Problem 1:

The student mistakenly believed that Fisher criterion $J(w)$ is equivalent to its numerator $N(w)$ in the determination of the optimum direction w:

i.e., he thought: $J(w) = \dfrac{w^T S_B w}{w^T S_W w}$ is equivalent to $N(w) = w^T S_B w$

That is not true. In this problem, in part (a) I will first implement an example to show a case when $J(w)$ is much better than $N(w)$. Then in part (b), I will theoretically explain why the student was confused and what is his missing point.

a) Implementation: Let's consider the case when two classes distribute as in Fig. 1 below.

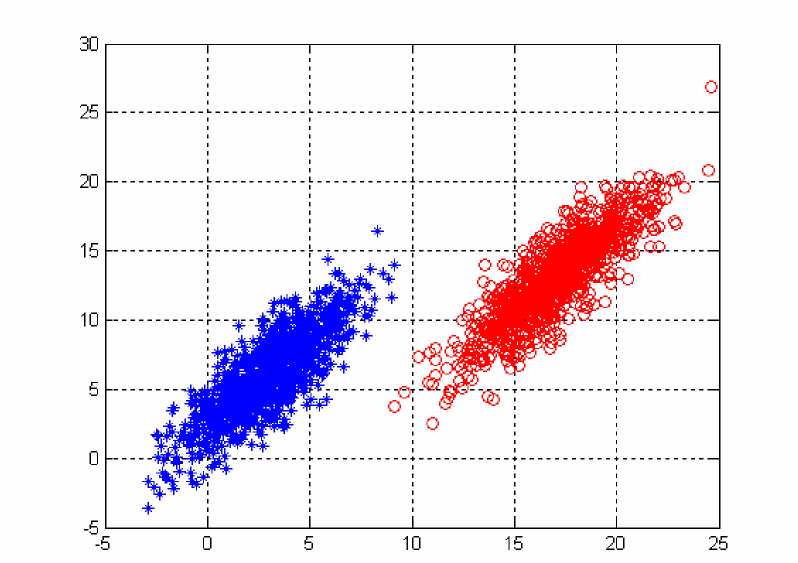

Figure 1: Two classes: class 1 (blue stars) and class 2 (red circles). Both normally distribute in 2-D space with the same covariance but different means.

```
Class 1: mean m1=[3; 6];
Class 2: mean m2=[17; 13];
Both two classes have the same covariance.
covar=[5    6;
       6    10];
```

Using $J(w)$ as the criterion, we find the optimum direction of w_F :

$$\overrightarrow{w_F} = \begin{bmatrix} \text{-6.7963} \\ 3.3072 \end{bmatrix}$$

Using $N(w)$, we find the optimum direction of w_N :

$$\overrightarrow{w_N} = \begin{bmatrix} \text{-14.1360} \\ \text{-7.1310} \end{bmatrix}$$

The projected data on w_F is shown in Fig. 2 and the projected data on w_N is shown in Fig.3.

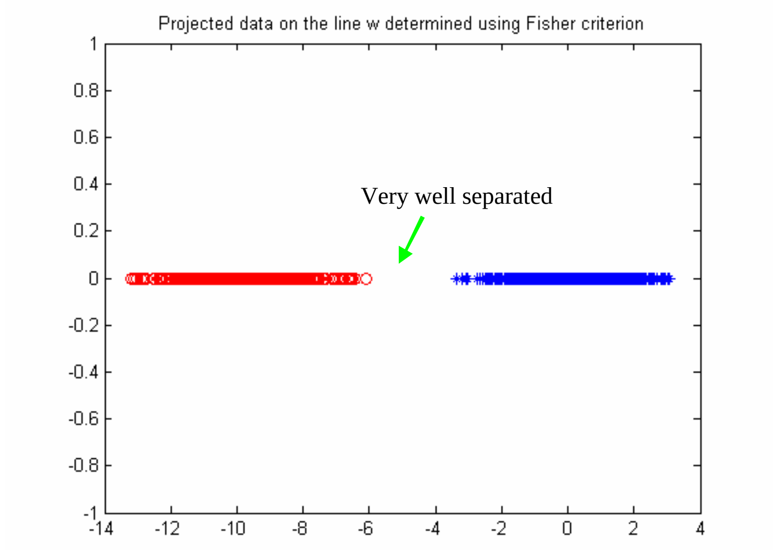

Figure 2: Projected data of the two classes on the line w_F determined by Fisher criterion.

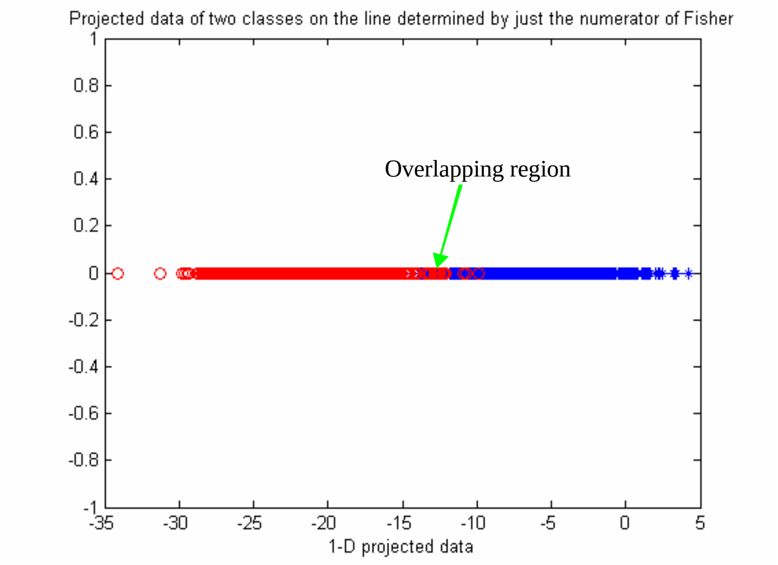

Figure 3: Projected data of the two classes on the line w_N determined by only the numerator of the Fisher criterion

Clearly, two projected classes are much better separated on w_F than that on w_N. This shows that Fisher criterion is much stronger than only its numerator in this case.

b) Explanation: Let's look at the meaning of the two criteria: Fisher $J(w)$ implies that two class means are well separated, measured relatively to the sum of the variances of the data of each class. $N(w)$, which the student suggested, only guarantees the first term:

two class means are well separated. Obviously $N(w)$ can't be as strong as Fisher $J(w)$ as shown in the example in part (a).

