

```
Kernelarg = input('Enter Kernel Argument ==> ');
Constant = input('Enter Regulation Constant ==> ');
options.ker = 'rbf';                % use Gaussian Kernel
options.arg = Kernelarg;            % kernel argument
options.C = Constant;               % regularization constant
model = smo(trn,options);           % SMO Sequential Minimal Optimization for binary SVM with
L1-soft margin.
ppatterns(trn);                     % PPATTERNS Plots pattern as points in feature space.
psvm(model);                        %PSVM Plots decision boundary of binary SVM classifier.
PredictedClass_SVM = svmclass(TestPattern,model); % SVMCLASS Support Vector Machines
Classifier.
Error_SVM = accuracy(PredictedClass_SVM, TrueClass)

%-----------------------------------------------------------
%  The neural network function
%-----------------------------------------------------------

function net = nnt_classify(net,trn,mu,sigma,num_iter)

rand('state', 20032007);
randn('state', 20032007);

% Now we draw the optimal classification borders
x_train = trn.X;
t_train = trn.y;
mu1 = mu(1,:);
mu2 = mu(2,:);
sigma1 = sigma(1:2,:);
sigma2 = sigma(3:4,:);

[X,Y] = meshgrid(linspace(-4, 5, 200), linspace(-4, 10, 200));
diff1 = ([X(:), Y(:)] - repmat(mu1, length(X(:)), 1));
diff2 = ([X(:), Y(:)] - repmat(mu2, length(X(:)), 1));

p1 = 1 / sqrt(det(sigma1)) * exp(- sum(diff1 * inv(sigma1) .* diff1, 2));
p2 =  1 / sqrt(det(sigma2)) * exp(- sum(diff2 * inv(sigma2) .* diff2, 2));
```

```
p = [p1, p2];
[maxVal, pIndex] = max(p');

pIndex = reshape(pIndex, sqrt(length(pIndex)) , sqrt(length(pIndex)));



% Plot the data
figure;
clf reset;
contourf(X, Y, pIndex);
hold on
gscatter(x_train(1,:), x_train(2, :), t_train, 'rb', 'so');
clear('diff1', 'diff2', 'diff3', 'p1', 'p2', 'p3', 'maxVal', 'pIndex');
pause(3);

% Create the target vector from the class indices
% We need a vector af the form [0 1 0]' for every class index.
T = full(ind2vec(t_train ));
num = num_iter;
for i = 1:num
    % Train the network
    % Here we have to supply the training input and target data (x_train, t_train).
    [net,tr_2hu] = train(net, x_train, T, [],[],[],[]);
    T_learned = sim(net, [X(:)'; Y(:)']);
    [maxVal, T_learned_index] = max(T_learned);
    T_learned_index = reshape(T_learned_index, sqrt(length(T_learned_index)) ,
    sqrt(length(T_learned_index)));

    if (rem(i,50) == 0)
        clf reset;
        contourf(X, Y, T_learned_index);
        hold on
        gscatter(x_train(1,:), x_train(2, :), t_train, 'rb', 'so');
        pause(1);
    end
```

```
    info = sprintf('Iteration:%03d/%03d',i,num);
    disp(info);
end

%-------------------------------------------------------------
% The  support vector machine function
%-------------------------------------------------------------
function model = smo( data, options, init_model)
% SMO Sequential Minimal Optimization for binary SVM with L1-soft margin.
%
% Synopsis:
%  model = smo( data )
%  model = smo( data, options )
%  model = smo( data, options, init_model)
%
% Description:
%  This function is implementation of the Sequential Minimal
%  Optimizer (SMO) [Platt98] to train the binary Support Vector
%  Machines Classifier with L1-soft margin.
%
% Input:
%  data [struct] Binary labeled training vectors:
%   .X [dim x num_data] Training vectors.
%   .y [a x num_data] Labels (1 or 2).
%
%  options [struct] Control parameters:
%   .ker [string] Kernel identifier (default 'linear');
%     See 'help kernel'for more info.
%   .arg [1 x nargs] Kernel argument(s) (default 1).
%   .C Regularization constant (default C=inf). The constant C can
%     be given as:
%      C [1x1] .. for all data.
%      C [1x2] .. for each class separately C=[C1,C2].
%      C [1xnum_data] .. for each training vector separately.
%   .eps [1x1] SMO paramater (default 0.001).
