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.

The criterion function can be written as

$$J(\mathbf{W}) = \frac{\mathbf{w^t S_B w}}{\mathbf{w^t S}_W \mathbf{w}}$$

Where $\mathbf{S_B}$ is the between-class scatter, which shows the scattering degree between 2 classes, and $\mathbf{S}_W$ is the within-class, which shows the scattering degree of the inside of each own class.

Because we can set $\mathbf{w}$ as $\|\mathbf{w}\|$=1, $J(\mathbf{W}) = \mathbf{w^t S_B w}$ means

$$J(\mathbf{W}) = \mathbf{w^t S_B w} = \frac{\mathbf{w^t S_B w}}{\|\mathbf{w}\|^2} = \frac{\mathbf{w^t S_B w}}{\mathbf{w}^t \mathbf{I w}},$$ that is, $\mathbf{S}_W$ is the identity matrix.

By Equation (106) in DHS, the solution to optimizes $J(\bullet)$ is

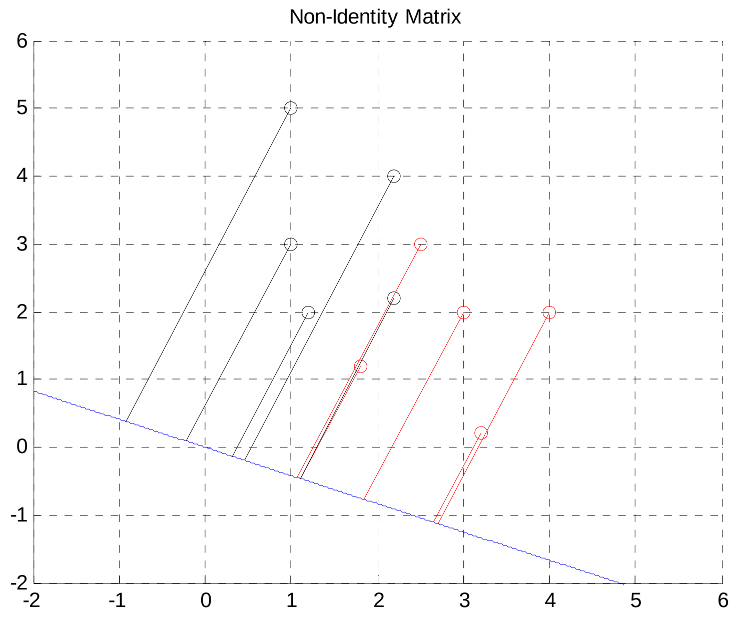
$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m_1} - \mathbf{m_2})$$

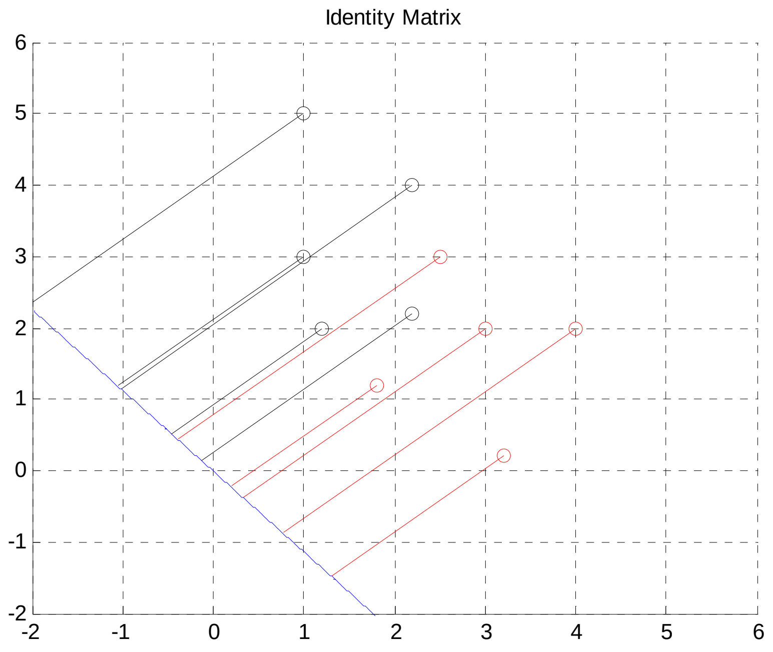
And, if $\mathbf{S}_W$ is the identity

$$\mathbf{w} = (\mathbf{m_1} - \mathbf{m_2})$$

Where, $\mathbf{m_1}, \mathbf{m_2}$ are sample means.

The results are following





Based on above graph, both two results are good. Nevertheless,

we can say that the Non-identity matrix shows better result.
The Non-identity shows greater separation between the red and black projected points.

The within scattering matrix $\mathbf{S}_W$ is the sum of two covariance matrix of two classes. That is,
$\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$
In this problem, $\mathbf{S}_W$ does not satisfy the identity matrix. So, $\mathbf{w} = (\mathbf{m}_1 - \mathbf{m}_2)$ cannot maximize the criterion function $J(\bullet)$.

