1) Parametric Method separation hyperplane between two classes obtained by $w_0 = S_w^{-1}(m_1 - m_2)$ compared to using $S_w = I$:

In order to investigate the difference between using the true within class scatter matrix and using $S_w = I$, we created two classes of 2-feature data with a Normal distribution using the "mvnrnd" Matlab function, with $P(w_1) = P(w_2) = ½$, $\mu_1 = [5\ 5]^T$, and $\mu_2 = [3\ 3]^T$. We set Σ_2 to a constant matrix of [1 0; 0 1], and varied Σ_1 by varying the values on the main diagonal of the covariance matrix independently from 0.1 to 2, in steps of 0.1. Then, in each of these twenty configurations, 300 training points were created and used to generate w_0 using both the formula that maximizes the cost function J(w) and the formula where $S_w = I$. 5,000 samples were then created using each of the twenty covariance configurations, and error performance was calculated for classification based on an optimal threshold used upon the projection of each sample point onto each w_0. The ratio of the error performance for the two cases with each of the 20 covariance matrices is shown in figure 1.1, below.

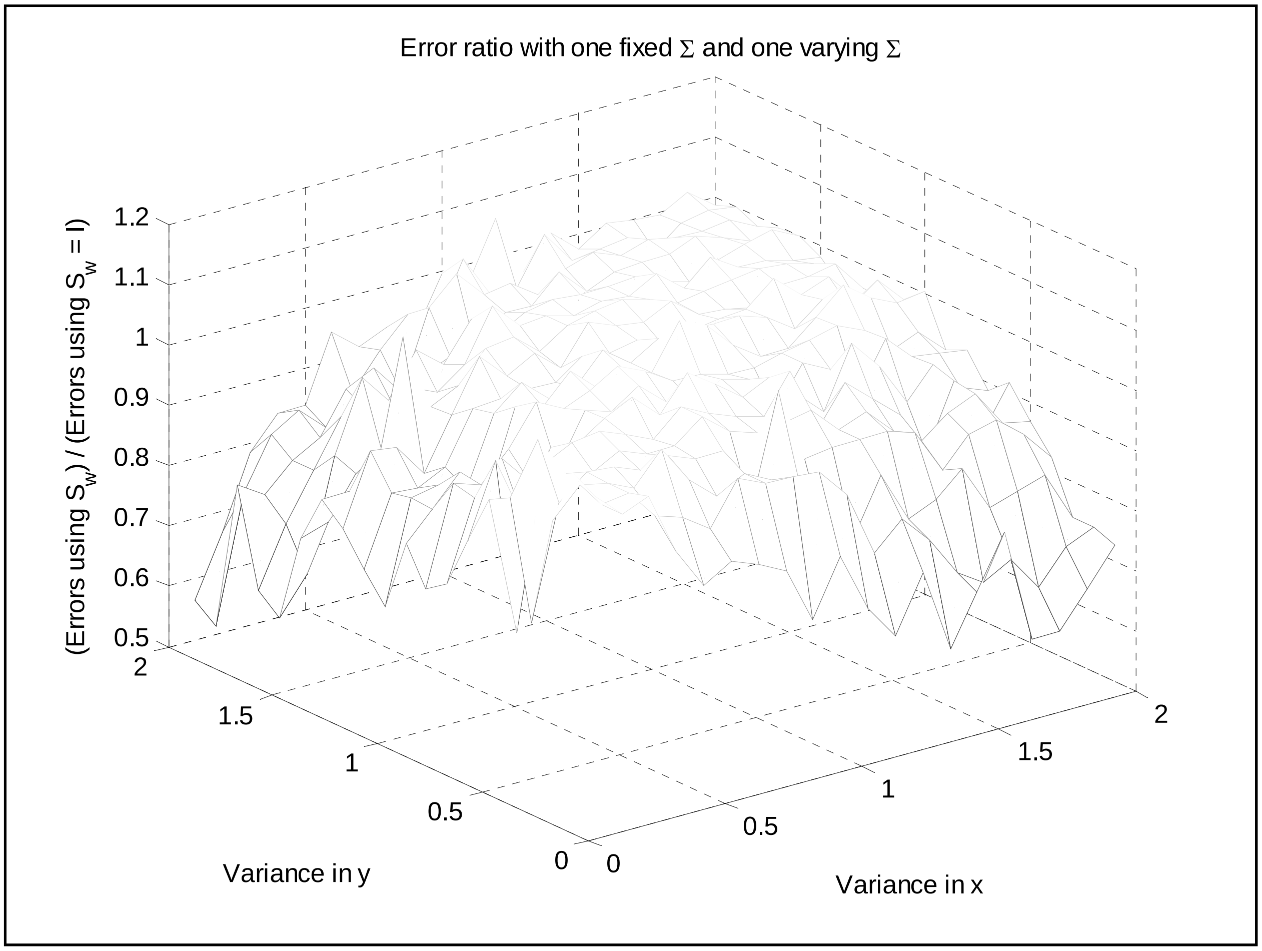


Figure 1.1

From the above figure, we see that the within class scatter matrix does, in fact, affect the error performance of this classifier. From the equation for the within class scatter matrix, we can see that as the variances of each feature vector become equal, the values in the main diagonal of S_w will become equal. With no covariance values, and equal variances, the within class scatter matrix will tend towards a scaled version of the identity matrix. As is seen in Figure 1.1, when the variances in x and y are nearly equal, the error performance of the two classifications are approximately equal, therefore the ratio between them is approximately one. However, the more varied the variances become, the less equal are the ratios. Thus, we have shown that replacing S_w with I is only valid if both covariance matrices for the Normal distributions are scaled versions of the identity matrix. Otherwise, error performance will decrease as the variances of the features vary.