The reason why the student was confused is that since $J(w)$ is invariant with w (i.e. we can scale $w = a * w$, where a is any scalar number, and the direction of the optimum scaled $\vec{w}$ is still the same). Therefore, we can scale w such that $w^T S_W w$ in $J(w)$ equals 1 and still get the same direction solution as with Fisher. (i.e now $J(w)$ becomes $N(w)$).

The missing point of the student is that he forgot the condition $w^T S_W w = 1$. In fact, if he wants to use $N(w)$ and obtains an equivalent solution as Fisher, he will have to solve the following constrained optimization problem:

$$\text{maximize } N(w) = w^T S_B w$$
$$\text{Subject to: } w^T S_W w = 1$$

where the constrain condition is important in determination of the direction of $\vec{w}$. Actually, S_W affects not only $|w|$ but also the direction $\vec{w}$.

Problem 2:

In this problem, I will implement a Support Vector Machine (SVM) classifier in part (a) and an Artificial Neural Network (ANN) classifier in part (b). Since both classifiers are trained on the same training set and tested on the same testing set, we can compare the classification performance of two techniques. The comparison will be presented in part (c). I consider the classification performance of a classifier according to its misclassification rate and its classification speed. In (a) and (b) I will also describe how to select the parameters for each classifier and the impact of data scaling on their classification speed.

The data I generated in this problem is two dimensional so we can much easier to visualize its distribution, as well as the classification results of two techniques. The data set is plotted in Fig. 4, where class 1 is a cloud of blue starts and class 2 is that of red circles. As we can see in the figure, two classes overlap a lot on each other and which makes linear discrimination techniques fail to classify them. The total sample number in class 1 is 10,000 and that is the same in class 2. I pick out from each class 1500 samples to build a training set of 3000 samples. The rest in the overall data set is the testing set which is plotted in Fig. 5.

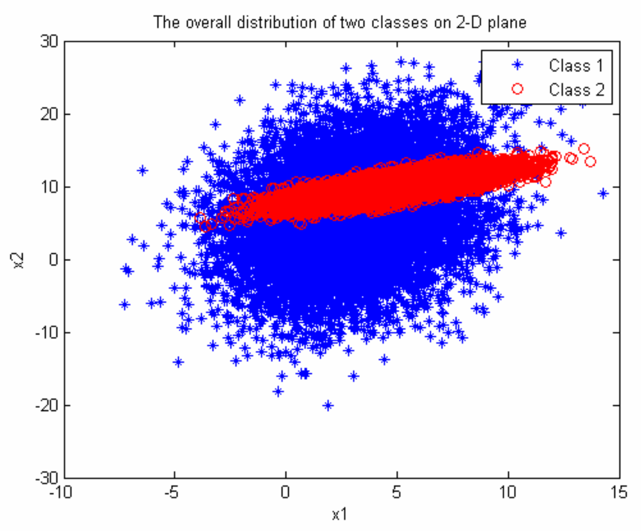

Figure 4: The distribution of whole data set on 2-D plane

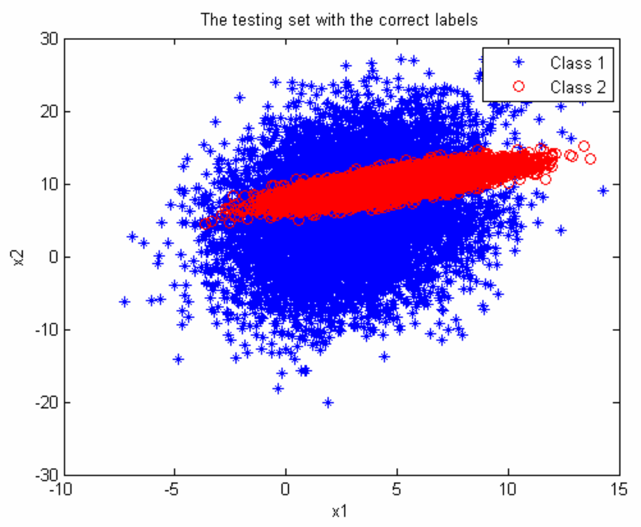

Figure 5: The distribution of the testing set with correct labels on 2-D plane.

a) ***Support Vector Machine:***

The core of the SVM algorithm is to solve the dual quadric programming (QP) optimization problem:

$$L_D = \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j y_i y_j K\left(x_i, x_j\right)$$

Subject to

$$0 \le a_i \le C$$

$$\sum_{i=1}^{n} a_i y_i = 0$$

where L_D: dual Lagrange form, a_i: Lagrange multipliers, n: number of training samples, $K\langle \bullet, \bullet \rangle$: kernel function, C: regularization parameters, y_i: for class 1 it equals -1 and for class 2 it equals 1.

We can use the Matlab function quadprog( ) to solve the problem. However, since the number of the training set is rather huge (i.e the size of the positive definite matrix in the QP is very large), it takes very long time for quadprog( ) to solve this QP. Furthermore, because Matlab has a limit in the size of QP which it can solve, in this problem I use OSU-SVM library [1] instead. The library OSU-SVM employs "sequential minimal optimization" (SMO) algorithm [2] to solve the QP. SMO breaks this large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. I won't describe more detail the SMO algorithm since it is beyond the scope of this homework. What I will concentrate on is what kernel function I choose and how I select the appropriate parameters for this chosen kernel function.

