ECE662, Homework#2

April 1, 2008

1. Consider the Fisher linear discriminant which is used to classify data into two classes via linear separation by minimizing a cost function $J(w)$. If we let m_1 and m_2 be the sample means of the data from classes 1 and 2, respectively, then

$$\begin{aligned} J(w) &\triangleq \frac{\|w \cdot (m_1 - m_2)\|^2}{\sum_{y_i \in \text{class1}} w \cdot (y_i - m_1) + \sum_{y_j \in \text{class2}} w \cdot (y_j - m_2)} \\ &= \frac{w^T S_B w}{w^T S_W w}, \qquad (1) \\ \text{where } S_B &\triangleq (m_1 - m_2)(m_1^T - m_2^T), \\ S_W &\triangleq \sum_{y_i \in \text{class1}} (y_i - m_1)(y_i^T - m_1^T) + \sum_{y_j \in \text{class2}} (y_j - m_2)(y_j^T - m_2^T). \end{aligned}$$

   We have that the minimizer of this function $J(w)$ is $w_0 = S_W^{-1}(m_1 - m_2)$. The set $\{\alpha w_0 | \alpha \in \mathbb{R}\}$ defines a plane on which the projection of the data will be maximally-separated. We then classify a data point x as class 1 if $(x - \bar{x}) \cdot w_0 > 0$, and class 2 if $(x - \bar{x}) \cdot w < 0$, where $\bar{x}$ is the sample mean of x.

   We explore here whether it would suffice to let $w_0 = (m_1 - m_2)$ in order to separate the data from the two classes (essentially putting $S_W \triangleq I$). We would define a proposed discriminant in the same manner as the Fisher discriminant, with a data point x classified as class 1 if $(x - \bar{x}) \cdot w_0 > 0$ (using the proposed definition for w_0), and class 2 otherwise. We carried out several numerical trials to compare the two definitions of w_0. Figure 1 displays the results of some of these experiments.

   We find that the denominator in (1) allows the Fisher discriminant to minimize the "spread" of the projected data onto $w_0 = S_W^{-1}(m_1 - m_2)$.

The proposed discriminant does not account for this, so we would expect that the Fisher discriminant would outperform the proposed discriminant if the data had a good deal of "spread" in a particular direction in the plane. This is precisely what was found in the experiments shown in Figure 1. We see that Experiment 1 (row 1) has data identically-spread in both dimensions, while Experiment 2 data is spread widely in one (horizontal) direction, but not the other. As expected, the discriminants perform the same in Experiment 1 (relative to P_{error}, the ratio of misclassified samples), but the Fisher discriminant significantly outperforms the proposed discriminant in Experiment 2. Each class of Experiment 3 has a high variance ("spread") in one dimension but not the other, however, unlike Experiment 2, the two classes are spread across perpendicular directions such that the spread is minimized when projected onto $w_0 = (m_1 - m_2)$. For this reason, the proposed discriminant performs similar to the Fisher discriminant (in fact, slightly better) in this case.

Motivated by the results of Experiment 2 (shown in Figure 1), we designed another experiment to test the performance of each discriminant as the spread of both class data is increased in a single direction. We let $m_1 = [1, 1]^T$, $m_2 = [-1, -1]^T$, and $\Sigma_1 = \Sigma_2 = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix}$, with alpha ranging from 1 to 20. Figure 2 shows that as α ("spread" in dimension 1) is increased from 1 to 20 that the Fisher discriminant increasingly outperforms the proposed discriminant in terms of P_{error}.

2. In order to compare classification approaches of artificial neural networks (ANN) and support vector machines (SVM), we develop sets of training data paired with test data. Each data set has two classes, as the software used for artificial neural networks functions only with two-class data. Several 2D Gaussian-distributed class data sets were generated from Matlab random number generators (note that test data and training for each experiment were generated from the same distributions). Experiments with both ANN and SVM were carried out using three datasets generated with classes having the following means (μ_i, $i = 1, 2$) and covariance matrices (Σ_i, $i = 1, 2$) where subscript i

indicates the class:

Experiment 3: $\mu_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, \mu_2 = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

Experiment 4: $\mu_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mu_2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

Experiment 5: $\mu_1 = \mu_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$

All experiments had equal priors $\pi_1 = \pi_2 = 0.5$. We now investigate the approaches of ANN and SVM.

(a) Artifical Neural Network (ANN) approach. The code used for these experiments is an implementation of the multi-layer perceptron (MLP) backpropagation algorithm, written by Yu Hen Hu of University of Wisconson, found at http://homepages.cae.wisc.edu/ ece539/matlab/.