<Matlab Code>

```
clear all;
close all;
clc;

Sample_size = 5;



%Samples
Sample1 = [1 3 ;1 5 ; 1.2 2;2.2 2.2; 2.2 4];
Sample2 = [1.8 1.2;2.5 3;3 2; 3.2 0.2; 4 2];



%Non-identity matrix
m1      = sum(Sample1)/Sample_size;
m2      = sum(Sample2)/Sample_size;
s1      =  cov(Sample1);
s2      =  cov(Sample2);
sw      = s1 + s2;
w       = sw^(-1)*(m1-m2)'; %The solution to optimize the criterion function J()



x       = linspace(-2,5,1000);

y       = w(2)/w(1) * x  ;



hold on
AXIS([-2 6 -2 6])
plot(Sample1(:,1),Sample1(:,2),'ko', Sample2(:,1),Sample2(:,2),'ro')
plot(x,y)
title('Non-Identity Matrix')

for i=1:1000
   w_line(i,:)=[x(i),y(i)];
end
```

```
for j=1:5
   for i=1:1000
      Sample1_norm(j,i)=norm(w_line(i,:)-Sample1(j,:));
      Sample2_norm(j,i)=norm(w_line(i,:)-Sample2(j,:));
   end
end

[val,Sample1_ind]=min(Sample1_norm');
[val,Sample2_ind]=min(Sample2_norm');
for i=1:5

plot([Sample1(i,1),w_line(Sample1_ind(i),1)],[Sample1(i,2),w_line(Sample1_ind(i),2)],
'k-');

plot([Sample2(i,1),w_line(Sample2_ind(i),1)],[Sample2(i,2),w_line(Sample2_ind(i),2)],
'r-');
end
grid;
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Identity matrix
w      = (m1-m2);   %The solution to optimize the criterion function J()
x      = linspace(-3,3,1000);
y      = w(2)/w(1) * x ;

figure;
hold on
AXIS([-2 6 -2 6])
plot(Sample1(:,1),Sample1(:,2),'ko', Sample2(:,1),Sample2(:,2),'ro')
plot(x,y)
title('Identity Matrix')

for i=1:1000
   w_line(i,:)=[x(i),y(i)];
end

for j=1:5
   for i=1:1000
      Sample1_norm(j,i)=norm(w_line(i,:)-Sample1(j,:));
      Sample2_norm(j,i)=norm(w_line(i,:)-Sample2(j,:));
   end
end

[val,Sample1_ind]=min(Sample1_norm');
[val,Sample2_ind]=min(Sample2_norm');
for i=1:5

plot([Sample1(i,1),w_line(Sample1_ind(i),1)],[Sample1(i,2),w_line(Sample1_ind(i),2)],
'k-');

plot([Sample2(i,1),w_line(Sample2_ind(i),1)],[Sample2(i,2),w_line(Sample2_ind(i),2)],
'r-');
end
grid;
hold off
```

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.

The neural network approach

In artificial neural network, we assume three layers: input, hidden, and output layer. Each layer includes several neurons and neurons interconnect each other. When transfer information from one to others, we have to consider weighting factors. Through back-propagation, we update the error of weighting factors and evolve the overall system based on training.

First we define a cost function to measure the error of the neural network with weighting $\vec{w}$. The cost function can be written as below

$$J(\vec{w}) = \frac{1}{2}\sum_k (t_k - z_k)^2 = \frac{1}{2}\left|\vec{t} - \vec{k}\right|^2$$

Then, we can optimize this cost function using gradient method

$$\text{new } \vec{w} = \text{old } \vec{w} + \Delta\vec{w}$$

The support vector machine approach

The objective of support vector machine is to find the hyperplanes with maximum margin among classes. Any hyperplane can be written as the set of points satisfying

$$\mathbf{w} \bullet \mathbf{x} - b = 0$$

Where $\mathbf{w}$ is a normal vector and $\mathbf{x}$ is training data. b is a bias unit to determine the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$.

We can find the class for test data through criterion that,

$$\text{if } \mathbf{w} \bullet \mathbf{x}_i - b \geq 1, \text{ then } \mathbf{x}_i \text{ is class1.}$$
$$\text{if } \mathbf{w} \bullet \mathbf{x}_i - b \geq -1, \text{ then } \mathbf{x}_i \text{ is class2.}$$