```

```
%    .tol [1x1] Tolerance of KKT-conditions (default 0.001).
%
%   init_model [struct] Specifies initial model:
%    .Alpha [num_data x 1] Initial model.
%    .b [1x1] Bias.
%   If not given then it is set to zero by default.
%
% Output:
%   model [struct] Binary SVM classifier:
%    .Alpha [nsv x 1] Weights (Lagrangians).
%    .b [1x1] Bias.
%    .sv.X [dim x nsv] Support vectors.
%    .nsv [1x1] Number of Support Vectors.
%    .kercnt [1x1] Number of kernel evaluations used by SMO.
%    .trnerr [1x1] Training classification error.
%    .margin [1x1] Margin of the found classifier.
%    .cputime [1x1] Used CPU time in seconds.
%    .options [struct] Copy of used options.
%
% Example:
%   trn = load('riply_trn');
%   model = smo(trn,struct('ker','rbf','C',10,'arg',1));
%   figure; ppatterns(trn); psvm(model);
%   tst = load('riply_tst');
%   ypred = svmclass( tst.X, model );
%   cerror( ypred, tst.y )
%
% See also
%   SVMCLASS, SVMLIGHT, SVMQUADPROG.
%


% About: Statistical Pattern Recognition Toolbox
% (C) 1999-2003, Written by Vojtech Franc and Vaclav Hlavac
% <a href="http://www.cvut.cz">Czech Technical University Prague</a>
% <a href="http://www.feld.cvut.cz">Faculty of Electrical Engineering</a>
% <a href="http://cmp.felk.cvut.cz">Center for Machine Perception</a>
```

```
% Modifications:
% 23-may-2004, VF
% 17-September-2001, V. Franc, created

% timer
tic;

% Input arguments
%------------------------------------------------------------
if nargin < 2,   options = []; else options=c2s(options); end
if ~isfield(options,'ker'), options.ker = 'linear'; end
if ~isfield(options,'arg'), options.arg = 1; end
if ~isfield(options,'C'), options.C = inf; end
if ~isfield(options,'eps'), options.eps = 0.001; end
if ~isfield(options,'tol'), options.tol = 0.001; end

[dim,num_data] = size(data.X);
if nargin < 3,
  init_model.Alpha = zeros(num_data,1);
  init_model.b = 0;
end

% run optimizer
%------------------------------------------------------------
[model.Alpha, model.b, model.nsv, model.kercnt, model.trnerr, model.margin]...
   = smo_mex(data.X, data.y, options.ker, options.arg, options.C, ...
     options.eps, options.tol, init_model.Alpha, init_model.b );

% set up output
%------------------------------------------------------------
inx = find( model.Alpha );
model.sv.X = data.X(:,inx);
model.sv.y = data.y(inx);
model.sv.inx = inx;
model.Alpha = model.Alpha(inx);
```

```
model.Alpha(find(model.sv.y == 2)) = -model.Alpha(find(model.sv.y == 2));

% computes normal vector of the hypeprlane if linear kernel used
if strcmpi(options.ker,'linear'),
  model.W = model.sv.X*model.Alpha;
end

model.options = options;
model.fun = 'svmclass';

model.cputime = toc;

return;



%-------------------------------------------------
%   HW2. Q3
%   a) Classifier uisng the Parzen window
%   b) Classifier using the K-nearest neighbor
%   c) Classifier the nearest neighbor
%-------------------------------------------------
%   JT Gong (03/29/2008)
%-------------------------------------------------
close all;
clear all;
%-- Initial Value Setup --%
Dim2 = 2;

u2_1 = [ 380   400 ];
u2_2 = [ 400   360 ];
covar2_1   = [ 300 20 ; 20 200 ];
covar2_2   = [ 150 15 ; 15 250 ];
%-- Parameter Set up ----%
if 1
    h    =    1;   % The width of parzen window for PARZEN WINDOW
    k    =    5;    % The number of nearest neighbors for KNN
```

```
    N    = 200 ; % The number of total training and test data(Class1-N, Class2-N)
    NT   = 100 ; % The number of training data(Class1_NT, Class2-NT)
end

if 0
    h = input('Enter the width of parzen window => ');
    k = input('Enter the number of nearest neighbors for KNN => ');
    N = input('Enter the number of total generating data(training & test) =>   ');
    NT= input('Enter the number of training data => ');
    %The number of test data = N(total generation data) - NT(training data)
end