Firstly, I try to use a linear kernel to solve the problem. Clearly since two classes overlap (non-linearly separated), this classifier fails to classify them (Fig. 6).

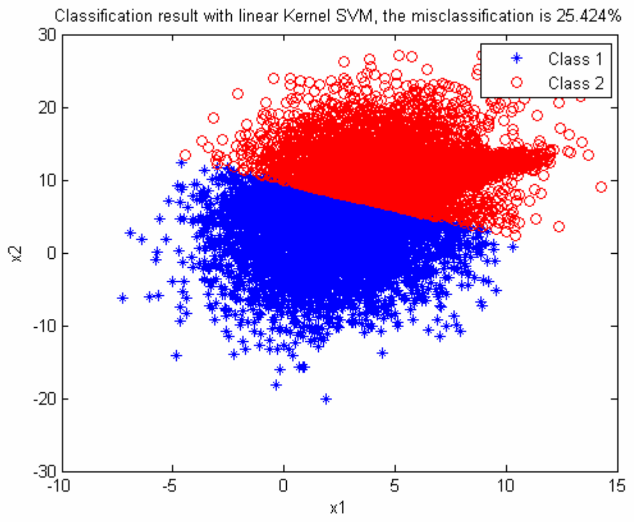

Figure 6: Classification result when using linear kernel.

The non-linearity in the relation between the class labels and class attributes suggests that we should use a non-linear kernel for the SVM algorithm. There are some possible choices for kernel function in OSU-SVM library: polynomial, radial basis function (RBF), and sigmoid. I decide to choose RBF for the following reasons:

- It maps samples to a higher dimension which hopefully can handle the overlap case.
- It can give equal performance to sigmoid with some certain parameters [3].
- It has fewer hyper-parameters than polynomial.
- Finally, it requires less computation than polynomial [4].

The RBF has the form:

$$K(xi,xj) = \exp\left(-g\,\|x_i - x_j\|\right);\ g > 0$$

There are two parameters which I have to determine for the SVM: g and C. A general way is to use cross-validation technique to determine these two parameters. However, in this problem, for the sake of simplicity, I use an alternative way: trial and error to search for the optimum g and C. Because these two parameters are independent, first I fix C and vary g with various values to find the local optimum value g which locally minimizes the misclassification rate. Then I fix that optimum g while varying C to find an local optimum C with which the misclassification rate is locally minimum. The search range of g is 1 to 100, of C is 1 to 20. Figure 7 shows the classification result using RBF kernel with the optimum g =3 and C=1.

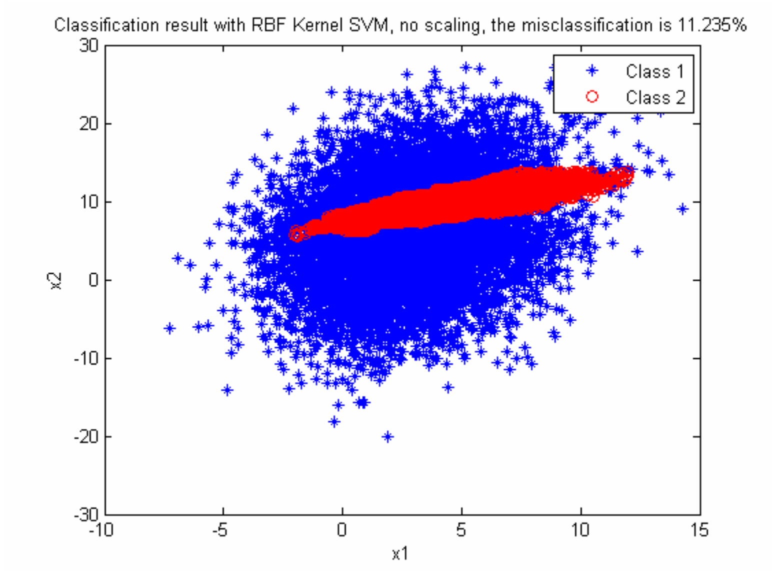

Figure 7: The classification result using RBF kernel function, no scaling data

As we can see in Fig. 7, the classifier does well on the classification performance. The misclassification is 11.23%.

Using Matlab function tic and toc before and after training and testing functions, respectively, I measure the time it takes for these tasks. This shows that the time for training is 3.62 seconds while for testing is 3.65 seconds.

To speed up the classifier, a technique is scaling the data to a smaller range. In this implementation, I scale the data to the range [0 1] and do again the classification including trial and error to find the optimum parameters. (Note that we need to scale both training and testing data sets). Figure 8 shows the classification result with scaled data. The misclassification rate is 11.29% (similar in Fig. 7) while the time for training and testing reduces significantly to 0.614 and 1.804 seconds, respectively. The optimum parameters for the RBF in this case are g =60 and C=5.

