ECE 662 **APRIL 1, 2008**
HOMEWORK 2

Question 1: In the Parametric Method section of the course, we learned how to draw a separation hyperplane between two classes by obtaining w_0, the argmax of the cost function $J(w)=w^TS_Bw$ / w^TS_ww. 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 $J(w)= w^TS_Bw$ instead (i.e. setting S_w as the identity matrix in the solution w_o? Investigate this question by numerical experimentation.

The Fisher's discriminant provides information about the separation between the means of two classes as well as the within-class spreads of the data points. Let's recall that the numerator of the cost function $J(w)$ is the squared distance of the projected means onto the vector *w*. Furthermore, such numerator can be written as seen in equation 1.

$$(m_{p2}-m_{p1})^2 = w^T.(m_2-m_1)(m_2-m_1)^T.w = w^TS_Bw \qquad \textbf{Equation 1}$$

The projected means for class 2 and 1 are represented by m_{p2}, m_{p1}, respectively, m_2, m_1 are the original means of the data, *w* is the vector on which the data is projected and the matrix S_B is called the between-class covariance matrix.

On the other hand, the denominator of Fisher's discriminant is the sum of the within-class scatters of the projected data. Equation 2 shows the relationship between the within-class scatters C_1, C_2 of the original data, and the projected ones C_{p1}, C_{p2} through the projection vector *w*. The matrix S_w is called the total within-class covariance matrix.

$$C_{p1}+C_{p2} = w^T.(C_1+C_2).w = w^TS_ww \qquad \textbf{Equation 2}$$

Notice that if we were going to use only the denominator to find the *w* that maximizes the cost function $J(w)$, we would be finding the best w that gives us the greatest separation between means. Although this approach is alluring, it is a naïve one, since it doesn't provide any information about the spread of the data.

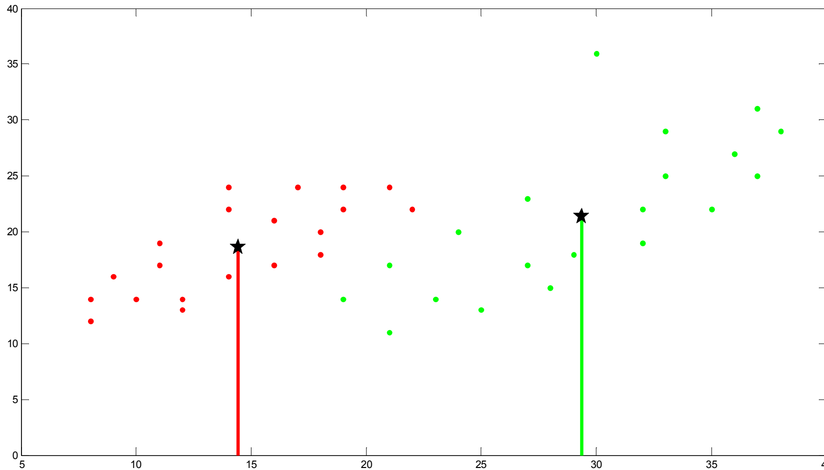

Figure 1. Class 1 is represented by the red dots and class 2 is denoted by the green dots. The black stars are the means of each class. The lines are the projections of the means onto the x-axis. The distance between the projected means is 14.09

In Figure 1 we can see how finding the greatest distance of the projected means doesn't yield the best separation between classes necessarily. According to the numerator of $J(w)$, the vector w_o that maximizes such expression for our data is the x-axis. If we project all the data on such vector, we will have misclassification inside the circle in Figure 2.

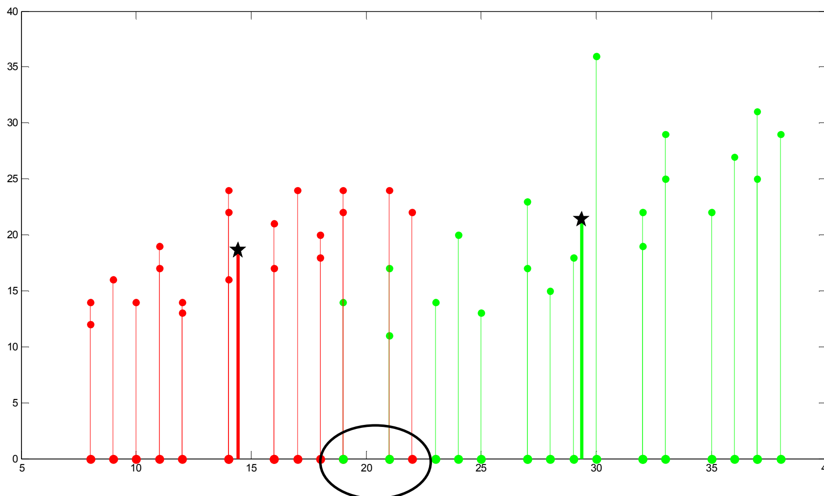

Figure 2. Projected data on x-axis. Red dots belong to class 1 and the green ones to class 2. The data inside the circle is misclassified.