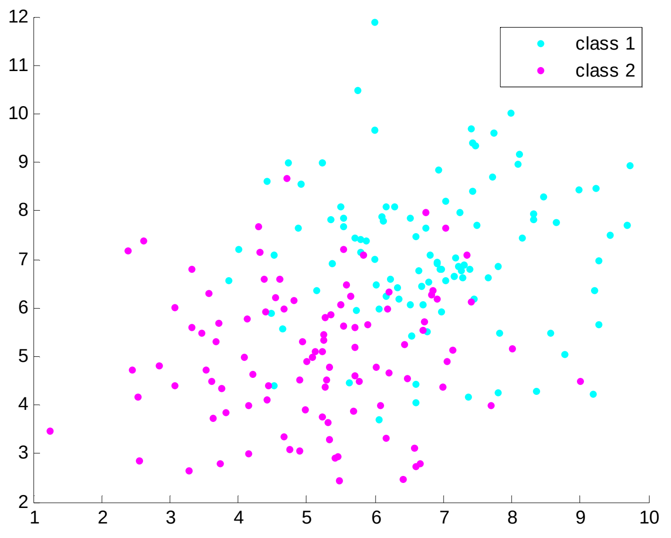
We choose $\mathbf{w}$ and b, to minimize $|\mathbf{w}|$, subject to $c_i(\mathbf{w} \bullet \mathbf{x}_i - b) \geq 1$, but generally, it is hard to find arguments to minimize $|\mathbf{w}|$. So, instead we change our problem into

$$\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2, \text{ subject to } c_i(\mathbf{w} \bullet \mathbf{x}_i - b) \geq 1.$$

Where, c_i indicates the classes that can be either 1 or -1.

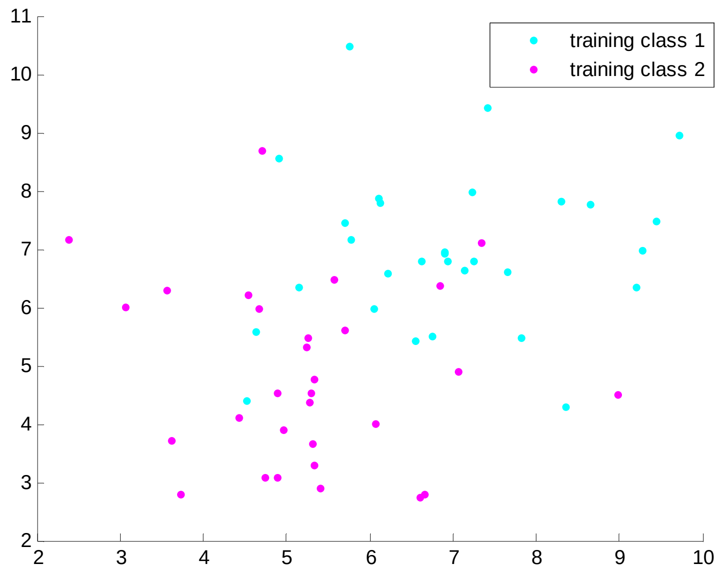
Simulation

First, we set 200 samples composed of 2 classes.

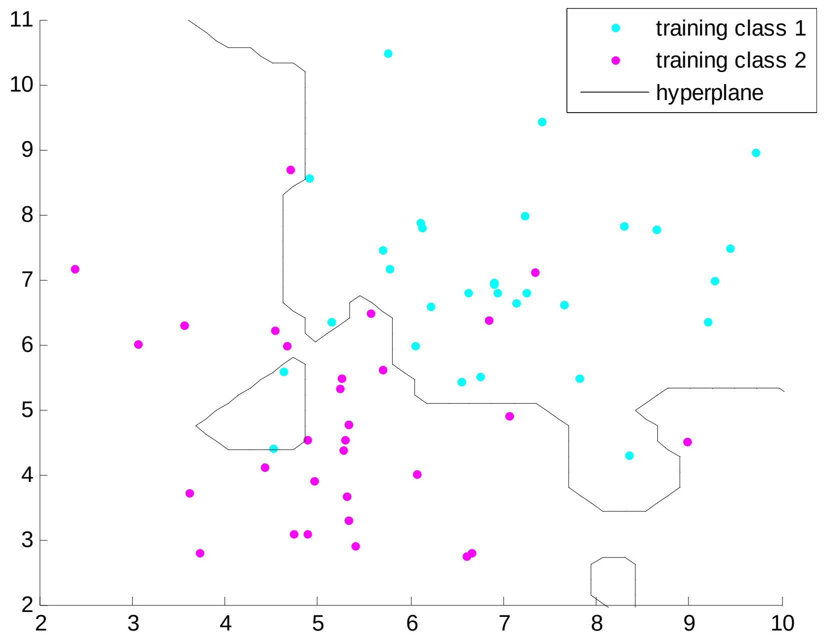


Among 200 samples, we set 60 samples as training data, and the others as test data.