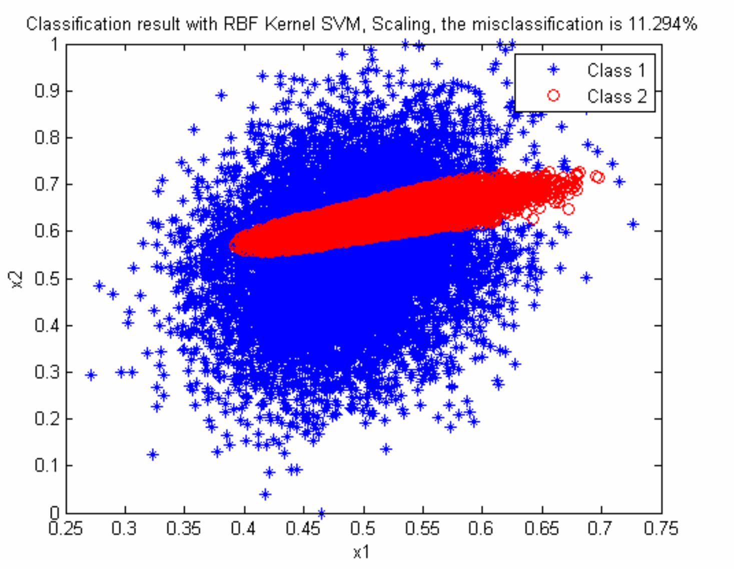

Figure 8: Classification result using RBF kernel with scaled data.

b) *Artificial Neural Network*

In this part, I will use a feed-forward multi-layer ANN to classify two classes. I use back-propagation training, adapting with a gradient-descent momentum approach, to train the ANN. Matlab's artificial neural network is a strong tool for me to solve this problem.

The first thing to determine is how many layers in the ANN, how many hidden units, and what transfer functions we need. Generally, an ANN with 1 hidden layer is sufficient to solve almost all problems. So I use 1 hidden layer ANN for this classification. In [5], the author suggests that the hidden unit number should be a number such that the total weight number in the network is roughly $\frac{n}{10}$ (n is number of training samples). Since we have two units in the input layer, two units in the output layers, and 3000 samples in the training set, the number of hidden layers can be computed accordingly as:

$N_H = \frac{3000}{10*4} - 1 = 74$, (the bias unit is also taken into account). 74 is a too big number and it costs a lot of computation and time when the classification is implemented in Matlab. I again use "trial and error" technique to look for an appropriate N_H. I search it in the range from 5 to 30 and find 20 as the "optimum" number (a compromise between misclassification rate and computation time). Thus, the structure of the ANN is (2-20-2). I also choose logsig( ) as the transfer function for the hidden layer and purelin( ) for output layer. (It turned out that the replacement of tansig( ) for whichever above doesn't affect very much the classification rate in this problem).

I also vectorize the class category as: class 1 corresponding with output $\begin{bmatrix}1\\0\end{bmatrix}$ and class 2 corresponding with output $\begin{bmatrix}0\\1\end{bmatrix}$.

Figure 9 shows the classification result using the built ANN. The misclassification is similar to SVM with RBF in part (a).

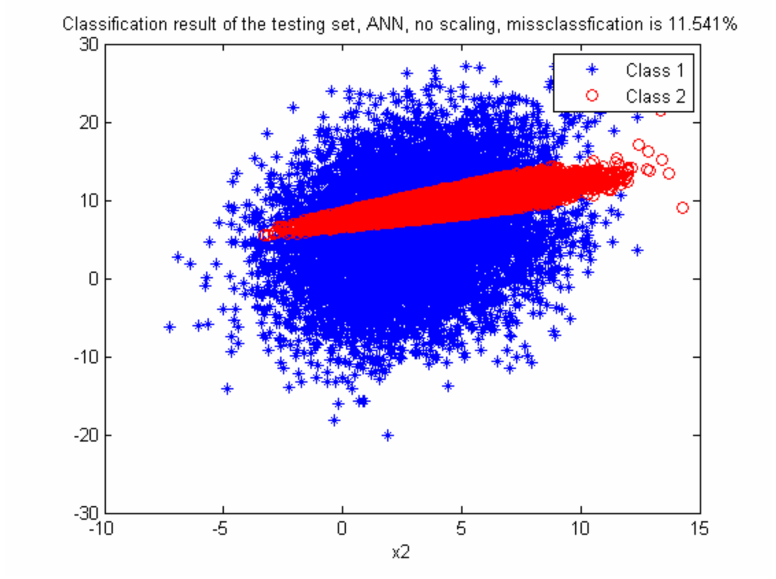

Figure 9: Classification result using ANN

c) Comparison between two techniques

As we see in Fig. 7 and Fig. 9, both techniques give very similar results with the training set of 3000 samples. (Actually, SVM is a little bit better).
To compare further the performance of two techniques, I reduce the training set to 100 samples while keeping the same testing set of 17000 samples. The following figures show the classification results of two techniques in the case of the small training set.