Contrarily, when we maximize the whole expression for $J(w)$, we achieve at the same time a good separation between classes and a considerable distance among their means in the projected data. Observe that on Figure 3 the misclassification has decreased compare to the first case. The vector w_o was found using equation 3.

$$w_o = S_w^{-1}(m_2 - m_1) \quad \textbf{Equation 3}$$

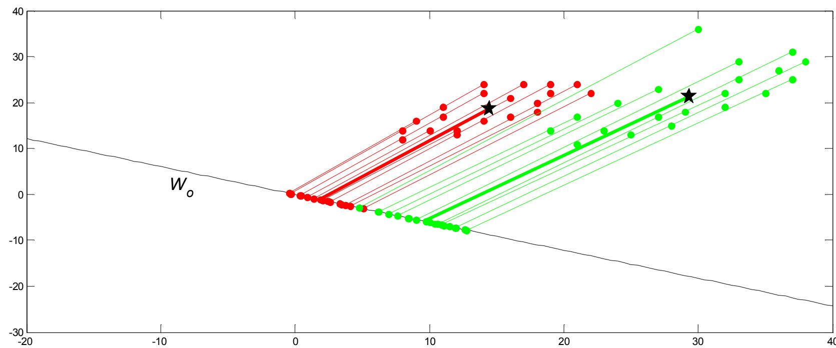

Figure 3. The data from class 1 and 2 is projected on the vector w_o. The misclassification occurs just on one sample. The distance between the projected means is 9.07.

It is important to add that the distance between the projected means on the second case is not the maximum, but it is enough to separate the data more accurately than on the first case.

The set of data used on the above analysis was found on [1].

MATLAB code

```
%%% First Question for HW 2 %%%
%%% Data fromTextbook: Pattern Recognition and Image Processing
page186, BOW

w1 = [ 8 12; 8 14; 9 16; 10 14; 12 13; 12 14; 11 17; 11 19; 14 24;...
    14 22; 16 21; 14 16; 16 17; 18 18; 18 20; 17 24; 19 22; 19 24;...
    21 24; 22 22];
w2 = [21 11; 19 14; 21 17; 23 14; 25 13; 24 20; 27 17; 28 15; 29
18;...
    27 23; 32 22; 30 36; 32 19; 33 25; 33 29; 35 22; 36 27; 37 25;...
    38 29; 37 31];

m1 = mean(w1);
m2 = mean(w2);

% The Fisher's determinant produces just 1 missclassified sample
C1 = (w1-[m1(1).*ones(20,1) m1(2).*ones(20,1)])'*(w1-
[m1(1).*ones(20,1) m1(2).*ones(20,1)])./20;
C2 = (w2-[m2(1).*ones(20,1) m2(2).*ones(20,1)])'*(w2-
[m2(1).*ones(20,1) m2(2).*ones(20,1)])./20;
Sw = C1 + C2;
Sw_i = inv(Sw);
Sb = (m1 - m2)'*(m1 - m2);
%%% Finding w_o
w_o = Sw_i*(m2'-m1')

%%% Projection of the classes into the vector w_o
x1 = w_o'*w1';
x2 = w_o'*w2';
y1 = w_o(2)/w_o(1).*x1;
y2 = w_o(2)/w_o(1).*x2;

%%% Visualizatio of projections and projection line
plot(w1(:,1),w1(:,2),'r.')
```

```
hold on;plot(w2(:,1),w2(:,2),'g.')
l = (w_o(2)/w_o(1)).*(-20:0.5:40);
hold on; plot((-20:0.5:40),l,'b')
hold on; plot(x1,y1,'r.')
hold on; plot(x2,y2,'g.')

for i = 1:length(x1)
    %%% orthogonal vectors to the w_o vector
    line([x1(i) w1(i,1)], [y1(i) w1(i,2)],'Color',[1 0 0])
    line([x2(i) w2(i,1)], [y2(i) w2(i,2)],'Color',[0 1 0])
end

x_m1 = w_o'*m1';
x_m2 = w_o'*m2';
y_m1 = w_o(2)/w_o(1).*x_m1;
y_m2 = w_o(2)/w_o(1).*x_m2;
hold on;plot(m1(1),m1(2),'k*','MarkerSize',10)
hold on;plot(m2(1),m2(2),'k*','MarkerSize',10)
line([x_m1 m1(1)],[y_m1 m1(2)],'Color',[1 0 1],'LineWidth',3)
line([x_m2 m2(1)],[y_m2 m2(2)],'Color',[1 0 1],'LineWidth',3)
distance_w_o = norm([x_m1 y_m1]-[x_m2 y_m2])

% Projection of means over the x-axis, the separation between means
is greater but the classification is NOT highly successful; 4
misclassified samples
figure;
plot(w1(:,1),w1(:,2),'r.')
hold on;plot(w2(:,1),w2(:,2),'g.')
hold on;plot(m1(1),m1(2),'k*','MarkerSize',10)
hold on;plot(m2(1),m2(2),'k*','MarkerSize',10)
line([m1(1) m1(1)],[0 m1(2)],'Color',[1 0 1],'LineWidth',3)
line([m2(1) m2(1)],[0 m2(2)],'Color',[1 0 1],'LineWidth',3)
for i = 1:length(x1)
    %%% orthogonal vectors to the w_o vector
    line([w1(i,1) w1(i,1)], [0 w1(i,2)],'Color',[1 0 0])
    line([w2(i,1) w2(i,1)], [0 w2(i,2)],'Color',[0 1 0])
end

distance_x_axis = m2(1)-m1(1)
```

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.
Note: you may use code downloaded from the web, but if you do so, please be sure to explain what the code does in your report and give the reference.