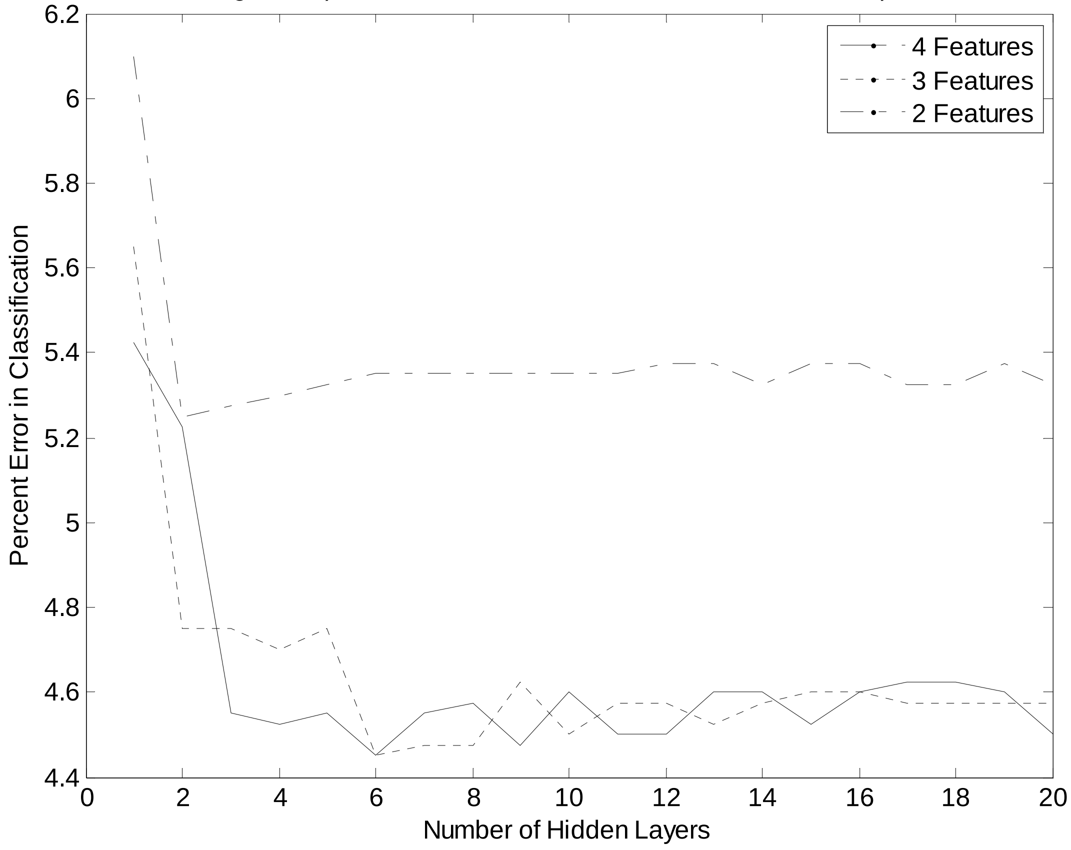
2) Comparison of Neural Network Approach of classifier design to Support Vector Machine Approach of classifier design:

Data used for these experiments is the “svmguide1” dataset available from the following website: http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. These data consist of two classes with four feature vectors. 3,089 training samples and 4,000 test samples were provided.

Neural Network Approach:

In order to first investigate the neural network approach to designing classifiers, the “nc_multiclass” neural network toolbox was obtained from http://isp.imm.dtu.dk/toolbox/ann/. This toolbox lets a user specify the number of hidden units to use in the neural network. It then uses the provided training data and class specifications to train the network, using the BFGS optimization algorithm. It implements hyperbolic tangents for the hidden layers, and a softmax function for the output layer.

Experimentation with the neural network classifier consisted of varying the number of hidden layers and feature vectors. For each training attempt tested, the same set of 300 training samples, 150 from each class, were used. Results were obtained using four, three, and two feature vectors. The number of hidden layers was varied from 1 to 20, and all 4,000 sample points were used to calculate the error performance that is plotted in figure 2.1, below.



From the above figure, as expected, using more feature vectors does, in fact, increase the overall error performance of the neural network. The performance error rates were about 5.3%, 4.6%, and 4.5% when using two features, three features, and four features, respectively. Also as expected, using more hidden layers does not mean that the performance of the network will improve. In all three trials, by using just a few hidden layers, and approximate floor of the error performance is reached, with only slight variations occurring with more hidden layers. There also is not a trend between the number of feature vectors and the number of hidden layers required. This is expected, because any two hidden layers could be combined into one more complicated hidden layer. Thus, theoretically, only one hidden layer would be required if adequate optimization were performed on the neural network.

Support Vector Machine Approach:

Support vector machines were investigated using the “svmtrain” and “svmclassify” commands from Matlab’s Bioinformatics Toolbox. Training was performed with a linear kernel classifier.

Again, 300 training samples were used from “svmguide1”, 150 from each class. 4,000 samples were used to evaluate the error performance of the SVM with each of 1-4 feature vectors being used. Figure 2.2, below, plots the error performance versus the number of feature vectors used for classification. As expected, the error performance increases with the number of feature vectors used. Error performance was 21.35%, 6.35%, 5.95%, and 5.8% for 1, 2, 3, and 4 feature vectors, respectively. Figure 2.3, below is a sample of the support vector created for the 2-feature case.