Training data set are

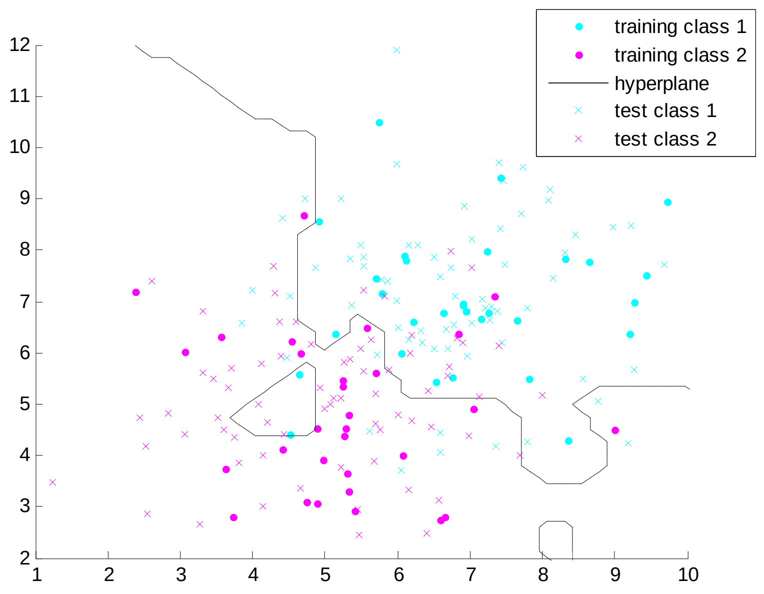


For neural network, I iterated and trained 30 times to obtain hyperplane below



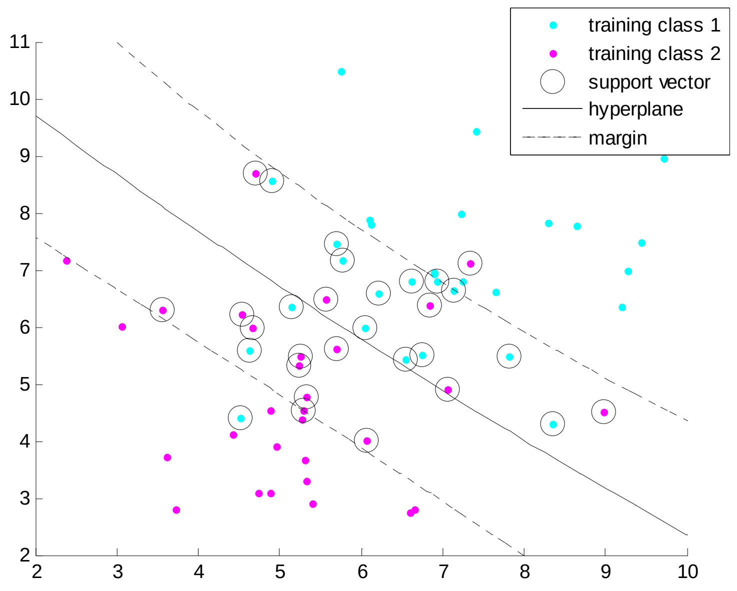
<Neural Network>

Now through above result, we decide the class of the remained test data.

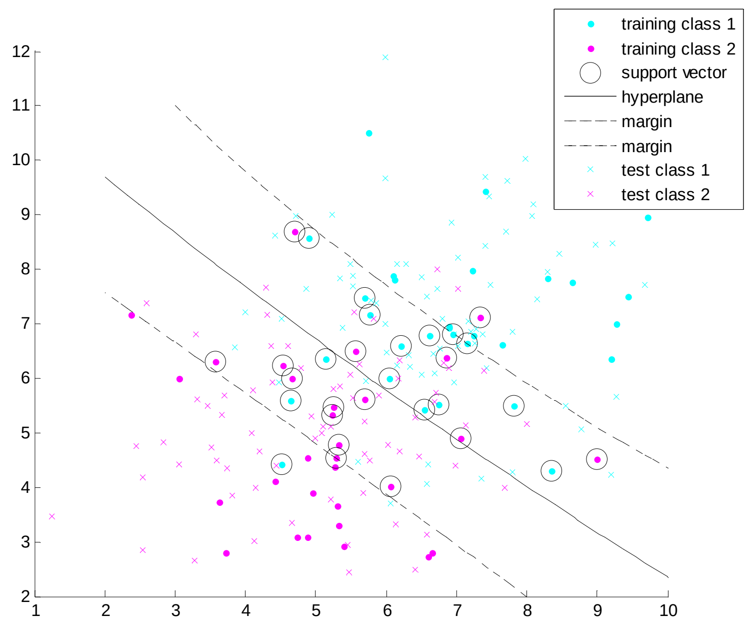


As shown in this figure, some samples are false classified.

Next we find hyperplane based on the support vector machine (SVM)



Through above SVM result, we decide the class of the remained test data.



As shown in this figure, some samples are false classified.

c) Compare the two approaches.

The error rate of neural network is 0.2214.
And the error rate of SVM is 0.1786.
In this homework, I made use of the Gaussian distribution to generate samples. SVM divided two patterns linearly, so in this problem, SVM shows better result. However, the distribution of class are nonlinear, neural network can show better result.