Neural Network (NN)

The NN training code used was found in [2]. It was modified to perform the classification stage and follow our requirements.

The code trains a 3-layer NN through back projection on dimension 2. On such training method, we have a set composed by inputs and outputs, and an initial array of weights. The weights are adjusted until the current output matches or gets close to the output given on the training set.

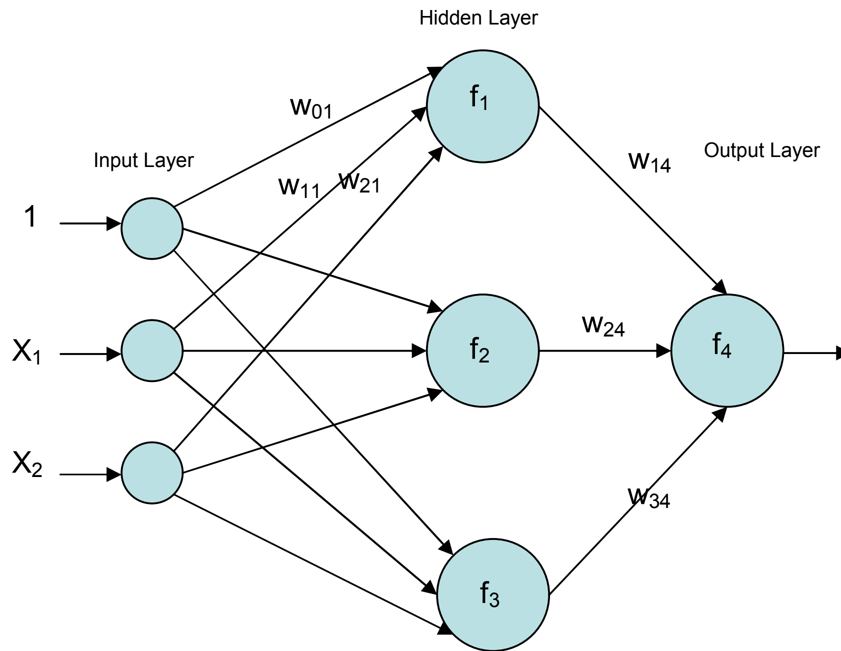

Figure 4. Diagram of the 3-layer Neural Network. f_1, f_2, f_3 are the hidden layer functions, we are using $f_i(x) = \tanh(x)$ for every *i*. f_4 is just a sign function

The optimization of the weights is done using the gradient descent algorithm. The stopping criteria are the number of iterations (10000) and a tolerance error difference on the output, whichever is achieved first stops the optimization process. There is a plot on Figure 5 showing how the error on the output decreases on each iteration of the gradient descent algorithm.

The hidden layer function is $f_i(x) = \tanh(x)$, for every *i=1,2,3*. The output layer function is a sign function.

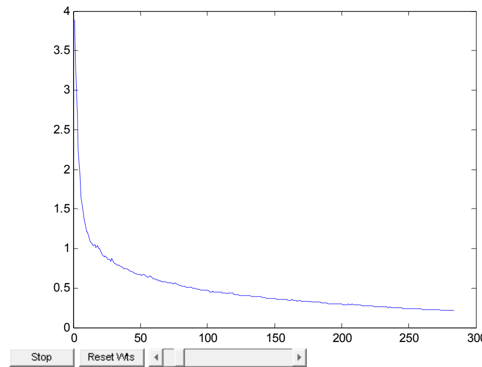

Figure 5. Screenshot showing how the error on the output decreases with each iteration of the gradient descent algorithm performed on the weights

After the weights w_{ij} are computed, the network is complete and we can proceed to classify the samples on the test set.

The data for the training and test set was generated randomly using a Gaussian distribution with unity variance and different means for each class. The initial value of the weights is also Gaussian distributed with a zero mean and a variance of 10.

On the experiments, it is evident that if the variance on both classes is fixed to 1 and we increase the difference between means, we will have a better classification. But when the difference between means is low, 1 or 2 units, the error rate is between 10% and 40%