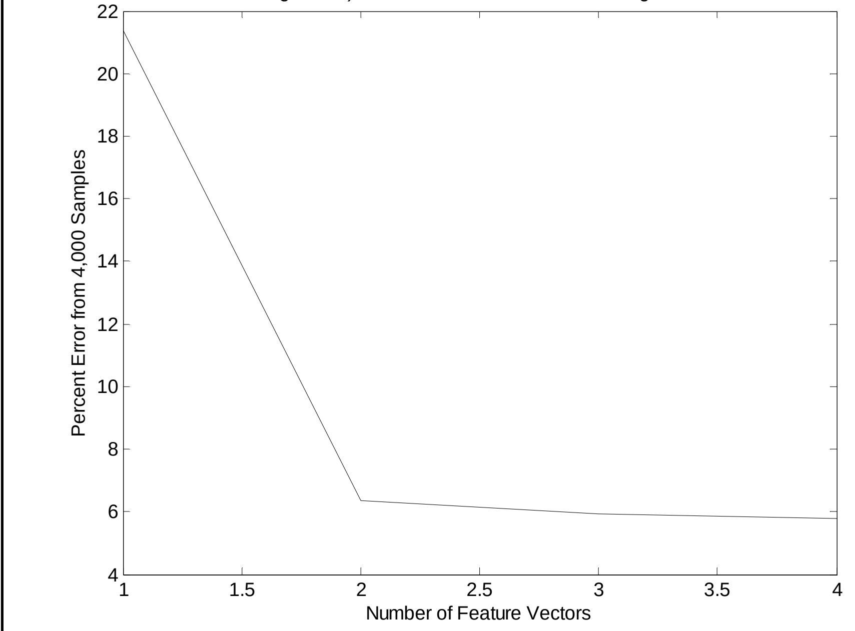


Figure 2.2) Classifier Error Performance Using SVM

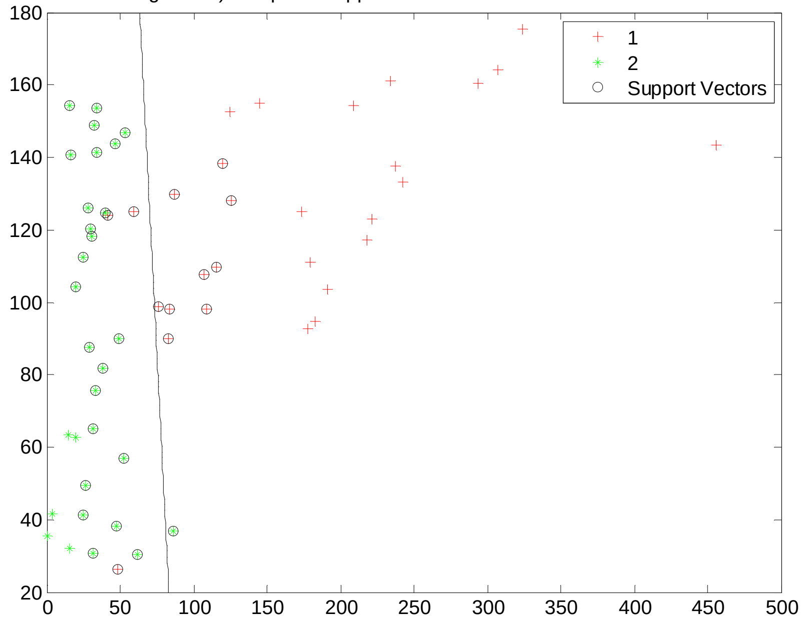


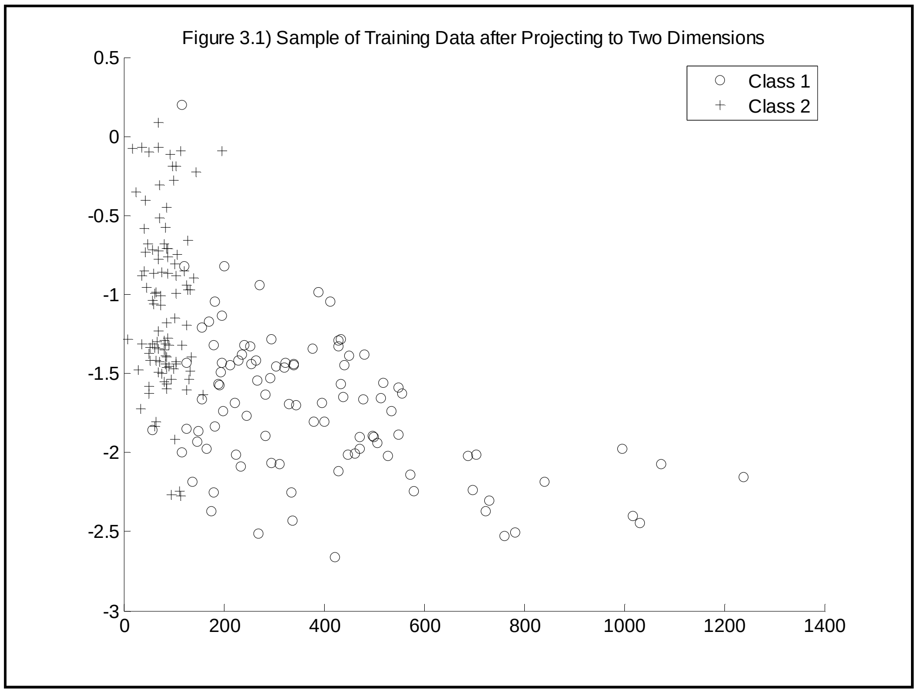
Figure 2.3) Sample of Support Vector Created for 2-Feature Case