<Matlab code>

```
clear all; close all; clc;

%% add path
addpath('./nn');
addpath('./stprtool');
stprpath('./stprtool');

hold on;

n=60;
%data
sample1=mvnrnd([7 7],[2 0;0 2],100);
sample2=mvnrnd([5 5],[2 0;0 2],100);



hold on;
plot(sample1(1:100,1),sample1(1:100,2),'c.');
plot(sample2(1:100,1),sample2(1:100,2),'m.');
legend('class 1','class 2');
hold off;

%training data
training(1:n/2,:)=sample1(1:n/2,:);
training(n/2+1:n,:)=sample2(1:n/2,:);



%test data
test(1:70,:)=sample1(31:100,:);
test(71:140,:)=sample2(31:100,:);



figure;
hold on;
```

```
plot(training(1:n/2,1),training(1:n/2,2),'c.');
plot(training((n/2+1):n,1),training((n/2+1):n,2),'m.');
legend('training class 1','training class 2');




%%%%%%%%%%%%%%%%%%%%Neural Network Test

%% classify using nn
nn_layer = [30, 2];
num_iter = 30;

grouping(1:n/2)=-ones(n/2,1);
grouping(n/2+1:n)=ones(n/2,1);
grouping=grouping';



trn.X = training';
trn.nanme = 'train';
trn.y = grouping';
trn.dim = 2;
trn.num_data = n/2;

trn.y=1/2*trn.y+1.5;

mu=[7 7; 5 5];
sigma=[[2 0;0 2];[2 0;0 2]];



net = newff(minmax(training'), [50, 2], {'logsig', 'logsig'}, 'trainbfg');
net.performFcn = 'mse';
net.trainParam.epochs = 5;
net.trainParam.show = NaN;
net = init(net);
net = nnt_classify(net,trn,mu,sigma,num_iter);
legend('training class 1','training class 2','hyperplane');
out = sim(net,test');
[maxVal,nn_test_idx] = max(out);
t_test(1:70)=1;
t_test(71:140)=2;
plot(test(1:70,1),test(1:70,2),'cx');plot(test(71:140,1),test(71:140,2),'mx');
legend('training class 1','training class 2','hyperplane','test class 1','test class
2');
cerror(nn_test_idx,t_test)



%%%%%%%%%%%%%%%%%%%%%%%%%
%% classify using svm
options.ker = 'rbf';       % use RBF kernel
%options.ker = 'linear';       % use RBF kernel
options.arg = 10;          % kernel argument
options.C = 10;           % regularization constant
model = smo(trn,options);
figure;
%ppatterns(trn);
figure;hold
on;plot(training(1:30,1),training(1:30,2),'c.');plot(training(31:60,1),training(31:60
,2),'m.');
psvm(model);
legend('training class 1','training class 2','support vector','hyperplane','margin');
```

```
svm_test_idx = svmclass(test',model);
plot(test(1:70,1),test(1:70,2),'cx');plot(test(71:140,1),test(71:140,2),'mx');
legend('training class 1','training class 2','support
vector','hyperplane','margin','margin','test class 1','test class 2');
cerror(svm_test_idx,t_test)
```

```
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(0, 14, 60), linspace(0, 14, 60));
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;


hold on
plot(x_train(1,1:30),x_train(2, 1:30),'c.');
plot(x_train(1,31:60),x_train(2, 31:60),'m.');
contour(X, Y,pIndex,1,'k');
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 (i==num)
```

```
        clf reset;

        hold on
        plot(x_train(1,1:30),x_train(2, 1:30),'c.');
        plot(x_train(1,31:60),x_train(2, 31:60),'m.');
        contour(X, Y,T_learned_index,1,'k');

        %gscatter(x_train(1,:), x_train(2, :), t_train, 'rb', '..');
        pause(1);
    end

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

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.

We assume the distribution of each sample follow the 2-dimensioanl multivariate normal density. So, the widow function for 2-dimensional space can be expressed as

$$\varphi(\mathbf{u}) = \frac{1}{(\sqrt{2\pi})^2 |\Sigma|^{1/2}} \exp[-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu})^t \Sigma^{-1}(\mathbf{u}-\boldsymbol{\mu})]$$

And the length of an edge of hypercube, h_n can be expressed as $h_n = h_1/\sqrt{n}$

The average of normal densities centered at the each samples is

$$p_n(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h_n}\varphi(\frac{\mathbf{x}-\mathbf{x_i}}{h_n})$$

Where, n is the number of training data.

For test data, we estimate $p_n(\mathbf{x})$ from training data. I have 2-dimensional samples composed 2 classes. Therefore, we can obtain two $p_n(\mathbf{x})$.

Between two $p_n(\mathbf{x})$, we declare test data as the class which has greater value. That is, data $\mathbf{x}$ is declared as class 1, if $p_n(\mathbf{x})\big|_{class\ 1} \geq p_n(\mathbf{x})\big|_{class\ 2}$, and vise versa.

In Matlab, this algorithm was be implemented as

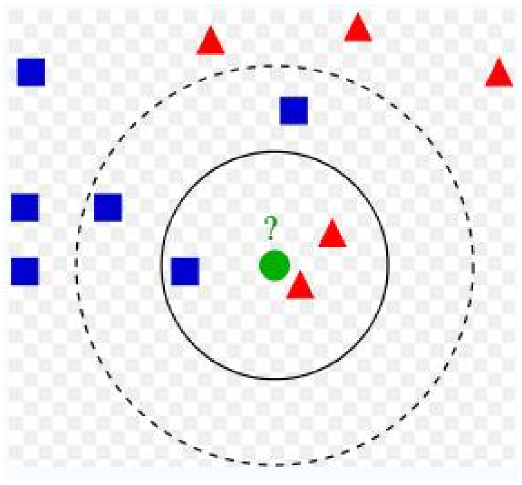
```
for j = 1 : TrainHsize
      nconst   = 1/((2*pi)^(dim/2) * sqrt(det(C)) );
      Prob(TrainLabel(j),i) = Prob(TrainLabel(j),i) + nconst*exp(-
0.5*(TestPattern(:,i)-TrainPattern(:,j))'*inv(C)*(TestPattern(:,i)-
TrainPattern(:,j)) );
    end
end