MATLAB code

```
%---------------------------------------------------------
% MATLAB neural network backprop code
% by Phil Brierley
% www.philbrierley.com
% 29 March 2006
%
% Modified by me
% March 28, 2008
%
% This code implements the basic backpropagation of
% error learning algorithm. The network has tanh hidden
% neurons and a linear output neuron.
%
% adjust the learning rate with the slider
%
% feel free to improve!
%
%--------------------------------------------------------
%user specified values
hidden_neurons = 3;
epochs = 10000;
% ------- load in the data -------
% XOR data
% train_inp = [1 1; 1 0; 0 1; 0 0];
```

```
% train_out = [1; 0; 0; 1];

m = 10;
t1 = [randn(m,1);1+randn(m,1)];
t2 = [randn(m,1);1+randn(m,1)];
train_inp = [t1 t2];
train_out = [zeros(m,1); ones(m,1)];

% check same number of patterns in each
if size(train_inp,1) ~= size(train_out,1)
    disp('ERROR: data mismatch')
   return
end

%standardise the data to mean=0 and standard deviation=1
%inputs
% mu_inp = mean(train_inp);
% sigma_inp = std(train_inp);
% train_inp = (train_inp(:,:) - mu_inp(:,1)) / sigma_inp(:,1);
%
%outputs
train_out = train_out';
mu_out = mean(train_out);
sigma_out = std(train_out);
train_out = (train_out(:,:) - mu_out(:,1)) / sigma_out(:,1);
train_out = train_out';

%read how many patterns
patterns = size(train_inp,1);

%add a bias as an input
bias = ones(patterns,1);
train_inp = [train_inp bias];

%read how many inputs
inputs = size(train_inp,2);

%---------- data loaded ------------


%--------- add some control buttons ---------

%add button for early stopping
hstop = uicontrol('Style','PushButton','String','Stop', 'Position',
[5 5 70 20],'callback','earlystop = 1;');
earlystop = 0;

%add button for resetting weights
hreset = uicontrol('Style','PushButton','String','Reset Wts',
'Position', get(hstop,'position')+[75 0 0 0],'callback','reset =
1;');
reset = 0;

%add slider to adjust the learning rate
hlr =
uicontrol('Style','slider','value',.1,'Min',.01,'Max',1,'SliderStep',
[0.01 0.1],'Position', get(hreset,'position')+[75 0 100 0]);


% ---------- set weights -----------------
%set initial random weights
```

```
weight_input_hidden = (randn(inputs,hidden_neurons) - 0.5)/10;
weight_hidden_output = (randn(1,hidden_neurons) - 0.5)/10;

%-----------------------------------
%--- Learning Starts Here! ---------
%-----------------------------------

%do a number of epochs
for iter = 1:epochs

    %get the learning rate from the slider
    alr = get(hlr,'value');
    blr = alr / 10;

    %loop through the patterns, selecting randomly
    for j = 1:patterns

        %select a random pattern
        patnum = round((rand * patterns) + 0.5);
        if patnum > patterns
            patnum = patterns;
        elseif patnum < 1
            patnum = 1;
        end

        %set the current pattern
        this_pat = train_inp(patnum,:);
        act = train_out(patnum,1);

        %calculate the current error for this pattern
        hval = (tanh(this_pat*weight_input_hidden))';
        pred = hval'*weight_hidden_output';
        error = pred - act;

        % adjust weight hidden - output
        delta_HO = error.*blr .*hval;
        weight_hidden_output = weight_hidden_output - delta_HO';

        % adjust the weights input - hidden
        delta_IH= alr.*error.*weight_hidden_output'.*(1-
(hval.^2))*this_pat;
        weight_input_hidden = weight_input_hidden - delta_IH';

    end
    % -- another epoch finished

    %plot overall network error at end of each epoch
    pred = weight_hidden_output*tanh(train_inp*weight_input_hidden)';
    error = pred' - train_out;
    err(iter) =  (sum(error.^2))^0.5;

    figure(1);
    plot(err)


    %reset weights if requested
    if reset
        weight_input_hidden = (randn(inputs,hidden_neurons) -
0.5)/10;
        weight_hidden_output = (randn(1,hidden_neurons) - 0.5)/10;
        fprintf('weights reaset after %d epochs\n',iter);
```

```
        reset = 0;
    end

    %stop if requested
    if earlystop
        fprintf('stopped at epoch: %d\n',iter);
        break
    end

    %stop if error is small
    if err(iter) < 0.001
        fprintf('converged at epoch: %d\n',iter);
        break
    end

end

   %-----FINISHED---------
   %display actual,predicted & error
   fprintf('state after %d epochs\n',iter);
   a = (train_out* sigma_out(:,1)) + mu_out(:,1);
   b = (pred'* sigma_out(:,1)) + mu_out(:,1);
   act_pred_err = [a b b-a]

%---------------------------------------------------------------------
---
% Classification stage added by me
% March 28, 2008

% The training stage is over, we have the weights for the network
n = 1;
samp = 1000;
set_1 = [randn(samp,n);1+randn(samp,n)];
set_2 = [randn(samp,n);1+randn(samp,n)];
% Add bias to the sets
set = [set_1 set_2 ones(samp*2,1)];
out = tanh(set*weight_input_hidden)*weight_hidden_output';
b = (out'* sigma_out(:,1)) + mu_out(:,1);
% Error rate
(samp-length(find(b(1:samp) < 0.2)))/samp
```

Support Vector Machine (SVM)

A SVM was implemented using MATLAB's bioinformatics toolbox. The data also belongs to MATLAB (fisheriris.mat). The data has useful information to recognize three different classes of fish. On the example found in [3], the classification consists on distinct one of the species from the other two.

On the first two lines the data is loaded, the next two lines divide the data into two sets, the training set and the test set. The classperf command evaluates the performance of the classifier based on the ground truth that has been passed to function prior the classification process.

The training stage (see Figure 6) is handled by *svmtrain*, which based on the ground truth finds the best separation surface between the two classes and

identifies the support vectors, i.e. the closest samples to the separation surface.