Comparison of neural network approach and support vector machine approach:

The neural network (NN) approach provided better error performance overall compared to the support vector machine (SVM) approach. In fact, with each of 2, 3, and 4 feature vectors, the NN approach outperformed the SVM approach by over 1%. This one percent error performance increase does not sound like a lot, but when it is considered relative to the roughly 5% error performance of the NN, this is an increase of nearly 20%. For both of these systems, relatively small training sets of 300 points were used because of the complexity involved in training the algorithms. However, once training has been optimized, decision thresholds can be utilized to efficiently classify samples very quickly, thus providing great systems for many real-life classifier needs.

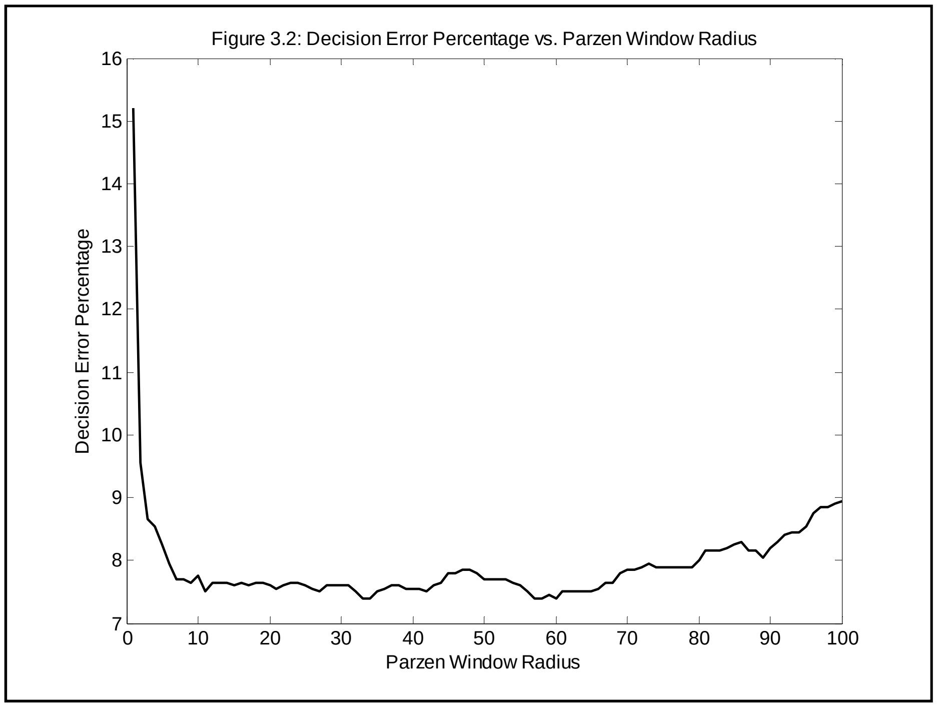
3) Designing Classifiers using Parzen Window Technique, K-Nearest Neighbor Technique, and Nearest Neighbor Technique:

Data for this problem were again obtained from the "svmguide1" dataset from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. The four feature data was first projected to two dimensions by using Fisher Linear Discriminants, similar to problem 1. Features 1 and 2 were combined into one feature, and features 3 and 4 were combined into a second feature. Figure 3.1, below, shows a sample of the training data points plotted after being mapped from four dimensions to two dimensions.

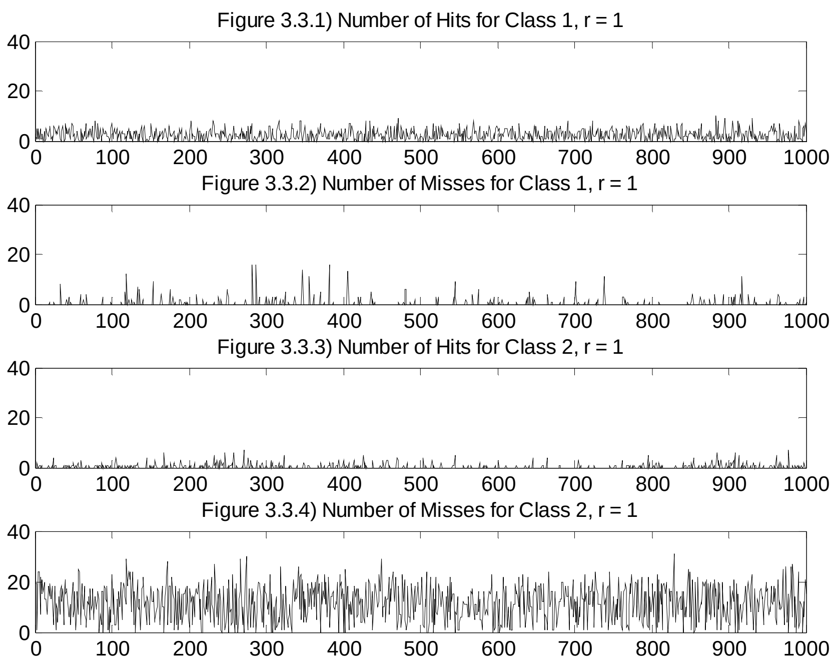


Parzen Window Technique:

The Parzen window technique classifier that we implemented is very straight forward. At each sample data point, the number of points from each class that fall within a radius r of that point are counted. Whichever class has a higher representation in that radius is the statistically favored, thus the sample is assigned to that class. Error performance for a range of radius values, r, is plotted in Figure 3.2, below. The best error performance comes with a rather large window size or r = 33, and a minimum error of 7.4%.