%- Generating training and test patterns using random gaussian generation
model.Mu = [u2_1' u2_2'];
model.C = zeros(Dim2,Dim2,2);
model.C(:,:,1) = covar2_1;
model.C(:,:,2) = covar2_2;
model.P = [0.5, 0.5];

XClass1 = mvnrnd(u2_1,covar2_1,N);
XClass2 = mvnrnd(u2_2,covar2_2,N);

TrainPattern     = [ XClass1(1:NT,:)' XClass2(1:NT,:)' ];
TrainLabel       = [ 1*ones(1,NT), 2*ones(1,NT) ];
TestPattern      = [ XClass1(NT+1:N,:)' XClass2(NT+1:N,:)'];
TrueClass        = [ 1*ones(1,N-NT), 2*ones(1,N-NT) ];

%-----------------------------------------------------------------
%   Classifier using the Bayesian Decision Rule
%-----------------------------------------------------------------

%-- Bayesian decision rule : function call --%
PredictedClass_baydec = baydec( TestPattern, model );

%-- The accuracy of the results ---%
Error_Baydec = accuracy(PredictedClass_baydec, TrueClass)
```

```
%-------------------------------------------------------------------
%    Classifier using the Parzen window technique
%-------------------------------------------------------------------

h = input('Enter the width of parzen window => ');

PredictedClass_parzen = parzen(h,TrainPattern,TrainLabel,TestPattern);
Error_ParzenWindow   = accuracy(PredictedClass_parzen, TrueClass)

%-------------------------------------------------------------------
%    Classifier using teh K-nearest neighbor technique
%-------------------------------------------------------------------

k = input('Enter the number of nearest neighbors for KNN => ');
PredictedClass_knn = knn(k,TrainPattern,TrainLabel,TestPattern);
Error_KNearestNeighbor = accuracy(PredictedClass_knn, TrueClass)

 %-- Classifier using the nearest neighbor technique
Range = TrainPattern';  %Range = [data_num dimension]
Range_min = min(Range);
x1_min = Range_min(1);
y1_min = Range_min(2);

Range_max = max(Range);
x1_max = Range_max(1);
y1_max = Range_max(2);

Range = TestPattern';  %Range = [data_num dimension]
Range_min = min(Range);
x2_min = Range_min(1);
y2_min = Range_min(2);

Range_max = max(Range);
x2_max = Range_max(1);
y2_max = Range_max(2);
```

```
x_min = min(x1_min,x2_min);
x_max = max(x1_max,x2_max);
y_min = min(y1_min,y2_min);
y_max = min(y1_max,y2_max);
region5 = max(NT,(N-NT));

region = [x_min x_max y_min y_max];
region = [region region5];
Nclasses = 2;
%-------------------------------------------------------------
%   Classifer using the nearest neighbor technique
%-------------------------------------------------------------
D = nn(TrainPattern,TrainLabel, region);
[x y] = size(D);

[train_err, test_err] = calculate_error (D, TrainPattern, TrainLabel, TestPattern,TrueClass,
region, Nclasses);
%Err = (test_err(2) - train_err(2));
Err = test_err(2);
if(Err < 0 ) Err = 0;
else         Err = Err;
end
Error_TheNearestNeighbor = Err

%-----------------------------------
%% Parzen window classifier function
%-----------------------------------

%1. Dividing the given dataset into test data and trainning data
%2. Generate the covariance matrix with only diagonal elements for the
%    gaussian distribution
%   (Assuming there is no correlationin between the classes)
%3. Take a test sample
%4. Develop parzen window around all the trainning samples
%5. Find the combined probability at the test pattern for all the classes
```

```
%6. The class,which has maximum combined density at the test pattern, will
%    be assigned to the test pattern
%7. Select next test sample and repeat the steps form 3 through 6

function PredictedLabels = parzen(h,TrainPattern,TrainLabel,TestPattern)

%%% Parameters
[dimTrain,TrainHsize]=size(TrainPattern);
[dimTest,TestHsize]=size(TestPattern);
class = 2;
dim = dimTrain;    % dimTrain = dimTest
hn = h / sqrt(TrainHsize) ; % Refer to DHS page 168
Vn = hn^dim;

C = zeros(dim,dim);
for i = 1:dim
    for j = 1:dim
        if(i == j) C(i,j) = hn;
        end
    end
end