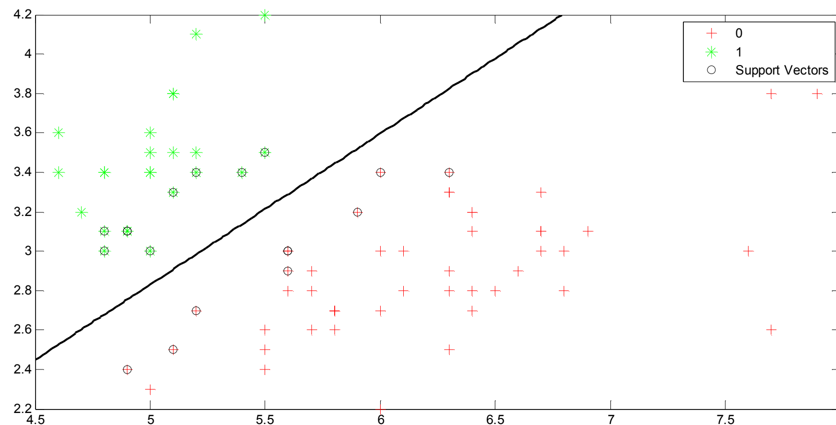

Figure 6. Support Vector Machine. Training stage. The classes contained on the training set are separated by the black surface. The samples enclosed in circles are the support vectors.

The information is kept on a structure, which in turn is passed to the function *svmclassify.* This function takes the test set and classifies the samples according to their distance to the separation surface. In Figure 7 we can see the results of the classification.

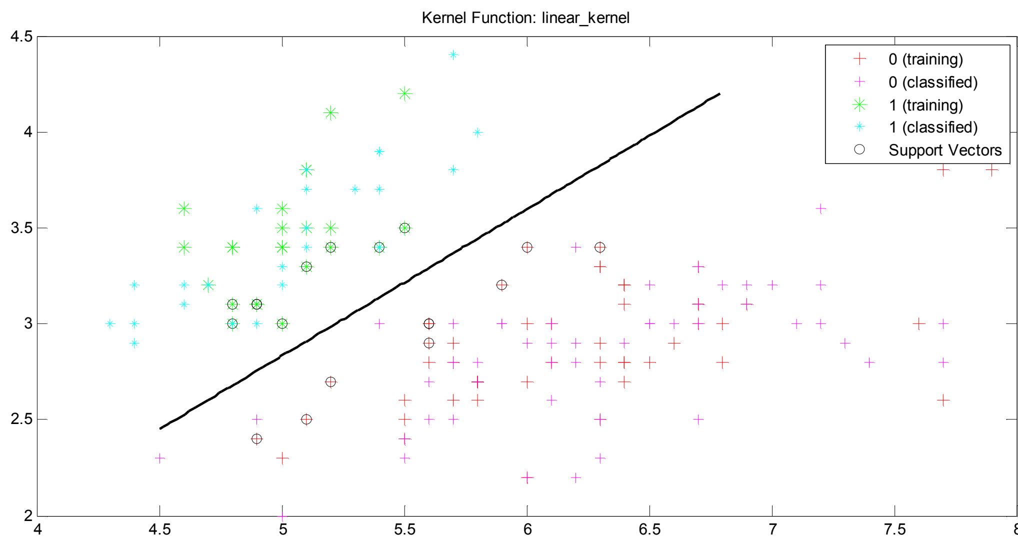

Figure 7. Classification of the samples in the test set based on the results from the training stage.

At the end, the function *classperf* is called again to evaluate the accuracy of the classification. The accuracy rate was 98.67% for a linear kernel. On the following plots (Figure 8), you will find the results of using different kernels.

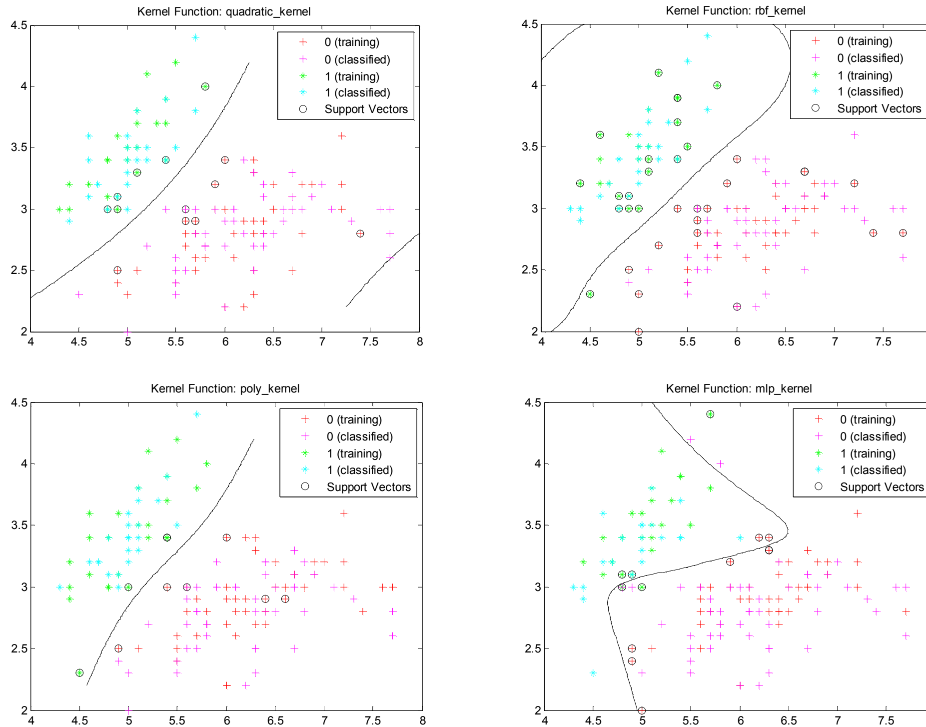

Figure 8. From top to bottom, left to right: quadratic kernel, rbf kernel, polynomial kernel, and mlp kernel