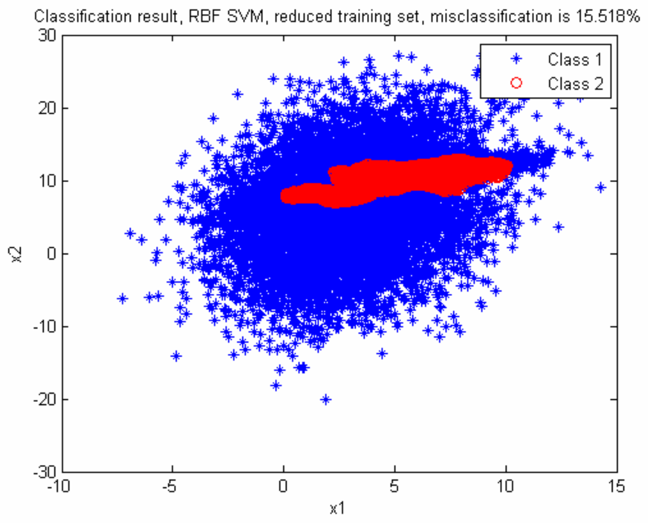

Figure 10: Classification using RBF SVM, training set of 100 samples

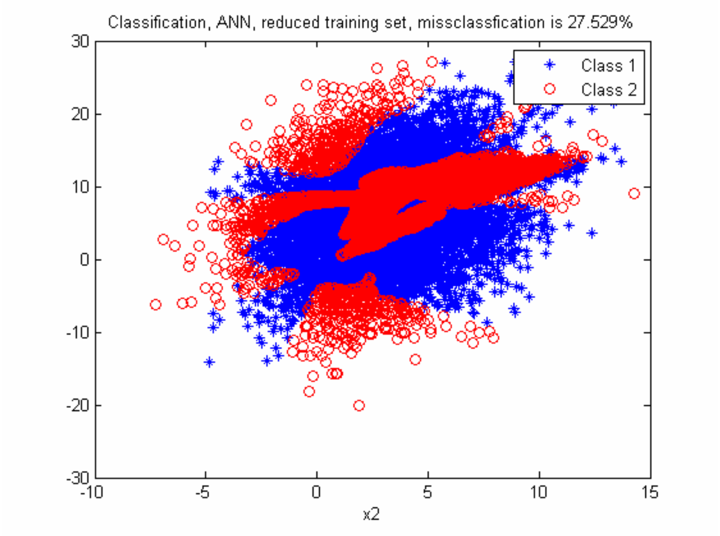

Figure 11: Classification using ANN, training set of 100 samples.

Figure 10 and Fig. 11 show that RBF SVM performs much better than ANN in this case.

The classification speed is also an important factor which we should consider. While we can use tic and toc functions to measure the time the system consumes for training and testing in SVM algorithm, unfortunately we can't do so for ANN in MATLAB. When training a network, we can't disable a pop-up window which updates the training progress. The time the system spends on generating and updating this window is considerable. We can't measure this time amount until we access the source codes of the toolbox. This access obviously is not recommended.

Problem 3:

In this problem, I will implement Parzel window in part (a), K-nearest neighbor in part (b), and nearest neighbor in part (c), and finally will compare their performance in part (d). For all techniques I use the reduced training set of 100 samples generated in problem 2. The testing set is the same as that in problem 2. Both training set and testing set are scaled to the range of [0 1] to reduce the complexity of the computation and thus increase the classification speed.

a) Parzel window:

The type and the smoothing parameter of the window are two things I need consider firstly in this problem. In practice, the most widely used window is the normal form. In this homework, I choose a circle as a Parzel window and determine the smoothing parameter according to the suggestion in [6]:

$$h = s * n^{\left(-\frac{1}{6}\right)}$$

where $s^2 = \frac{1}{2}\sum_{i=1}^{2} s_{ii}$; s_{ii} are diagonal elements of the sample covariance matrix of the training set; n: the number of samples in the training set. The computed h is 0.0382.

The algorithm is based on the larger votes for a class in a circle with h=0.0382, centered at a new point, to assign this point to that class.

Figure 12 shows the classification using the Parzel window designed above. Note that both training and testing sets are scaled to [0 1]. With the reduced training set (100 samples), the classification is quite good.

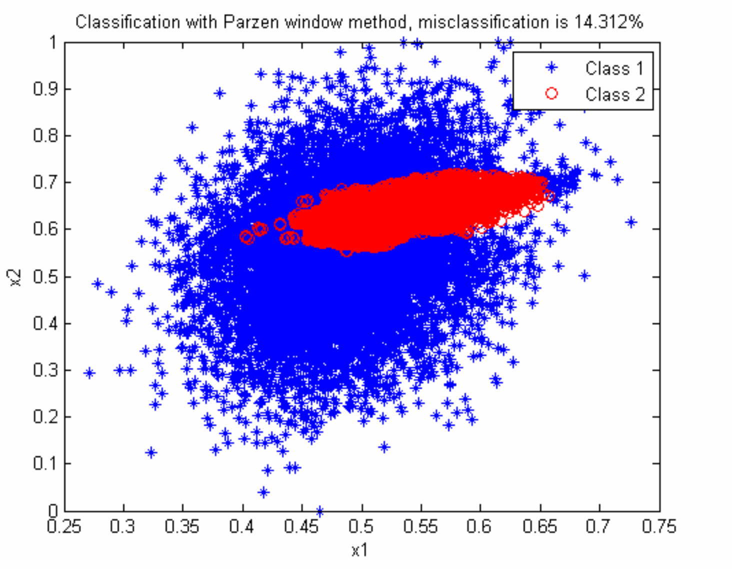

Figure 12: Classification result using Parzen window (a circle with radius of 0.0382), training set of 100 samples.

b) K-nearest neighbor technique

The algorithm is based on the larger votes for a class in a circle centered at a new added point that covers K training samples. The reader may want to direct to my MATLAB codes to see about the algorithm.

For K nearest neighbor technique, similarly to part (a), the first thing is to determine the number K. According to the suggestion in [7], K is determined as follows:

$$K = round\left(n^{\frac{3}{8}}\right) = round\left(100^{\frac{3}{8}}\right) = 6$$

where n=100 is number of the training samples.

Similarly to part (a) the algorithm assigns a new point to a certain class based on the votes for that class in a window centered at the new point and containing 6 training samples.

Figure 13 shows the classification result using 6-nearest neighbor technique. The misclassification rate is slightly larger than that in part (a).

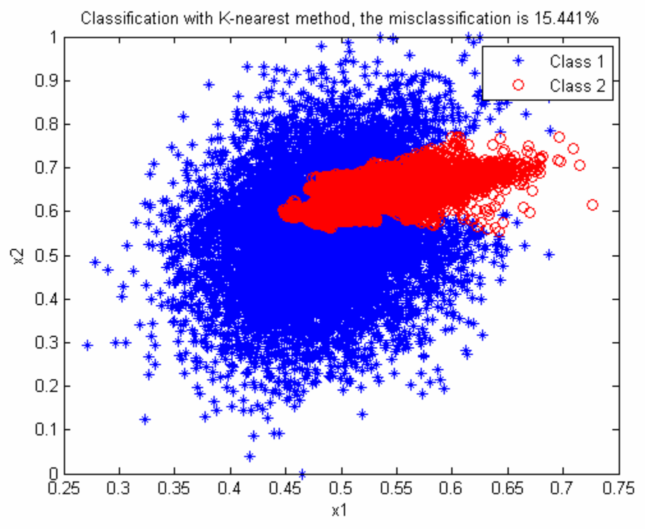

Figure 13: Classification with 6-nearest neighbor technique, training set of 100 samples.

c) Nearest neighbor technique

This actually is a specific case of K-nearest neighbor technique where K=1. The classification is shown in Fig. 14 below.