This code gives the user plenty of room to design the neural network to their pleasing, however, we used the default settings wherever possible. This resulted in creating a two-layer neural network (one hidden layer, one output layer), with two neurons in the hidden layer. The hidden layer used a hyperbolic tangent (tanh) activation function (scaled to -0.8 to 0.8), and output layer uses a sigmoidal activation function (scaled to 0.2 to 0.8). The code runs iteratively, checking convergence at our specified interval of five iterations. We terminate the evolution at 100 iterations.

Results of experiments with this ANN are shown in Figure 3.

(b) Support Vector Machine (SVM) approach. The code used for these experiments is part of an extensive SVM code library written by Chih-Jen Lin of National Taiwan University and Chih-Chung Chang. The code can be found at http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

This code functions as a typical SVM, training the support vectors on user input training data (as described above) before classifying the test data.

Results of experiments with this SVM are shown in Figure 3.

(c) Comparison of ANN and SVM. We compare the performance of the ANN and SVM on the three two-class datasets specified above. Figure 3 shows a comparison in the performace across these datasets, plotting the class samples, and specifying the misclassified samples and P_{error}. We find that across all datasets

(which vary in their class separability), ANN and SVM perform virtually identically, although differing slightly in precisely which samples are misclassified.

3. We now use the same three datasets from Question 2 to compare non-parametric classifiers defined by each of Parzen windows (PW), K-nearest neighbors (KNN), and nearest neighbor (NN).

   (a) Parzen window (PW) technique. For the parzen window technique, we use a set of training data (in particular, those created for Question 2) to drive the classifier. At each 2D data point of the data set, we center a 2D circular Parzen window, with a user-defined radius. The point is then classified as the class which has the most training data points inside of the circular Parzen window. In the case that there is a "tie" between classes, a class label is randomly selected with probability proportional to its prior, as estimated by the training set. If there are no training data points inside the Parzen window, the data point is left unclassified.

   The results of the Parzen window classifier on the three datasets (Experiments 3, 4, and 5) from Question 2 are shown in Figure 4. We use various settings for the radius r of the Parzen window, displaying performance at $r = 0.2$, 0.5, and 1.5 in Figure 4. We also evaluate the performance at a finer sampling of window radii and record P_{error} and P_{unclass} (the ratio of samples left unclassified) as r is increased. The results of this experiment are shown in Figure 5.

   We see from the results in both Figures 4 and 5 that the performance of the Parzen window classifier generally increases with an increase in window size (at least in the range of sizes tested). We find that for small window sizes ($r < 0.4$), a large majority of samples are left unclassified, driving up P_{error} (note that the samples which are unclassified are also considered incorrectly labelled, such that P_{error} increases with any increase in P_{unclass}). On the other hand, with sufficiently large window sizes ($r > 1$), the Parzen window classifier outperforms the ANN and SVM of Question 2.

   (b) K-nearest neighbor (KNN) technique. Similar to the Parzen window technique, we use training data to drive a classifier. This time we do not use a set circular window size, but a variable one such that the radius is set such that precisely K train samples

fall within the window. The class of which the majority of the K training samples in the window are labeled is the class which the data point is labelled. In short, for each data point in the data set, we measure the (Euclidean) distance to each of the training data points, and use the K closest to determine the classification of the data point (by "majority vote"). If there is a "tie vote," as in the Parzen window technique, the tie is broken by randomly selecting a class label with probability proportional to its prior, as estimated by the training set. Note that we avoid ties in our two-class case by letting K be an odd integer.

We again test this technique on the three data sets from Question 2. The results of the KNN classifier for various settings of K is shown in Figure 6. Similar to Figure 5 for the Parzen window, Figure 7 shows how the performance (P_{error}) of the KNN classifier changes with adjustments to parameter K.

From both Figures 6 and 7, we find that the performance of the KNN classifier varies very little with changes in K. However, we find that performance begins degrading when K is increased beyond about 20 (see Figure 7 for Experiments 3 and 5).

(c) Nearest neighbor (NN) technique. This technique may be viewed as a special case of the KNN classifier, with $K = 1$. In this case, we simply label every sample in the dataset as the class of the nearest training sample (as measured by a Euclidian distance metric). Unlike the Parzen window or KNN techniques, there is no parameter to adjust. Results of the NN classifier are shown in Figure 8.

(d) Comparison of PW, KNN and NN classifiers. As seen from Figures 5 and 7 and PW and KNN classifier performance changes with adjustments to the parameters r and K, respectively. In order to fairly compare the two, we compare their best performance across their parameter settings on the three datasets in Table 1. The performance of the NN classifier (as seen in Figure 8) is also recorded in Table 1.