The accuracy rates for the different kernels were very close together thus, it seems that for the data set used, it is sufficient to implement a linear kernel. Although the accuracy rates were very similar, it is important to note that the corresponding to the multilayer perceptron kernel (mlp) was the lowest, 93%. Notice that the data used is well separated, thus the error rate is low and the separation surface is found easily.

MATLAB code

```
%%%% Bioinformatics toolbox MATLAB %%%%%%
%%%% Example:
%%%
http://www.mathworks.com/access/helpdesk/help/toolbox/bioinfo/index.h
t
%%% ml?/access/helpdesk/help/toolbox/bioinfo/ref/svmclassify.html

load fisheriris
data = [meas(:,1), meas(:,2)];
groups = ismember(species,'setosa');
[train, test] = crossvalind('holdOut',groups);
cp = classperf(groups);
svmStruct = svmtrain(data(train,:),groups(train),'showplot',true);
title(sprintf('Kernel Function: %s',...
              func2str(svmStruct.KernelFunction)),...
              'interpreter','none');
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
classperf(cp,classes,test);
cp.CorrectRate
```

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.

\

Parzen Window

The Parzen window chosen to estimate the density of the data is a rectangle with amplitude of 1 between -½ and ½.

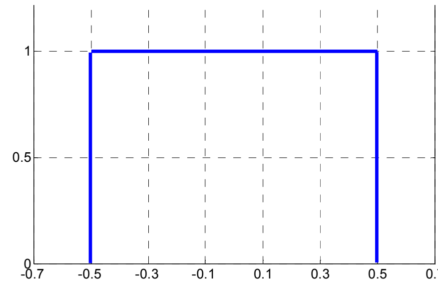

Figure 9. Parzen window. Rectangle with amplitude 1 between -½ and ½

The data is divided into two sets, the estimation set and the test set. The former set is estimates the density of the samples, and the latter is used in the classification stage.

The window is shifted in such a way that the sample of interest, x_o, is on its center. Then, we count the number of samples that fall inside the window centered on x_o and we divide by the total number of samples. We repeat the process with each sample of the estimation set and, at the end, we will have the estimate of the density of the set, as shown on equation 4.

$$p_i(\vec{x_0}) = \frac{k_i}{iV_i} = \frac{1}{iV_i}\sum_{l=1}^{i}\varphi\left(\frac{\vec{x_l}-\vec{x_0}}{h_i}\right)$$ **Equation 4**

Although the Parzen window approach provides an estimation of the density, such density is not used directly on the classification stage. Instead, the class to which x_o belongs is chosen based on a majority vote criterion.

$$P(w_j|\vec{x_0}) \geq P(w_i|\vec{x_0})$$ **Equation 5**

After some manipulation of equation 5, one can verify that the classification of x_o could be based on the number of samples of each class that fall inside the kernel. Therefore, x_o belongs to the class that has the greater number inside the kernel centered at x_o, as it is displayed on equation 6.

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

Equation 6

Two data sets on 1D, one for each class, were generated using Gaussian distributions. As mentioned before, each set was divided in two, for estimation and testing purposes. The variance was fixed to one for both sets and the difference between the means was at least one. Each set has 1000 samples.

An example of the density estimation that this method provides is displayed on Figure 10.