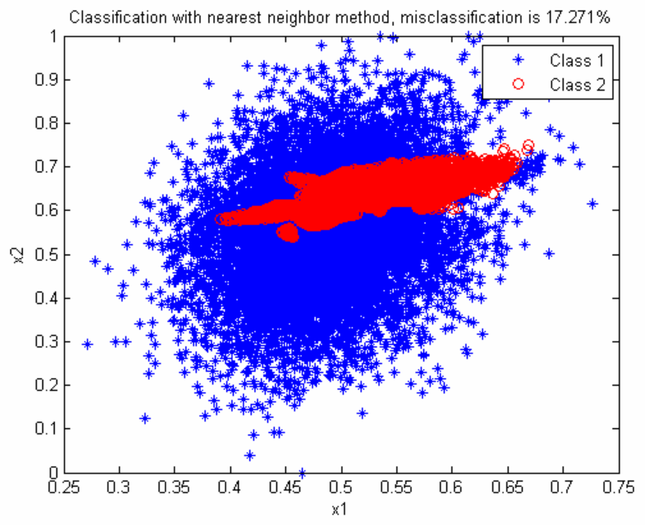

Figure 14: Classification using nearest neighbor technique, training set of 100 samples

d) Comparison of three techniques:

We can see the K-nearest neighbor and Parzel window provide very similar result. And compared to problem 2, with the small training set (100 samples) these two techniques perform even better than ANN.

The nearest neighbor gives the poorer result compared to the two above. However it requires less computation costs.

References:

[1] OSU-SVM library for MATLAB, http://svm.sourceforge.net/license.shtml

[2] J. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," Microsoft Research Technical Report MSR-TR-98-14, 1998.

[3] Lin, H-T and C.-J Lin, "A study of sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods," Technical report, Department of Computer Science, National Taiwan University, 2003.

[4] Chih Wei Hsu, Chih Chung Chang, and Chih Jen Lin, "A practical guide to support vector classification," Department of Computer Science, National Taiwan University, 2008.

[5] Duda, Hart, and Stork, "Pattern classification", 2nd edition, John Wiley & Sons, LTD, 2001.

[6] Silverman, "Density estimation for statistics and data analysis," Chapman &Hall, London, 1986.

[7] Enas, G. G and Choi, S.C, "Choice of the smoothing parameter and efficiency of k-nearest neighbor classification," Computer and Mathematics Application, 12A(2):235-244.