[value, PredictedLabels] = max(Prob);
```

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

K-nearest neighbor is a technique to count the number of the nearest training data from the target test data. If we have 2 classes, we count the training data belonging to each class, and declare the test data as the class which have more numbers. To estimate the distance from the position of test data, we simply use the Euclidean distance (l_2 norm).

The below figure show the algorithm of K-nearest neighbor.



In Matlab code, it was implemented as the following

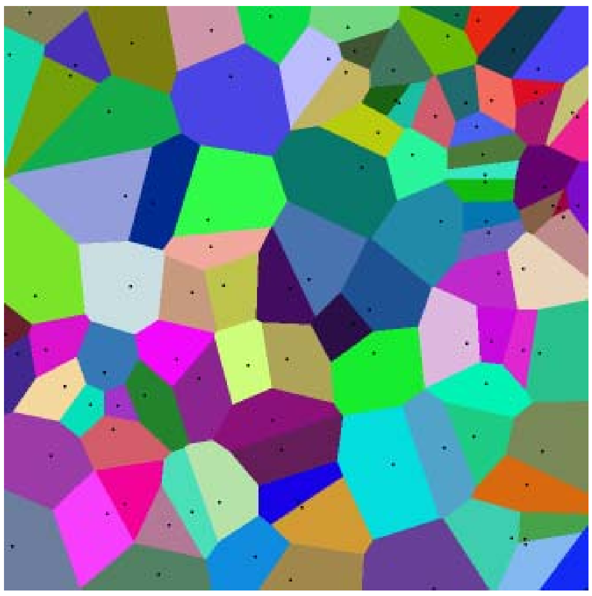
```
for i = 1 : TestHsize
    for j = 1 : TrainHsize
        sqdist(1,j) = sum( ( TestPattern(:,i) - TrainPattern(:,j) ).^2 );
    end
    [tmp index] = sort(sqdist);
    labels = TrainLabel(index(1:k));
    PredictedLabels(1,i) = round(mean(labels));
end
```

c) Design a classifier using the nearest neighbor technique.

Brief algorithm of the nearest neighbor is following,

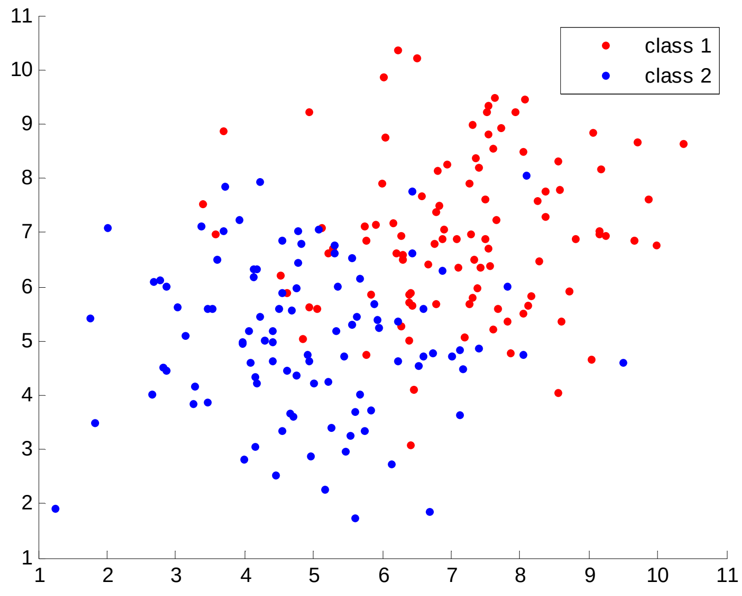
```
begin initialize : j=0, D=data set, n=number of prototypes.
      construct the full Voronoi diagram of D
            do j=j+1; for each prototype $\mathbf{x}'_j$
                 Find the Voronoi neighbors of $\mathbf{x}'_j$