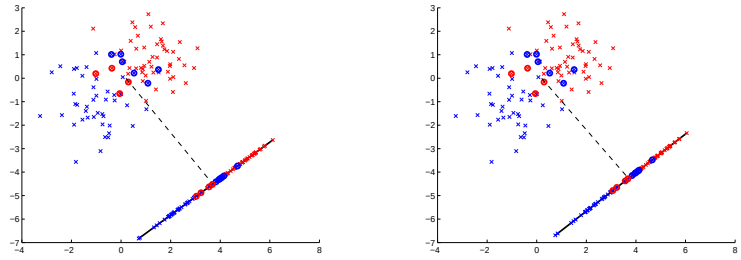
We see from Table 1 that the KNN classifier consistently outperforms the NN classifier. This is to be expected, since the NN classifier is simply a special case of the KNN classifier. We also find, however, that the KNN classifier also consistently outperforms the PW classifier. This may be expected, as the KNN classifier is more computationally-expensive (requiring a sort op-

	Best Case P_{error}		
Experiment	PW classifier	KNN classifier	NN classifier
Experiment 3	0.289	0.268	0.320
Experiment 4	0.140	0.112	0.148
Experiment 5	0.210	0.183	0.236

Table 1: Question 3(d). Comparison of the best case performance (lowest P_{error}) of each of the Parzen window (PW), K-nearest neighbor (KNN), and nearest neighbor (NN) classifiers across the three data sets from Question 2.

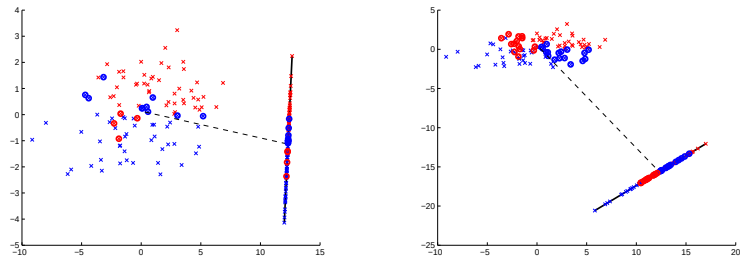
eration) than the Parzen window classifier, and therefore could be considered a more sophistocated algorithm. We also find, overall, that each of the PW and KNN classifiers perform better than either ANN or SVM classifiers of Question 2.

Experiment 1: $m_1 = -m_2 = \begin{bmatrix}1\\1\end{bmatrix}$, $\Sigma_1 = \Sigma_2 = I_2$.



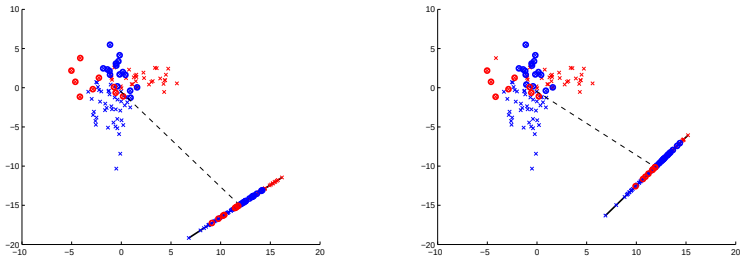
Fisher discriminant: $P_{\text{error}} = 0.1$ Proposed discriminant: $P_{\text{error}} = 0.1$

Experiment 2: $m_1 = -m_2 = \begin{bmatrix}1\\1\end{bmatrix}$, $\Sigma_1 = \Sigma_2 = \begin{bmatrix}8 & 0\\0 & 1\end{bmatrix}$.



Fisher discriminant: $P_{\text{error}} = 0.14$ Proposed discriminant: $P_{\text{error}} = 0.31$

Experiment 3: $m_1 = -m_2 = \begin{bmatrix}1\\1\end{bmatrix}$, $\Sigma_1 = \begin{bmatrix}8 & 0\\0 & 1\end{bmatrix}$, $\Sigma_2 = \begin{bmatrix}1 & 0\\0 & 8\end{bmatrix}$



Fisher discriminant: $P_{\text{error}} = 0.25$ Proposed discriminant: $P_{\text{error}} = 0.24$

Figure 1: Question 1. Empirical comparison of Fisher discriminant plane $w_0 = S_W^{-1}(m_1 - m_2)$ to the proposed plane $w_0 = (m_1 - m_2)$. Each row represents a different experiment with parameters for the Gaussian-distributed data from each class is written above each pair of figures. Data samples of each class distinguished by colors, while the vector w_0 is shown in solid black with the data points orthogonally-projected onto the vector. The decision boundary perpendicular to w_0 shown as a dashed line, and misclassified samples circled. Note that the decision boundary may not appear perpendicular to w_0 due to different scaling on each axis.

$$m_1 = [1, 1]^T,\ m_2 = [-1, -1]^T,\ \Sigma_1 = \Sigma_2 = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix}$$