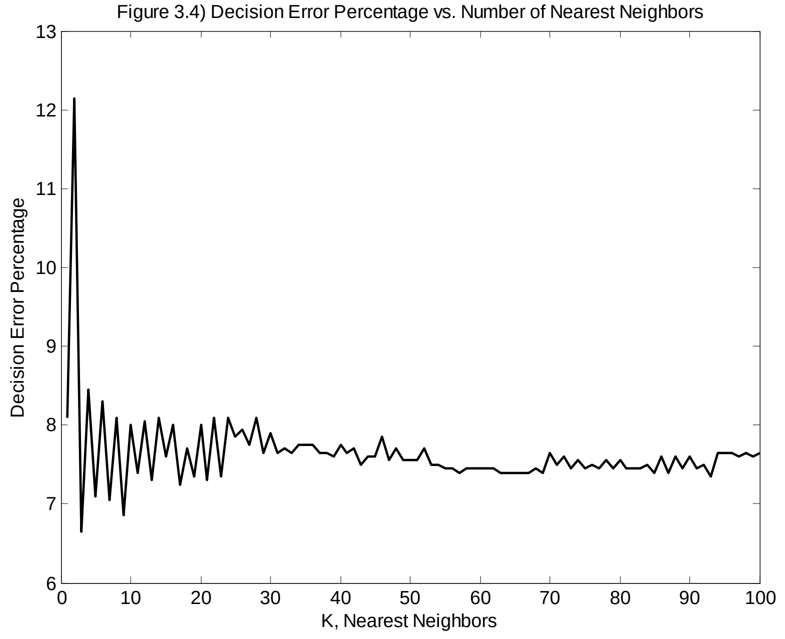
Figures 3.3.1 – 3.3.4, below, illustrate how the Parzen window decides a class for r = 1. The number of "hits" in the graph is the number of same-class training points that lie within the radius of the sample point. The number of "misses" is the number of different-class training points that lie within the radius of the sample point. When the number of hits is larger than the number of misses at a point, then the classification is correct. If it is less, then the classification is in error. Thus, Figure 3.3.1 and Figure 3.3.1 shows hits and misses for class 1, respectively, and Figure 3.3.3 and Figure 3.3.4 show hits and misses for class 2 respectively. Note that in Figures 3.3.1, the number of hits is fairly consistent, and in Figure 3.3.4, the number of misses is very large for the majority of the sample points. This implies that class 1 is tightly grouped; much more so than is class 2. The overlap of the data, and the relative closeness of Class 1 implies that r = 1 would be a poor choice for the radius of a Parzen window in this case.



K-Nearest Neighbor Technique:

The K-nearest neighbor classifier that was implemented is also very simple. The algorithm finds the K training points that are closest, in terms of Euclidean distance, to the sample point. The class is then decided by a majority vote of those training points, which also corresponds to the sample point coming from the class of highest probability. K is typically chosen to be an odd number, so that that the count of nearby classes cannot be equal. In figure 3.4, below, the affects of using an even number is evident by the spikes of higher error probabilities at most even K values. The figure shows the probability of error versus the number of K-nearest neighbors used in classification. The minimum error occurs at K = 3, with an error of 6.65%.

Figure 3.4) Decision Error Percentage vs. Number of Nearest Neighbors



Nearest Neighbor Technique:

The nearest neighbor technique is simply a special case of the K-nearest neighbor technique. In this instance, the algorithm simply assigns the class of the sample point to that of the training point that is the minimum Euclidean distance away. Figure 3.4, above, includes the nearest neighbor point, or the error corresponding to K = 1. The error is 8.1% for this data set, using the nearest neighbor classification technique.

Comparing Classifiers designed using Parzen Window Technique, K-Nearest Neighbor Technique, and Nearest Neighbor Technique:

Of the three classifier methods designed, K nearest neighbors produced the best error performance with minimum error of 6.65%. The Parzen window technique was second best with a minimum error performance of 7.4%, and the nearest neighbor technique was the worst, with an error of 8.1%. It is expected that nearest neighbor will perform worse than K-nearest neighbor, in general, for real data, as statistical outliers have a much higher probability of being misclassified. In fact, all three approaches yield poor performance for statistical outliers, which is a serious problem for use with real data sets. Overall, these methods are all quite computationally expensive, especially as the number of dimensions increases, which tends to imply that they would be poor choices for use with very large data sets.

```
%ECE 662
%Homework 2 - Problem 1
clear all; close all; clc;

pw1 = 1/2;
training = 300;
samples = 5000;
error_rat = zeros(20,20);

for sx1=.1:.1:2
    for sy1=.1:.1:2
        mu1 = [5 5];
        sigma1 = [sx1 0; 0 sy1];
        trainclass1 = mvnrnd(mu1, sigma1, round(pw1 * training));
        sampsclass1 = mvnrnd(mu1, sigma1, round(pw1 * samples));

        mu2 = [3 3];
        sigma2 = [1 0; 0 1];
        trainclass2 = mvnrnd(mu2, sigma2, training - round(pw1 * training));
        sampsclass2 = mvnrnd(mu2, sigma2, samples - round(pw1 * samples));

        m1 = mean(trainclass1,1); %mean of columns
        m2 = mean(trainclass2,1); %mean of columns
        n = length(m1);

        %scatter
        Sb = (m1'-m2')*(m1-m2); %2x1
        % Sw = sum yi elements of class i (yi-mi)*(yi'-mi')
        ym1 = trainclass1;
        ym2 = trainclass2;
        Swa = zeros(n,n);
        Swb = zeros(n,n);
        %subtracting the means
        for x = 1:n
            ym1(:,x) = ym1(:,x)-m1(x);
            ym2(:,x) = ym2(:,x)-m2(x);
        end

        [x1,y1] = size(trainclass1);
        [x2,y2] = size(trainclass2);

        for x = 1:x1
            Swa = Swa + ym1(x,:)'*ym1(x,:);
        end

        for x = 1:x2
            Swb = Swb + ym2(x,:)'*ym2(x,:);
        end

        Sw1 = ((ym1')*(ym1));
        Sw2 = ((ym2')*(ym2));
        Sw = Sw1+Sw2;
        % w = projection line;

        w0 = inv(Sw)*(m1-m2)';
        w0new = (m1-m2)';

        % Find threshold for w0
        z0 = w0' * [trainclass1' trainclass2'];
        z0 = sort(z0);
        if w0'*m1' < w0'*m2'
            w0_thr = (z0(round(pw1 * training)) + z0(round(pw1 * training) + 1))/2;
            thr_comp0 = 1;
        else
            w0_thr = (z0(training - round(pw1 * training)) + z0(training - round(pw1 *↙
training) + 1))/2;
            thr_comp0 = -1;
        end

        % Find threshold for w0new
        z0 = w0new' * [trainclass1' trainclass2'];
        z0 = sort(z0);
        if w0new'*m1' < w0new'*m2'
            w0new_thr = (z0(round(pw1 * training)) + z0(round(pw1 * training) + 1))/2;
            thr_comp_new = 1;
        else
            w0new_thr = (z0(training - round(pw1 * training)) + z0(training - round(pw1 *↙
training) + 1))/2;
            thr_comp_new = -1;
        end

        % Classify samples
        errors0 = sum([(w0'*sampsclass1' * thr_comp0 > w0_thr * thr_comp0) ...
            (w0'*sampsclass2' * thr_comp0 < w0_thr * thr_comp0)]);
        errors0new = sum([(w0new'*sampsclass1' * thr_comp_new > ...
            w0new_thr * thr_comp_new) (w0new'*sampsclass2' * ...
            thr_comp_new < w0new_thr * thr_comp_new)]);

        error_rat(uint8(sx1*10),uint8(sy1*10)) = errors0/errors0new;

    end
end

mesh(.1:.1:2,.1:.1:2,error_rat)
xlabel('Variance in x')
ylabel('Variance in y')
zlabel('(Errors using S_w) / (Errors using S_w = I)')
title('Error ratio with one fixed \Sigma and one varying \Sigma')
colormap('gray')
```

```
% ECE662 HW#2 Problem 2a

clear all; close all; clc;

load svmguide;

x = [trainclass1(1:150,:); trainclass2(1:150,:)];
t = [ones(150,1);2*ones(150,1)];
x_test = [testclass1(:,:);testclass2(:,:)];
t_test = [ones(2000,1);2*ones(2000,1)];

errors4 = zeros(20,1);

for Nh=1:20
    results = nc_main(x,t,x_test,t_test,Nh);
    classes = results.t_est_test;
    error = (classes ~= [ones(2000,1);2*ones(2000,1)]);
    errors4(Nh) = sum(error)/length(error)*100;
end

x = [trainclass1(1:150,[1,2,4]); trainclass2(1:150,[1,2,4])];
t = [ones(150,1);2*ones(150,1)];
x_test = [testclass1(:,[1,2,4]);testclass2(:,[1,2,4])];
t_test = [ones(2000,1);2*ones(2000,1)];

errors3 = zeros(20,1);

for Nh=1:20
    results = nc_main(x,t,x_test,t_test,Nh);
    classes = results.t_est_test;