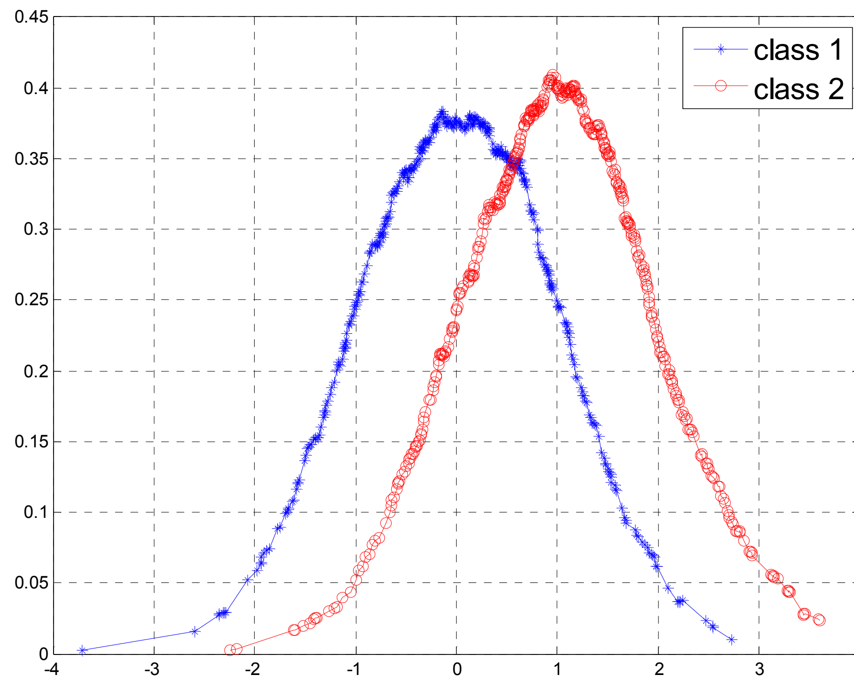

Figure 10. Density estimation. Class 1 is originally Gaussian distributed with variance 1 and mean 0. Class 2 is originally Gaussian distributed with variance 1 and mean 1.

The classification error depends on how close the classes are. For instance, when the means are 0 and 1, for class 1 and 2 respectively, the error is on the interval of 20% to 35%. But, when the classes are well separated (i.e. difference of the means greater than 4), the error is less than 5%.

MATLAB Code

```
%%% Parzen Windows
clear
n = 1;
samp = 1000;
h = 1;
set_1 = randn(samp,n);
set_2 = 3+randn(samp,n);
train = 3*samp/4;
drawn_1 = set_1(floor(samp.*rand(train,n)));
drawn_1 = sort(drawn_1);
```

```
drawn_2 = set_2(floor(samp.*rand(train,n)));
drawn_2 = sort(drawn_2);

% The window is a rectangle centered at zero from with 1 from -1/2 to
1/2
for i = 1:length(drawn_1)
    ind = find(set_1 <= drawn_1(i)+1/2 & set_1 >= drawn_1(i)-1/2);
    p_1(i) = length(ind)/length(drawn_2);

    clear ind
    ind = find(set_2 <= drawn_2(i)+1/2 & set_2 >= drawn_2(i)-1/2);
    p_2(i) = length(ind)/length(drawn_2);

    clear ind
end
plot(drawn_1,p_1,'*-')
hold on; plot(drawn_2,p_2,'ro-')

%%% Classification
for i = 1:samp
    ind = find(set_1 <= set_1(i)+1/2 & set_1 >= set_1(i)-1/2);
    t_1(i,1) = length(ind);
    clear ind
    ind = find(set_2 <= set_1(i)+1/2 & set_2 >= set_1(i)-1/2);
    t_1(i,2) = length(ind);
    clear ind
%     ind = find(set_1 <= set_2(i)+1/2 & set_1 >= set_2(i)-1/2);
%     t_2(i,1) = length(ind);
%     clear ind
%     ind = find(set_2 <= set_2(i)+1/2 & set_2 >= set_2(i)-1/2);
%     t_2(i,2) = length(ind);
%     clear ind
end
% error rate
1 - length(find(t_1(:,1) >= t_1(:,2)))/samp
```

K-Nearest Neighbor (kNN)

The choice of the kernel was the same as for Parzen windows (see Figure 9). The difference relies on the fact that such kernel doesn't have a fixed amplitude and width. Instead, those parameters are determined by the default number of neighbors *k* that we want to have around x_o.

As before, the data is Gaussian distributed with a unity variance and different means for each class. One part of the data is in charge of estimating the density of the set, and the other is used for testing.

The plots on Figure 11 present the density estimation based on different values of *k*. The greater the number of k-neighbors, the coarser the approximation of the density.