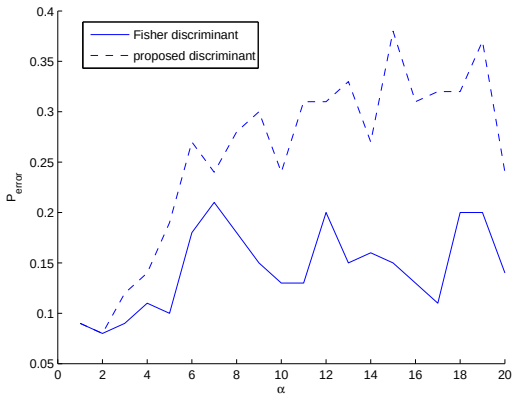
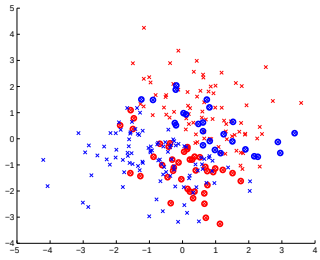
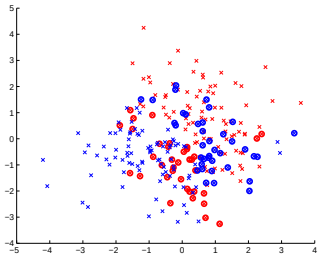


Figure 2: Question 1. P_{error} of the two discriminant functions as the spread of the data of each class is increased in a single direction.

Experiment 3

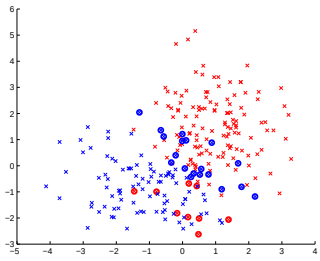


ANN: $P_{\text{error}} = 0.272$

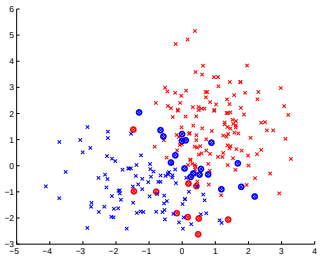


SVM: $P_{\text{error}} = 0.296$

Experiment 4

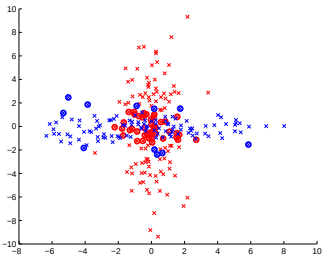


ANN: $P_{\text{error}} = 0.112$

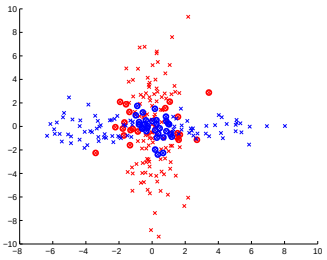


SVM: $P_{\text{error}} = 0.116$

Experiment 5



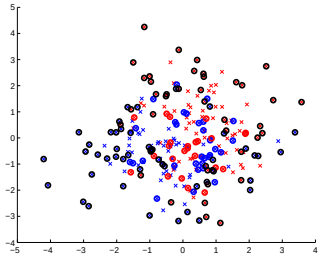
ANN: $P_{\text{error}} = 0.216$



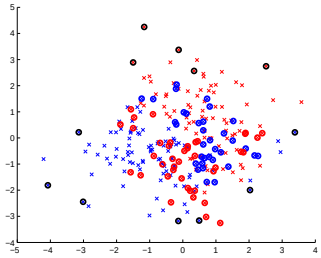
SVM: $P_{\text{error}} = 0.208$

Figure 3: Question 2. Comparison of performance of ANN and SVM on three datasets. Data samples of each class distinguished by colors while misclassified samples are circled. P_{error} for each classifier in each experiment listed below its corresponding figure.

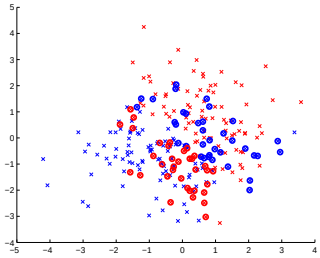
Experiment 3



$r = 0.2$: $P_{\text{error}} = 0.572$ ($P_{\text{unclass}} = 0.352$)

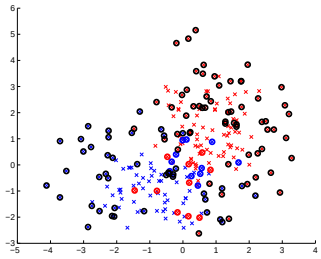


$r = 0.5$: $P_{\text{error}} = 0.344$ ($P_{\text{unclass}} = 0.048$)