                 if any neighbor is not from the same class as $\mathbf{x}'_j$, then mark $\mathbf{x}'_j$
            until j==n
      discard all points that are not marked
      construct the Voronoi diagram of the remaining prototypes
end
```

Where, Voronoi diagram (Voronoi tesselation) is a special kind of decomposition of a metric space determined by distances to a specified discrete set of objects in the space.

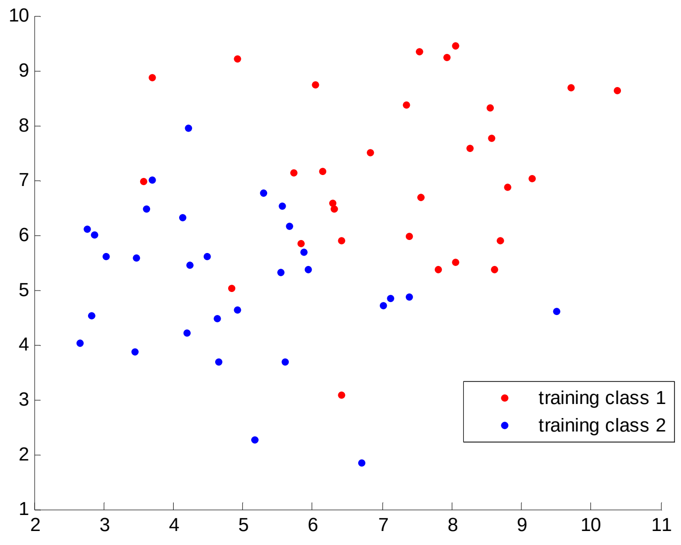


d) Compare the three approaches.

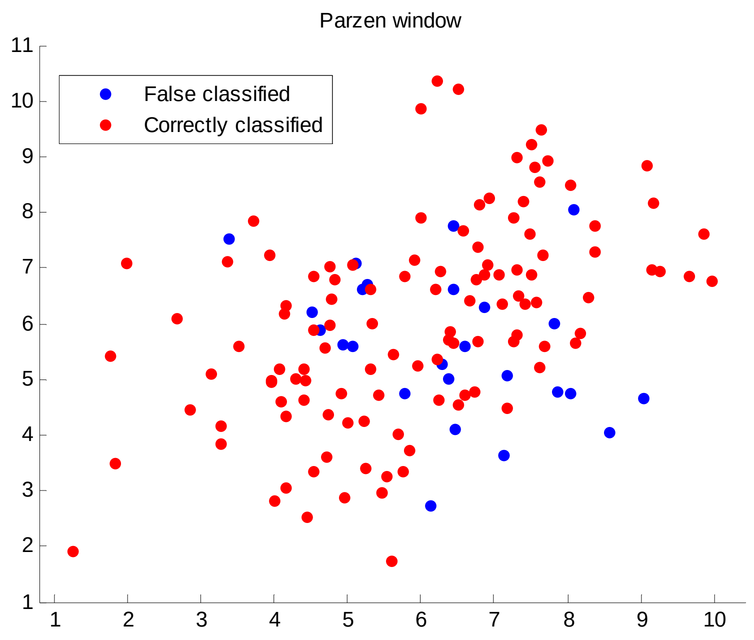
First, generate the 200 sample data set

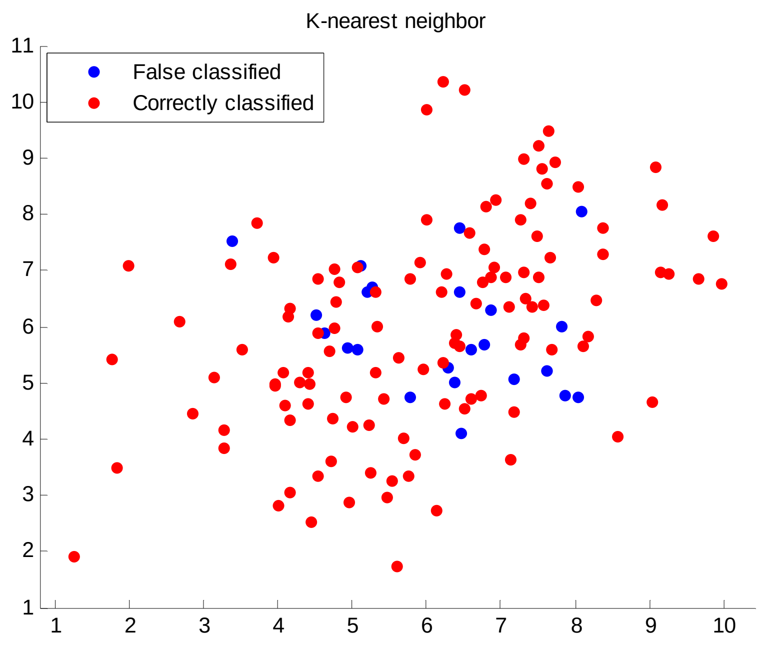


And we set the 60 training data as



At the Parzen window, K-nearest neighbor and Nearest neighbor, there exist several errors (false classified samples).


Parzen window
False classified
Correctly classified
11
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10


K-nearest neighbor
False classified
Correctly classified
11
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10

The error rate of the each approach is,

Error_ParzenWindow =

    0.1571

Error_KNearestNeighbor =

    0.1500

Error_TheNearestNeighbor =

    0.0952

From above result, the nearest neighbor show the best result among three simulations.

<Matlab code>