    error = (classes ~= [ones(2000,1);2*ones(2000,1)]);
    errors3(Nh) = sum(error)/length(error)*100;
end

x = [trainclass1(1:150,[2,4]); trainclass2(1:150,[2,4])];
t = [ones(150,1);2*ones(150,1)];
x_test = [testclass1(:,[2,4]);testclass2(:,[2,4])];
t_test = [ones(2000,1);2*ones(2000,1)];

errors2 = zeros(20,1);

for Nh=1:20
    results = nc_main(x,t,x_test,t_test,Nh);
    classes = results.t_est_test;
    error = (classes ~= [ones(2000,1);2*ones(2000,1)]);
    errors2(Nh) = sum(error)/length(error)*100;
end

%% Plot Results
plot([errors4 errors3 errors2])
legend('4 Features','3 Features','2 Features')
title('Figure 2.1) Percent of Classification Errors from 4,000 Samples')
xlabel('Number of Hidden Layers')
ylabel('Percent Error in Classification')
```

```
% ECE662 HW#2 Problem 2b

clear all; close all; clc;

load svmguide;

Training = [trainclass1(1:150,:); trainclass2(1:150,:)];
Group = [ones(150,1);2*ones(150,1)];
SVMStruct = svmtrain(Training, Group);%, 'Showplot',true);
classes = svmclassify(SVMStruct,[testclass1(:,:);testclass2(:,:)]);%,'showplot',true);
error = (classes ~= [ones(2000,1);2*ones(2000,1)]);
percent_error4 = sum(error)/length(error)*100;

Training = [trainclass1(1:150,[1,2,4]); trainclass2(1:150,[1,2,4])];
Group = [ones(150,1);2*ones(150,1)];
SVMStruct = svmtrain(Training, Group);%, 'Showplot',true);
classes = svmclassify(SVMStruct,[testclass1(:,[1,2,4]);testclass2(:,↙
[1,2,4])]);%,'showplot',true);
error = (classes ~= [ones(2000,1);2*ones(2000,1)]);
percent_error3 = sum(error)/length(error)*100;

Training = [trainclass1(1:150,[2,4]); trainclass2(1:150,[2,4])];
Group = [ones(150,1);2*ones(150,1)];
SVMStruct = svmtrain(Training, Group);%, 'Showplot',true);
classes = svmclassify(SVMStruct,[testclass1(:,[2,4]);testclass2(:,[2,4])]);%,'showplot',↙
true);
error = (classes ~= [ones(2000,1);2*ones(2000,1)]);
percent_error2 = sum(error)/length(error)*100;

Training = [trainclass1(1:150,1); trainclass2(1:150,1)];
Group = [ones(150,1);2*ones(150,1)];
SVMStruct = svmtrain(Training, Group);%, 'Showplot',true);
classes = svmclassify(SVMStruct,[testclass1(:,1);testclass2(:,1)]);%,'showplot',true);
error = (classes ~= [ones(2000,1);2*ones(2000,1)]);
percent_error1 = sum(error)/length(error)*100;

plot([percent_error1,percent_error2,percent_error3,percent_error4])
title('Figure 2.2) Classifier Error Performance Using SVM')
xlabel('Number of Feature Vectors')
ylabel('Percent Error from 4,000 Samples')
```

```
% ECE662
% Homework 2
% Problem 3
close all; clear all; clc;
tic
load('svmguide.mat');
% http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#svmgui
% de1 taking svmguide1 data
% data 3089x10
% test 4000x10
% trainclass1 2000x4
% trainclass2 1089x4
% testclass1 2000x4
% testclass2 2000x4

%% Project 4 dimensions to two (two to one and two to one)
trainc11 = trainclass1(1:1000,1:2);
trainc12 = trainclass2(1:1000,1:2);
testc11 = testclass1(1:1000,1:2);
testc12 = testclass2(1:1000,1:2);

trainc21 = trainclass1(1:1000,3:4);
trainc22 = trainclass2(1:1000,3:4);
testc21 = testclass1(1:1000,3:4);
testc22 = testclass2(1:1000,3:4);

% Combine features 1&2 into 1
m1 = mean(trainc11,1); %mean of columns
m2 = mean(trainc12,1); %mean of columns
n = length(m1);
%scatter
% Sw = sum yi elements of class i (yi-mi)*(yi'-mi')
ym1 = trainc11;
ym2 = trainc12;
%subtracting the means
for x = 1:n
    ym1(:,x) = ym1(:,x)-m1(x);
    ym2(:,x) = ym2(:,x)-m2(x);
end
Sw1 = ((ym1')*(ym1));
Sw2 = ((ym2')*(ym2));
Sw = Sw1+Sw2;
% w = projection line;
w0 = inv(Sw/(Sw(1,1)))*(m1-m2)';
w0 = w0/(w0(1));

train1 = zeros(1000,3);
train2 = zeros(1000,3);
test1 = zeros(1000,2);
test2 = zeros(1000,2);
train1(:,1) = (w0'*trainc11')';
train2(:,1) = (w0'*trainc12')';
test1(:,1) = (w0'*testc11')';
test2(:,1) = (w0'*testc12')';

% Combine features 3&4 into 1
```

```
m1 = mean(trainc21,1); %mean of columns
m2 = mean(trainc22,1); %mean of columns
n = length(m1);
%scatter
% Sw = sum yi elements of class i (yi-mi)*(yi'-mi')
ym1 = trainc21;
ym2 = trainc22;
%subtracting the means
for x = 1:n
    ym1(:,x) = ym1(:,x)-m1(x);
    ym2(:,x) = ym2(:,x)-m2(x);
end
Sw1 = ((ym1')*(ym1));
Sw2 = ((ym2')*(ym2));
Sw = Sw1+Sw2;
% w = projection line;
w0 = inv(Sw/(Sw(1,1)))*(m1-m2)';
w0 = w0/(w0(1));

train1(:,2) = (w0'*trainc21')';
train2(:,2) = (w0'*trainc22')';
test1(:,2) = (w0'*testc21')';
test2(:,2) = (w0'*testc22')';
%%
train1(:,3) = 1; %corresponds to class1
train2(:,3) = -1;%corresponds to class2
% figure(1)
% hold on;
% plot(train1(1:100,1),train1(1:100,2), 'ro');
% plot(train2(1:100,1),train2(1:100,2), 'b+');
% % axis([0 1400 -3 .5])