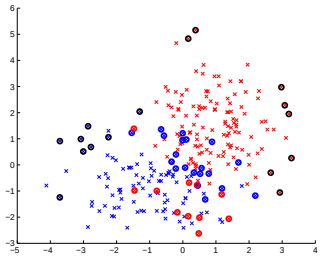


$r = 1.5$: $P_{\text{error}} = 0.240$ ($P_{\text{unclass}} = 0$)

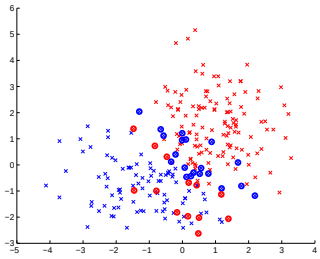
Experiment 4



$r = 0.2$: $P_{\text{error}} = 0.488$ ($P_{\text{unclass}} = 0.404$)

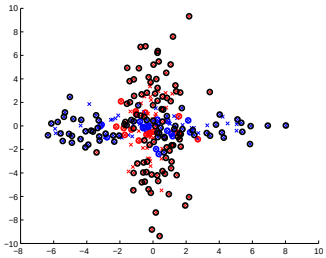


$r = 0.5$: $P_{\text{error}} = 0.188$ ($P_{\text{unclass}} = 0.064$)

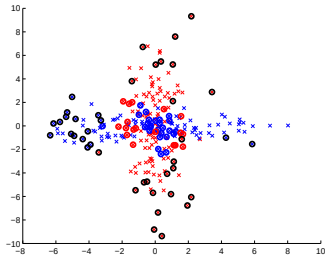


$r = 1.5$: $P_{\text{error}} = 0.132$ ($P_{\text{unclass}} = 0$)

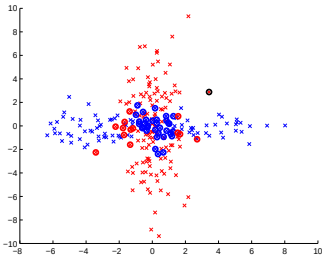
Experiment 5



$r = 0.2$: $P_{\text{error}} = 0.712$ ($P_{\text{unclass}} = 0.572$)

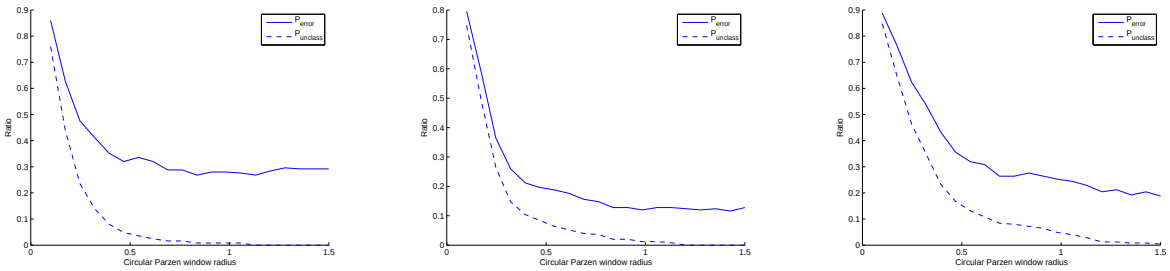


$r = 0.5$: $P_{\text{error}} = 0.340$ ($P_{\text{unclass}} = 0.164$)



$r = 1.5$: $P_{\text{error}} = 0.188$ ($P_{\text{unclass}} = 0.004$)

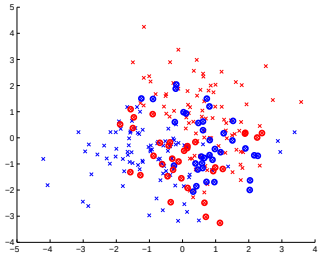
Figure 4: Question 3(a). Results of Parzen window classifier (with various circular window radii) on the three test data sets from Question 2. The separate classes are indicated by color, with misclassified samples circled. *Unclassified* samples are circled in black. P_{error} and P_{unclass} (the ratio of unclassified samples) are indicated below each figure.



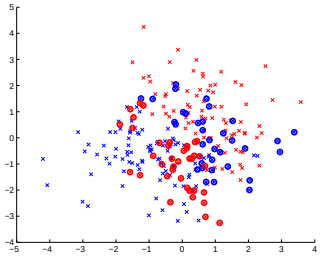
Experiment 3 Experiment 4 Experiment 5

Figure 5: Question 3(a). Performance of Parzen window classifier (in terms of P_{error} and P_{unclass}) with adjustments to the radius of the circular window. Results for each of the three data sets from Question 2 shown.