```
close all;
clear all;
clc;
%-- Initial Value Setup --%

%data

Dim2 = 2;
u2_1 = [ 7  7 ];
u2_2 = [ 5  5 ];
covar2_1  = [ 2 0  ; 0 2 ];
covar2_2  = [ 2 0  ; 0 2 ];

%-- Parameter Set up ----%

   h   = 1;   % The width of parzen window for PARZEN WINDOW
   k   = 5;    % The number of nearest neighbors for KNN
   N   = 100 ; % The number of total training and test data(Class1-N, Class2-N)
```

```
    NT  = 30 ; % The number of training data(Class1_NT, Class2-NT)




%- Generating training and test patterns using random gaussian generation
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) ];


figure;
hold on;
plot(XClass1(1:N,1),XClass1(1:N,2),'r.');
plot(XClass2(1:N,1),XClass2(1:N,2),'b.');
legend('class 1','class 2');
hold off;



figure;
hold on;
plot(TrainPattern(1,1:NT),TrainPattern(2,1:NT),'r.');
plot(TrainPattern(1,NT+1:2*NT),TrainPattern(2,NT+1:2*NT),'b.');
legend('training class 1','training class 2');
hold off;

%-----------------------------------------------------------------------
%    Classifier using the Parzen window technique
%-----------------------------------------------------------------------
PredictedClass_parzen = parzen(h,TrainPattern,TrainLabel,TestPattern);
Error_ParzenWindow  = accuracy(PredictedClass_parzen, TrueClass)


figure; hold on;
gscatter(TestPattern(1,:),TestPattern(2,:),(PredictedClass_parzen==TrueClass),'br','.
.')
legend('False classified','Correctly classified');


title('Parzen window');
hold off;

%-----------------------------------------------------------------------
%    Classifier using teh K-nearest neighbor technique
%-----------------------------------------------------------------------
PredictedClass_knn = knn(k,TrainPattern,TrainLabel,TestPattern);
Error_KNearestNeighbor = accuracy(PredictedClass_knn, TrueClass)


figure; hold on;
gscatter(TestPattern(1,:),TestPattern(2,:),(PredictedClass_knn==TrueClass),'br','..')
legend('False classified','Correctly classified');


title('K-nearest neighbor');
hold off;


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

```
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 class1and class2
[value, PredictedLabels] = max(Prob);
return;
```

```
 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);

%%% K-nearest neighbor 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;
```

```
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;
```

```
function D = nearest_neighbor(train_features, train_targets, Knn, region)

% Classify using the Nearest neighbor algorithm
% Inputs:
%   features    - Train features
%   targets - Train targets
%   Knn     - Number of nearest neighbors
%   region  - Decision region vector: [-x x -y y number_of_points]
%
% Outputs
%   D          - Decision sufrace

L          = length(train_targets);
N          = region(5);
x          = linspace (region(1),region(2),N);
y          = linspace (region(3),region(4),N);

D          = zeros(N);

if (L < Knn),
   error('You specified more neighbors than there are points.')
end

y_dist  = (ones(N,1) * train_features(2,:) - y'*ones(1,L)).^2;

for i = 1:N,
   x_dist = ones(N,1)  * (train_features(1,:)-x(i)).^2;
    dist   = abs(x_dist + y_dist);
   [sorted_dist, indices] = sort(dist');
   k_nearest = train_targets(indices(1:Knn,:));
   if (Knn > 1),
      D(:,i)    = (sum(k_nearest) > Knn/2)';
   else
      D(:,i)  = (k_nearest');
   end
end
return;
```

```
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;
```

Reference

I referred some Matlab codes from the following website

1. Nearest neighbor approach:
http://stuff.mit.edu/afs/sipb.mit.edu/user/arolfe/matlab/

2. Support vector machine:

Statistical Pattern Recognition Toolbox

http://cmp.felk.cvut.cz/cmp/software/stprtool/index.html