MATLAB codes:

```
%ECE 662, HW2 Prob 1, the demonstration of the strength of Fisher vs
its
%numerator
clear all;
clc;
hold off;
close all;
m1=[3; 6];
covar1=[5    6;
        6    10];
omega1= mvnrnd(m1,covar1,1000);
omega1=omega1';
m2=[17; 13];
omega2= mvnrnd(m2,covar1,1000);
omega2=omega2';

%Plot the data
figure(1);
plot(omega1(1,:),omega1(2,:),'*b',omega2(1,:),omega2(2,:),'or');
grid on;

[m_turde1 sigma1]=mle_1(omega1);
[m_turde2 sigma2]=mle_1(omega2);

S_w=(1/(2000-2))*(1000*sigma1+1000*sigma2);

%Fisher
w_F=inv(S_w)*(m_turde1-m_turde2);
proj_F1=(w_F'*omega1)./norm(w_F); % projection
proj_F2=(w_F'*omega2)./norm(w_F);
figure(2);
plot(proj_F1,zeros(1,1000),'*b',proj_F2,zeros(1,1000),'or'); % Project
the data on the line
ylim([-1,2]);

%no_Fisher
w_0=m_turde1-m_turde2;

proj_01=(w_0'*omega1)./norm(w_0);
proj_02=(w_0'*omega2)./norm(w_0);
figure(3);
plot(proj_01,zeros(1,1000),'*b',proj_02,zeros(1,1000),'or');  % Project
the data on the line
ylim([-1,2]);
```

```
 % ECE 662 HW2
% Generate data for problem 2 and problem 3
clear all;
clc;
```

```
hold off;
% the case of 2-D vector
m1=[3; 6];
covar1=[1.4370    0.8615
        0.8615    8.3650];
covar1=covar1*5;

m2=[5;10];
covar2=[5.1492    2.3820
        2.3820    1.8606 ];

omega1= mvnrnd(m1,covar1,10000);
omega2= mvnrnd(m2,covar2,10000);
train=[omega1(1:1500,:)' omega2(1:1500,:)'] ;
label=[zeros(1,1500) ones(1,1500)];
test_set=[omega1(1501:end,:)' omega2(1501:end,:)'];
label2=[zeros(1,8500) ones(1,8500)];

clear m1 m2 covar1 covar2;
save data
```

```
%Prob 2a1
%This shows SVM using linear kernel
clear all;
clc;
hold off;
%load data
load data;


%build linear classifier

[AlphaY, SVs, Bias, Parameters, nSV, nLabel] = LinearSVC(train_set,
label); % This function is from OSU-SVM library

figure(1);
plot(test_set(1,1:8500),test_set(2,1:8500),'*b',test_set(1,8501:end),te
st_set(2,8501:end),'or');
title('The testing set with the correct labels');
Xlabel('x1');
Ylabel('x2');
legend('Class 1','Class 2');

[ClassRate, DecisionValue, Ns, ConfMatrix, PreLabels]=
SVMTest(test_set, label2, AlphaY, SVs, Bias,Parameters, nSV, nLabel);

%Compute misclassification rate
omega_1=[];
omega_2=[];
err1=0;
err2=0;
for i=1:8500
    if(PreLabels(i))
        err1=err1+1;
        omega_2=[omega_2 test_set(:,i)];
    else
        omega_1=[omega_1 test_set(:,i)];

    end
    if(PreLabels(i+8500))
        omega_2=[omega_2 test_set(:,i+8500)];
    else
        err2=err2+1;
        omega_1=[omega_1 test_set(:,i+8500)];


    end
end
err=((err1+err2)/17000)*100;
figure(2);
plot(omega_1(1,:), omega_1(2,:),'b*',omega_2(1,:), omega_2(2,:),'ro');
title(['Classification result with linear Kernel SVM, the
misclassification is ' num2str(err,'%2.3f') '%']);
Xlabel('x1');
Ylabel('x2');
legend('Class 1','Class 2');
```

```
%Prob 2a2
%ECE 662
%SVM using RBF
clear all;
clc;
hold off;
load data

%scaling data to [0 1]
overal_data=scale_data([train_set test_set],0,1);
train_set=overal_data(:,1:3000);
test_set=overal_data(:,3001:end);

% train_set=[train_set(:,1:50) train_set(:,2951:end)]; <==this code
will be
% used when training a reduced training set
% label=[label(:,1:50) label(:,2951:end)];


% The best optimum parameters found by trial and error method
%scalling Gamma=50 C=6 are the best paras
%no scalling Gamma=3 C=1 are the best paras

tic;
[AlphaY, SVs, Bias, Parameters, nSV, nLabel] = RbfSVC(train_set,
label,50,6);
toc
tic;
[ClassRate, DecisionValue, Ns, ConfMatrix, PreLabels]=
SVMTest(test_set, label2, AlphaY, SVs, Bias,Parameters, nSV, nLabel);
toc
omega_1=[];
```

```
omega_2=[];
err1=0;
err2=0;
for i=1:8500
    if(PreLabels(i))
        err1=err1+1;
        omega_2=[omega_2 test_set(:,i)];
    else
        omega_1=[omega_1 test_set(:,i)];

    end
    if(PreLabels(i+8500))
       omega_2=[omega_2 test_set(:,i+8500)];
    else
       err2=err2+1;
       omega_1=[omega_1 test_set(:,i+8500)];
    end
end

err=((err1+err2)/17000)*100;
figure(3);
plot(omega_1(1,:), omega_1(2,:),'b*',omega_2(1,:), omega_2(2,:),'ro');
% title(['Classification result with RBF Kernel SVM, no scaling, the 
misclassification is ' num2str(err,'%2.3f') '%']);
title(['Classification result with RBF Kernel SVM, scaling, the 
misclassification is ' num2str(err,'%2.3f') '%']);
Xlabel('x1');
Ylabel('x2');
legend('Class 1','Class 2');
```

```
%Prob2b
%Using Neural network for classification
% Large training set 3000 samples
clear all;
clc;
hold off;
close all;
load data

% will be used in the case of reducing training set
% train_set=[train_set(:,1:50) train_set(:,2951:end)];
% target=[ones(1,50) zeros(1,50);zeros(1,50) ones(1,50)];

PR=minmax(overall_data);
% vectorize the ouputs
target=[ones(1,1500) zeros(1,1500);zeros(1,1500) ones(1,1500)];
%determine the number of hidden units:
net = newff(PR,[20 2],{'tansig' 'purelin'});
% training
net.trainParam.epochs = 500;
net.trainParam.goal = 0.01;

tic;
net = train(net,train_set,target);
toc

tic;
```

```
Y = sim(net,test_set);
toc
omega_1=[];
omega_2=[];
err1=0;
err2=0;
for i=1:8500
    if (Y(1,i)>=Y(2,i))
        omega_1=[omega_1 test_set(:,i)];
    else
        omega_2=[omega_2 test_set(:,i)];
        err1=err1+1;
    end

    if (Y(1,i+8500)<=Y(2,i+8500))
        omega_2=[omega_2 test_set(:,i+8500)];
    else
        omega_1=[omega_1 test_set(:,i+8500)];
        err2=err2+1;
    end
end

err=(err1+err2)/170;
figure (2);

plot(omega_1(1,:), omega_1(2,:),'b*',omega_2(1,:), omega_2(2,:),'ro');
title(['Classification result of the testing set, without scaling
missclassfication is ' num2str(err,'%2.3f') '%']);
% title(['Classification result of the testing set, scaling,
missclassfication is ' num2str(err,'%2.3f') '%']);
xlabel('x1');
xlabel('x2');
legend('Class 1','Class 2');
```

```
%This function is to scale data to a smaller range
function scaled=scale_data(data,min_lev, max_lev) %work for 2_D data