% title('Figure 3.1) Sample of Training Data after Projecting to Two Dimensions')
% figure(2)
% hold on;
% plot(test1(:,1),test1(:,2),'ro');
% plot(test2(:,1),test2(:,2),'b+');
% axis([0 1400 -3 .5])

% Parzen Window
r = 1;
% r_max = 100;
% parzen_errors = zeros(r_max,1);
% for r=1:r_max
% K Nearest Neighbor knn
% Nearest Neighbor nn
k = 5;
knn_errors = zeros(100,1);
for k=1:100
listnear1 = zeros(2*k,2);%distance, value
listnear2 = zeros(2*k,2);%distance, value
distance1 = zeros(1000,1);%distances to the train1 values
distance2 = zeros(1000,1);%distances to the train2 values
distance3 = zeros(1000,1);%distances to the train1 values
distance4 = zeros(1000,1);%distances to the train2 values
parzen1a = zeros(1000,1);%probability of class 1 using parzen
parzen1b = zeros(1000,1);%probability of class 1 using parzen
```

```
parzen2a = zeros(1000,1);%probability of class 2 using parzen
parzen2b = zeros(1000,1);%probability of class 2 using parzen
parzen1 = zeros(1000,1);%correct decision of parzen for class1
parzen2 = zeros(1000,1);%correct decision of parzen for class2
knn1 = zeros(1000,1);%correct decision of k nearest neighbors for class1
knn2 = zeros(1000,1);%correct decision of k nearest neighbors for class2
nn1 = zeros(1000,1);%correct decision of nearest neighbor for class1
nn2 = zeros(1000,1);%correct decision of nearest neighbor for class2
% index1 = zeros(1000,2*k);
% index2 = zeros(1000,2*k);

%for each of the 1000 test points (both classes)
for x = 1:1000
    %for test data in class1
    %distances to all training points
    distance1 = sqrt((train1(:,1)-test1(x,1)).^2+(train1(:,2)-test1(x,2)).^2);
    distance2 = sqrt((train2(:,1)-test1(x,1)).^2+(train2(:,2)-test1(x,2)).^2);

    [distance1] = sort(distance1,'ascend');%sort distances to training class1
    [distance2] = sort(distance2,'ascend');%sort distances to training class2

    parzen1(x) = (sum(distance1<r)-sum(distance2<r))>0; %if more class1 than class2
    parzen1a(x) = sum(distance1<r);%number of class 1 in window (hit)
    parzen1b(x) = sum(distance2<r);%number of class 2 in window (miss)

    %concatenate the shortest distances to both classes
    listnear1(1:k,1) = distance1(1:k,1);
    listnear1(1:k,2) = 1; %class 1
    listnear1(k+1:2*k,1) = distance2(1:k,1);
    listnear1(k+1:2*k,2) = -1; %class 2

    %_________________________________________________
    [listnear index1] = sort(listnear1(:,1),'ascend'); %sort for all nearest
    decision1 = sum(listnear1(index1(1:k),2));
    knn1(x) = decision1>0;
    nn1(x) = listnear1(index1(1),2)>0;

    %for test data in class2
    distance3 = sqrt((train1(:,1)-test2(x,1)).^2+(train1(:,2)-test2(x,2)).^2);
    distance4 = sqrt((train2(:,1)-test2(x,1)).^2+(train2(:,2)-test2(x,2)).^2);

    [distance3] = sort(distance3,'ascend');
    [distance4] = sort(distance4,'ascend');

    parzen2(x) = (sum(distance4<r)-sum(distance3<r))>0;%if more class2 than class1
    parzen2a(x) = sum(distance3<r);%number of class 1 in window (miss)
    parzen2b(x) = sum(distance4<r);%number of class 2 in window (hit)

    listnear2(1:k,1) = distance3(1:k,1);
    listnear2(1:k,2) = 1;
    listnear2(k+1:2*k,1) = distance4(1:k,1);
    listnear2(k+1:2*k,2) = -1;
    %___________________________________________________
    [listnear index2] = sort(listnear2(:,1),'ascend');
    decision2 = sum(listnear2(index2(1:k),2));
    knn2(x) = decision2<0;
    nn2(x) = listnear2(index2(1),2)<0;


end

% figure(3)
% subplot(4,1,1)
% plot(parzen1a)
% title(['Figure 3.3.1) Number of Hits for Class 1, r = ',num2str(r)])
% axis([0 1000 0 40])
%
% subplot(4,1,2)
% plot(parzen1b)
% title(['Figure 3.3.2) Number of Misses for Class 1, r = ',num2str(r)])
% axis([0 1000 0 40])
%
% subplot(4,1,3)
% plot(parzen2a)
% title(['Figure 3.3.3) Number of Hits for Class 2, r = ',num2str(r)])
% axis([0 1000 0 40])
%
% subplot(4,1,4)
% plot(parzen2b)
% title(['Figure 3.3.4) Number of Misses for Class 2, r = ',num2str(r)])
% axis([0 1000 0 40])

percentparzen = (2000-sum(parzen1)-sum(parzen2))/2000*100;
percenterrknn = (2000-sum(knn1)-sum(knn2))/2000*100;
percenterrnn = (2000-sum(nn1)-sum(nn2))/2000*100;
knn_errors(k,1) = percenterrknn;
end
figure
plot(knn_errors)
title('Figure 3.4) Decision Error Percentage vs. Number of Nearest Neighbors')
xlabel('K, Nearest Neighbors')
ylabel('Decision Error Percentage')
%     parzen_errors(r,1) = percentparzen;
% end
% figure
% plot(parzen_errors)
% title('Figure 3.2: Decision Error Percentage vs. Parzen Window Radius')
% xlabel('Parzen Window Radius')
% ylabel('Decision Error Percentage')
```