Experiment 3

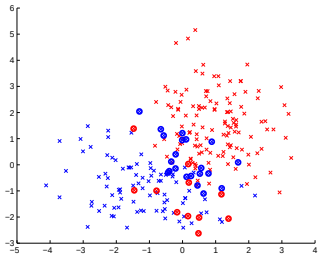


$K = 3$: $P_{\text{error}} = 0.276$

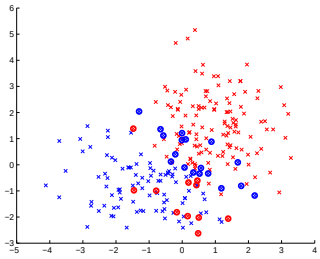


$K = 21$: $P_{\text{error}} = 0.288$

Experiment 4

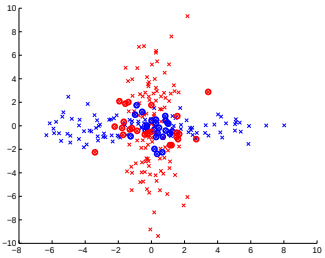


$K = 3$: $P_{\text{error}} = 0.128$

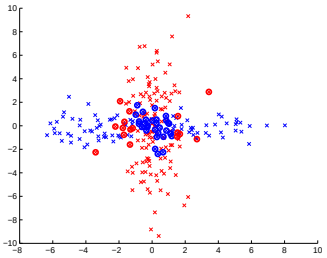


$K = 21$: $P_{\text{error}} = 0.116$

Experiment 5

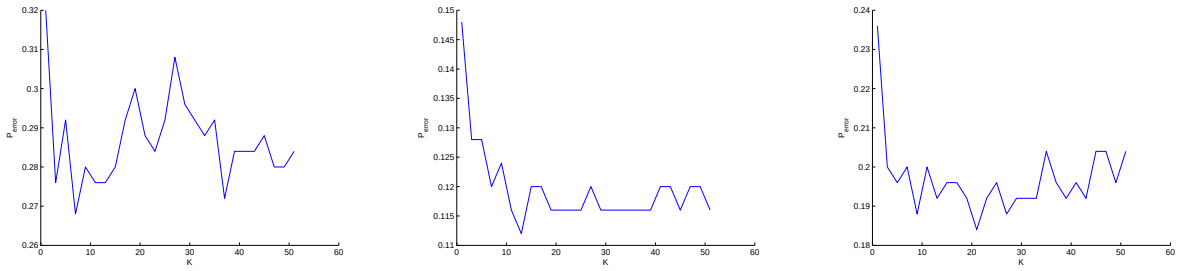


$K = 3$: $P_{\text{error}} = 0.200$



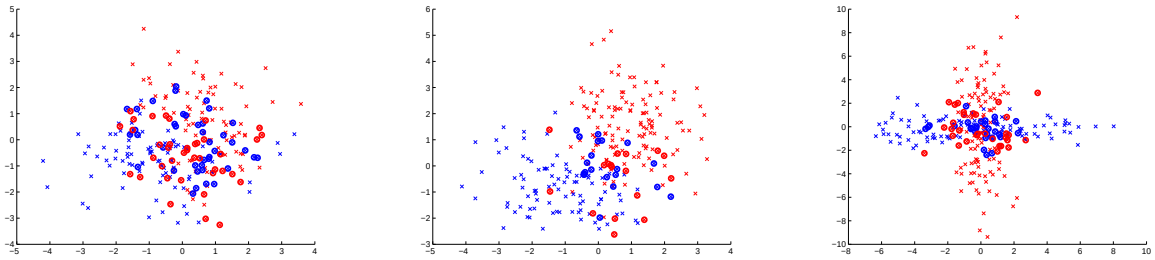
$K = 21$: $P_{\text{error}} = 0.184$

Figure 6: Question 3(b). Results of KNN classfication (with $K = 3$ and $K = 21$) on the three test data sets from Question 2. The separate classes are indicated by color, with misclassified samples circled. P_{error} is indicated below each figure.



Experiment 3 Experiment 4 Experiment 5

Figure 7: Question 3(b). Performance of KNN classifier (in terms of P_{error}) with adjustments to the parameter K. Results for each of the three data sets from Question 2 shown.



Experiment 3: $P_{\text{error}} = 0.32$ Experiment 4: $P_{\text{error}} = 0.148$ Experiment 5: $P_{\text{error}} = 0.236$

Figure 8: Question 3(c). Results of NN classification on the three test data sets from Question 2. The separate classes are indicated by color, with misclassified samples circled. P_{error} is indicated below each figure.