min_data=min(min(data));
max_data=max(max(data));
dist1=max_data-min_data;
dist2=max_lev-min_lev;

scaled=(((data-min_data).*dist2)./dist1)+min_lev;
```

```
%This is to compute the smoothness for Parzel window using Silverman
(1986); this works well
%for normal-distribution
function h=h_compute(data)

mean_data=mean(data,2);
mean_data=repmat(mean_data,1,100);
A=data-mean_data;
B=A*A';
B=B./size(data,2);
```

```
sigma=mean([B(1,1) B(2,2)]);
h=sqrt(sigma)*(size(data,2).^(-1/6));
```

```
%ECE 662
%ECE K-nearest
%K nearest neighbor (part b, probelm 3, HW2)
clear all;
clc;
hold off;
close all;
load data

%scaling data to increase the speed
train_set=[train_set(:,1:50) train_set(:,2951:end)];
label=[zeros(1,50) ones(1,50)];
overall_data=scale_data([train_set test_set],0,1);
train_set=overall_data(:,1:100);
test_set=overall_data(:,(end-17000):end);



K=round((100)^(3/8)); % choose the number K according to Enas and
Choice (1986)

omega_1=[];
omega_2=[];
err1=0;
err2=0;
for i=1:17000
     arr_tmp=[];
     for j=1:100
        arr_tmp=[arr_tmp norm(test_set(:,i)-train_set(:,j))];
     end
     [B idx]=sort(arr_tmp); %idx is the index array sorted by the
values of the arr_tmp
     vote1=0; %vote for class 1;
     vote2=0; %vote for class 2;
     for k=1:K
             if (~label(idx(k)))
                vote1=vote1+1;

             else
                vote2=vote2+1;
             end
     end

     if(i<=8500)
         if (vote1>=vote2) % correct classification
             omega_1=[omega_1 test_set(:,i)];
         else %wrong classification
             omega_2=[omega_2 test_set(:,i)];
             err1=err1+1;
         end
     else
        if (vote1 < vote2) % correct classification
             omega_2=[omega_2 test_set(:,i)];
         else %wrong classification
             omega_1=[omega_1 test_set(:,i)];
             err2=err2+1;
         end
     end
end

err=(err1+err2)/170;
plot(omega_1(1,:), omega_1(2,:),'b*',omega_2(1,:), omega_2(2,:),'ro');
title(['Classification with K-nearest method, the misclassification is
' num2str(err,'%2.3f') '%']);
Xlabel('x1');
Ylabel('x2');
legend('Class 1','Class 2');
```

```
%ECE 662, part c, problem 3, HW 2
%Nearest Neighbor Technique
clear all;
clc;
hold off;
close all;
load data

%scaling data to increase the speed
train_set=[train_set(:,1:50) train_set(:,2951:end)];
label=[zeros(1,50) ones(1,50)];
overall_data=scale_data([train_set test_set],0,1);
train_set=overall_data(:,1:100);
test_set=overall_data(:,(end-17000):end);



K=1;% nearest neighborhood

omega_1=[];
omega_2=[];
err1=0;
err2=0;
for i=1:17000
     arr_tmp=[];
     for j=1:100
        arr_tmp=[arr_tmp norm(test_set(:,i)-train_set(:,j))];
     end
     [B idx]=sort(arr_tmp); %idx is the index array sorted by the
values of the arr_tmp
     vote1=0; %vote for class 1;
     vote2=0; %vote for class 2;
     for k=1:K
             if (~label(idx(k)))
                vote1=vote1+1;

             else
                vote2=vote2+1;
             end
     end

     if(i<=8500)
         if (vote1>=vote2) % correct classification
```

```
             omega_1=[omega_1 test_set(:,i)];
         else %wrong classification
             omega_2=[omega_2 test_set(:,i)];
             err1=err1+1;
         end
     else
        if (vote1 < vote2) % correct classification
             omega_2=[omega_2 test_set(:,i)];
         else %wrong classification
             omega_1=[omega_1 test_set(:,i)];
             err2=err2+1;
         end
     end
end

err=(err1+err2)/170;
plot(omega_1(1,:), omega_1(2,:),'b*',omega_2(1,:), omega_2(2,:),'ro');
title(['Classification with nearest neighbor method, misclassification
is ' num2str(err,'%2.3f') '%']);
Xlabel('x1');
Ylabel('x2');
legend('Class 1','Class 2');
```

```
%Part a Probelm 3 HW 2
% Parzel window
clear all;
clc;
hold off;
close all;
load data

%scaling data to increase the speed
train_set=[train_set(:,1:50) train_set(:,2951:end)];
label=[zeros(1,50) ones(1,50)];
overall_data=scale_data([train_set test_set],0,1);
train_set=overall_data(:,1:100);
test_set=overall_data(:,(end-17000):end);



h=h_compute(train_set); % using Silverman (1986) to compute the
smoothness para of the window
% h=h/2;
omega_1=[];
omega_2=[];
err1=0;
err2=0;
for i=1:17000
     arr_tmp=[];
     for j=1:100
        arr_tmp=[arr_tmp norm(test_set(:,i)-train_set(:,j))];
     end
     [B idx]=sort(arr_tmp); %idx is the index array sorted by the
values of the arr_tmp
     K=find(B>h,1,'first')-1; % Using a circle with radian=h as the
Perzen windonw
     if (isempty(K) || K==0) % no training sample in the window, assign
it to class 1
```

```
        omega_1=[omega_1 test_set(:,i)];
        if (i>8500)
            err2=err2+1;
        end
     else
         idx=idx(1:K);
         vote1=0; %vote for class 1;
         vote2=0; %vote for class 2;
         for k=1:K
                 if (~label(idx(k)))
                    vote1=vote1+1;
                 else
                    vote2=vote2+1;
                 end
         end

         if(i<=8500)
             if (vote1>=vote2) % correct classification
                 omega_1=[omega_1 test_set(:,i)];
             else %wrong classification
                 omega_2=[omega_2 test_set(:,i)];
                 err1=err1+1;
             end
         else
            if (vote1 < vote2) % correct classification
                 omega_2=[omega_2 test_set(:,i)];
             else %wrong classification
                 omega_1=[omega_1 test_set(:,i)];
                 err2=err2+1;
             end
         end
     end
end

err=(err1+err2)/170;
plot(omega_1(1,:), omega_1(2,:),'b*',omega_2(1,:), omega_2(2,:),'ro');
title(['Classification with Parzen window method, misclassification is
' num2str(err,'%2.3f') '%']);
Xlabel('x1');
Ylabel('x2');
legend('Class 1','Class 2');
```