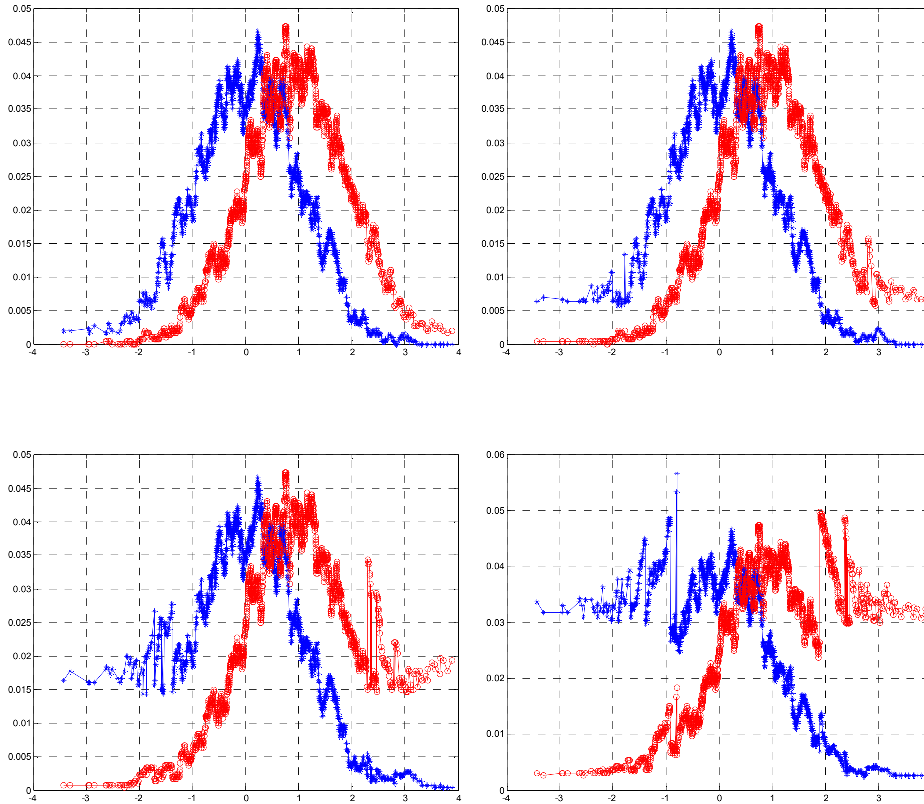

Figure 11. Class 1 in red and class 2 in blue. From top to bottom, left to right: density estimation with k = 5, k=21, k=51 and k=101 respectively

As it was mentioned on the case of Parzen Windows, the estimated density is not used directly on the classification. The decision is found by assigning to x_o the class that has the greatest number of samples among the k-neighbors.

The classification error was lower that the error presented using the Parzen windows technique. When the difference between the means is one, the error is between 2% and 11%, using k=5. As expected, the error decreases when the difference between the means is greater, being less that 1% when the means are 4 units apart.

Even thought the estimation of the density drawn from this approach looks noisier than the estimation given by the Parzen window, the classification results are much better than the results obtained through Parzen windows.

MATLAB Code

```
%%% k-nearest Neighbor
clear
n = 1;
samp = 1000;
k = 5;
h = 1;
randn('seed',1)
set_1 = randn(samp,n);     % Class 1
randn('seed',3)
```

```
set_2 = 1+randn(samp,n);  % Class 2
train = samp/2;
% The first half of the vector 'drawn' is composed by elements of
class 1,
% the second part is composed by class 2
drawn = [set_1(1:train)' set_2(1:train)'];
drawn = sort(drawn);
class = zeros(1,samp);
l = 0.1;
for i = 1:samp
    ind = 0;
    c = 1;
    while ind < k
        in_1 = find(set_1 <= drawn(i)+c*l/2 & set_1 >= drawn(i)-
c*l/2);
        ind_1 = length(in_1);
        in_2 = find(set_2 <= drawn(i)+c*l/2 & set_2 >= drawn(i)-
c*l/2);
        ind_2 = length(in_2);
        ind = ind_2 + ind_1;
        c = c+1;
    end
    p_1(i) = ind_1/length(drawn);
    p_2(i) = ind_2/length(drawn);
    if ind_2 > ind_1
        class(i) = 2;
    else
        class(i) = 1;
    end
    clear ind c
end
figure;
plot(drawn,p_1,'*-')
hold on; plot(drawn,p_2,'ro-')

% Error rate
abs(train - length(find(class == 2)))/train
```

Nearest Neighbor (NN)

In the NN approach, there is not need to use a kernel and the density is never estimated nor used. This method calculates the distance from x_o, the sample to be classified, to every sample on each class. Afterwards, x_o is assigned to the class that contains the sample with the minimum distance.

The data was Gaussian distributed, unity variance and different mean for each class. For the distance, the Euclidean metric was picked. The classification error depends on the data, especially on how spread it is.

The complexity of the algorithm depends on the number of samples in the set, the dimension size and the metric implemented to find the distance. In general, the error rate is low when the data is separable.

<u>MATLAB Code</u>

```
%%% NN
```

```
clear
n = 10;
samp = 1000;
set_1 = randn(samp,n);     % Class 1
set_2 = 1+randn(samp,n);   % Class 2
train = samp/2;
% The first half of the vector 'drawn' is composed by elements of
class 1,
% the second part is composed by class 2
drawn = [randn(train,n); (1+randn(train,n))];
class = zeros(1,samp);
for i = 1:samp
    dv = repmat(drawn(i,:)',1,samp);
    if min(norm(dv - set_1')) < min(norm(dv - set_2'))
        class(i) = 2;
    else
        class(i) = 1;
    end
end
```

CONCLUSIONS

In general, I would recommend, as a first approach, to implement the kNN method. It is a simple method that performed very well on the data that was used on this assignment. If it is important to achieve a good density estimate, the Parzen Windows method should be used instead; since the estimate provided by kNN was noisier than the Parzen Windows' one.

Most of the artificial data used on the assignment was nicely distributed, that didn't have outliers. The presence of outliers makes the NN and SVM techniques less reliable. SVM also depends on the kernel used to find the separation surface. The plots on Figure 8 show that it is not required to have several degrees of freedom on the separation surface to accomplish good classification results.

It was demonstrated numerically that Fisher's discriminant should be used as it is. When the denominator is removed, the information regarding the spread of the data is removed as well.

BIBLIOGRAPHY

[1] Shing-Tze Bow, *Pattern Recognition and Image Preprocessing*, Marcel Dekker, New York, 2002, pp 182-188.
[2] Phil Brierley, Neural Network training code for MATLAB, **www.philbrierley.com**
[3] Bioinformatics Toolbox Example, http://www.mathworks.com/access/helpdesk/help/toolbox/bioinfo/index.html?/access/helpdesk/help/toolbox/bioinfo/ref/svmclassify.html