FisherDiscriminant.m

```
function [FisherPerr,proposedPerr] = FisherDiscriminant(m1,cov1,prior1,m2,cov2,prior2)
%Function compares the Fisher linear discriminant function to a proposed
%function obtained by projecting the class data onto the plane (w0-w1).
%
%Two-class data is generated (based on provided input parameters)
%and classified using the Fisher linear
%discriminant and the proposed discriminant.  The data is plotted, and
%projection onto each (Fisher and proposed) plane is visualized.
%Misclassified data is highlighted in the plot and Perror calculated for
%both discriminant functions.

%Extract and check input dimensions
n = length(m1);

if(size(m1)~=size(m2))
   error('Please enter identically-sized mean matrices');
end

[M,N] = size(cov1);
if(M~=N)
   error('Please enter a square covariance matrix');
elseif(N~=n)
   error('Pleast enter a covariance matrix with dimensions matching the mean input');
elseif(size(cov1)~=size(cov2));
   error('Please enter identically-sized covariance matrices');
end

%Assure that the priors are normalized (i.e. prior1 + prior2 = 1)
cst = (prior1 + prior2);
prior1 = prior1/cst;
prior2 = prior2/cst;

%Generate experimental data from two classes
means = [m1,m2]; priors = [prior1,prior2];
covs = zeros([size(cov1),2]); covs(:,:,1) = cov1; covs(:,:,2) = cov2;

numsamp = 100;
[x,trueclass] = genMultClassSamps(means,covs,priors,numsamp);
class1samps = find(trueclass==0);
class2samps = find(trueclass>0);

%Calculate sample means
m1hat = mean(x(:,class1samps),2); m2hat = mean(x(:,class2samps),2); %@@@@ MAY NOT ACTUALLY USE
SAMPLE MEANS!@@@@
%Calculate "between scatter" matrix
Sb = (m1hat - m2hat)*(m1hat' - m2hat');
%Calculate "within scatter" using cov command.
Sw1 = cov(x(:,class1samps)')*(length(class1samps)-1); %multiply by the
%length of the vector - 1 to undo scaling in cov command
Sw2 = cov(x(:,class2samps)')*(length(class2samps)-1);
Sw = Sw1 + Sw2;
SwI = inv(Sw); %inverse of Sw

w0 = SwI*(m1hat - m2hat);
w0proposed = (m1hat - m2hat); %proposed definition for decision plane

%***** STEP 1 *****
%First classify samples based on original Fisher discriminant w0
w0norm = w0/(sqrt(sum(w0.^2))); %normalize w0 vector for visualization purposes
m = mean(x,2);
xshift = x - repmat(m,[1,length(x)]); %Offset data by the mean for classification
```

```
g = w0norm'*xshift;
estclass = (g<0); %g>0 estimates as class 1, g<0 estimates as class 2
estclass1samps = find(~estclass); %indices of samples estimated as class 1
estclass2samps = find(estclass); %indices of samples estimated as class 2
misclass1idx = find(estclass==0 & trueclass==1);
misclass2idx = find(estclass==1 & trueclass==0);

FisherPerr = sum(estclass~=trueclass)/length(trueclass);

%Create plots if 2D data generated
if(n==2)
   %Plot data and estimates
   figure
   hold on
   %plot samples with colors representing true classes.  Circle misclassified
   %samples
   plot(x(1,class1samps),x(2,class1samps),'rx')
   plot(x(1,class2samps),x(2,class2samps),'bx')
   plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
   plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)
   %Plot samples projected onto w0 line
   %Calculate several parameters to help plot
   alphamin = min(g); alphamax = max(g); alpharange = [alphamin,alphamax];
   %Compute a normalized normal vector to the projection plane
   normvec = [1;-w0norm(1)/w0norm(2)]; normvect = normvec/(sqrt(sum(normvec.^2)));
   offsetloc = m + 1.5*max(max(cov1(:)),max(cov2(:)))*normvec;
   %plot line of w0 projection plane
   plot(alpharange*w0norm(1)+offsetloc(1),alpharange*w0norm(2)+offsetloc(2),'k','lineWidth',2)
   %plot decision boundary line (perpendicular to w0 projection plane)
   plot([m(1),offsetloc(1)],[m(2),offsetloc(2)],'k--');
   %Plot samples projected onto w0 line
   plot(g(class1samps)*w0norm(1)+offsetloc(1),g(class1samps)*w0norm(2)+offsetloc(2),'rx')
   plot(g(class2samps)*w0norm(1)+offsetloc(1),g(class2samps)*w0norm(2)+offsetloc(2),'bx')
   plot(g(misclass1idx)*w0norm(1)+offsetloc(1),g(misclass1idx)*w0norm(2)+offsetloc(2),'bo','lineWidth',2)
   plot(g(misclass2idx)*w0norm(1)+offsetloc(1),g(misclass2idx)*w0norm(2)+offsetloc(2),'ro','lineWidth',2)
   title(['Fisher discriminant yeilds Perr = ',num2str(FisherPerr)])
end

%***** STEP 2 *****
%Now classify samples based on proposed discriminant w0 = (m1hat - m2hat);
w0propnorm = w0proposed/(sqrt(sum(w0proposed.^2))); %normalize w0 vector for visualization purposes
g = w0propnorm'*xshift;
estclass = (g<0); %g>0 estimates as class 1, g<0 estimates as class 2
estclass1samps = find(~estclass); %indices of samples estimated as class 1
estclass2samps = find(estclass); %indices of samples estimated as class 2
misclass1idx = find(estclass==0 & trueclass==1);
misclass2idx = find(estclass==1 & trueclass==0);

proposedPerr = sum(estclass~=trueclass)/length(trueclass);

%Create plots if 2D data generated
if(n==2)
   %Plot data and estimates
   figure
   hold on
   %plot samples with colors representing true classes.  Circle misclassified
   %samples
   plot(x(1,class1samps),x(2,class1samps),'rx')
   plot(x(1,class2samps),x(2,class2samps),'bx')
   plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
   plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)

   %Plot samples projected onto w0 line
   %Calculate several parameters to help plot
```

```
    alphamin = min(g); alphamax = max(g); alpharange = [alphamin,alphamax];
    %Compute a normalized normal vector to the projection plane
    normvec = [1;-w0propnorm(1)/w0propnorm(2)]; normvect = normvec/(sqrt(sum(normvec.^2)));
    offsetloc = m + 1.5*max(max(cov1(:)),max(cov2(:)))*normvec;
    %plot line of w0 projection plane
    plot(alpharange*w0propnorm(1)+offsetloc(1),alpharange*w0propnorm(2)+offsetloc(2),'k','lineWidth',2)
    %plot decision boundary line (perpendicular to w0 projection plane)
    plot([m(1),offsetloc(1)],[m(2),offsetloc(2)],'k--');

    %Plot samples projected onto w0 line
    plot(g(class1samps)*w0propnorm(1)+offsetloc(1),g(class1samps)*w0propnorm(2)+offsetloc(2),'rx')
    plot(g(class2samps)*w0propnorm(1)+offsetloc(1),g(class2samps)*w0propnorm(2)+offsetloc(2),'bx')
    plot(g(misclass1idx)*w0propnorm(1)+offsetloc(1),g(misclass1idx)*w0propnorm(2)+offsetloc(2),'bo','lineWidth',2)
    plot(g(misclass2idx)*w0propnorm(1)+offsetloc(1),g(misclass2idx)*w0propnorm(2)+offsetloc(2),'ro','lineWidth',2)
    title(['Proposed discriminant yeilds Perr = ',num2str(proposedPerr)])
end
```