PredictedLabels = zeros(1,TestHsize);
Prob = zeros(class,TestHsize);

%%% Parzen window estimation
for i = 1 : TestHsize
     for j = 1 : TrainHsize
       nconst    = 1/((2*pi)^(dim/2) * sqrt(det(C)) );
       Prob(TrainLabel(j),i) = Prob(TrainLabel(j),i) + nconst*exp(-0.5*(TestPattern(:,i)-
TrainPattern(:,j))'*inv(C)*(TestPattern(:,i)-TrainPattern(:,j)) );
     end
end

%%% Choose a class with larger prob. between classland class2
[value, PredictedLabels] = max(Prob);
```

```
return;

%-------------------------------------------------------------------
%% K-nearest Classifier function
%-------------------------------------------------------------------
% Determines distaces of all TrainPattern points from TestPattern points
% Outputs associated with nearest TrainPattern points
%-------------------------------------------------------------------

function PredictedLabels = knn(k,TrainPattern,TrainLabel,TestPattern)

%%% Parameters
[dimTrain,TrainHsize]=size(TrainPattern);
[dimTest,TestHsize]=size(TestPattern);
class = 2;
dim = dimTrain;    % dimTrain = dimTest

PredictedLabels = zeros(1,TestHsize);
sqdist = zeros(1,TrainHsize);

%%% Parzen window estimation
for i = 1 : TestHsize
    for j = 1 : TrainHsize
        sqdist(1,j) = sum( ( TestPattern(:,i) - TrainPattern(:,j) ).^2 );
    end
    [tmp index] = sort(sqdist);
    labels = TrainLabel(index(1:k));
    PredictedLabels(1,i) = round(mean(labels));
end
return;
%-------------------------------------------------------------------
%   Drawing the Voronoi diagram of the training data
%-------------------------------------------------------------------
%   train_features  - Train features
%   train_targets   - Train targets
%   Unused  - Unused
```

```
%   region  - Decision region vector: [-x x -y y number_of_points]
%-------------------------------------------------------------------

function D = nn(train_features, train_targets,region)

% Construct the Voronoi region of the data
D    = voronoi_regions(train_features,region);

mark     = zeros(1,size(train_features,2));
for i = 1:size(train_features,2),
   %For each prototype Xj, find the Voronoi neighbors of Xj
   [x,y] = find(D==i);

   if ~isempty(x),
      %x and y are the locations of the Voronoi region for the i-th prototype
      %These can be used to find the Voronoi neighbors
      around = [x-1 x+1 x x; y y y-1 y+1];
      indices= find((around(:,1)>0) & (around(:,2)<=region(5)) & (around(:,2)>0) &
(around(:,2)<=region(5)));
      around = around(indices,:);

      neighbors = zeros(1,size(around,1));
      for j = 1:length(neighbors),
         neighbors(j) = D(around(j,1),around(j,2));
      end
      neighbors = unique(neighbors);

      %If any neighbor is not from the same class, mark the i-th prototype
      if (length(unique(train_targets(neighbors))) > 1),
         mark(i) = 1;
      end
   end
end

%Discard all unmarked points
prototypes   = find(mark == 1);
```

```
if isempty(prototypes)
   error('No prototypes found')
else
   D = nearest_neighbor(train_features(:,prototypes),train_targets(prototypes),1,region);
end
return;

%-------------------------------------------------------------------
%   The Voronoi diagram function
%-------------------------------------------------------------------

function D = voronoi_regions(features, region)

% Make a Voronoi diagram from sample points
% Inputs:
%   features    - Input data features
%   targets - Input data targets
%   region   - Decision region vector: [-x x -y y number_of_points]

N       = region(5);
x       = linspace (region(1),region(2),N);
y       = linspace (region(3),region(4),N);
D       = zeros(N);
[r,c] = size(features);

y_dist   = (ones(N,1) * features(2,:) - y'*ones(1,c)).^2;
for i = 1:N,
   x_dist = ones(N,1)   * (features(1,:)-x(i)).^2;
   dist    = abs(x_dist + y_dist);
   [sorted_dist, indices] = min(dist');
   D(:,i) = indices(1,:)';
end
return;
```