Problem3.m

```
function [ParzenPerr,ParzenPunclass,KNNPerr] = Problem3(x,trueclass,xtrain,traintrueclass,parzenrad,K,ShowTrain-
Plots)
%Problem 3 from ECE662 HW2 (Spring 2008).  Compares the performance of
%Parzen window, K-nearest neighbor and nearest neighbor classifiers.
%Classified training data is input to train the classifiers, while input
%data, x, is classified using each of the three techniques and compared to
%its true classification.  If 2D data is input, plots acompany each
%classifier, plotting the data x, and circling misclassified samples.
%
%xtrain is training data with true class labels traintrueclass.
%x is the input data with true class labels trueclass (for use in analyzing
%the performance of the classifiers

[d,numsamp] = size(x);
[dtrain,numtrainsamp] = size(xtrain);
if(d~=dtrain)
   error('Dimension of input does not match dimension of training data');
end

%Define "quantization bins" for tie breaker
labels = unique(traintrueclass);
trainpriors = zeros(1,length(labels));
for(ii=1:length(labels))
   trainpriors(ii) = sum(traintrueclass==labels(ii));
end
trainpriors = trainpriors/length(traintrueclass);

%Define logical operators
```

```
KISEVEN = round(K/2)==K/2;
KISODD = ~KISEVEN;
TWOCLASSES = length(labels)==2;

%Convert traintrueclass and trueclass to double for use in the mode
%function
traintrueclass = double(traintrueclass);
class1samps = find(~trueclass); %Class 1 labeled as 0 in trueclass
class2samps = find(trueclass); %Class 2 labeled as 1 in trueclass

%Calculate Euclidean distances from each value in our data vector to each of the
%training data points.  We will use this matrix in each of the classifiers
%implemented below
distmat = zeros(numtrainsamp,numsamp);
tempmat = zeros(size(distmat));
for(ii=1:d)
   trainvec = xtrain(ii,:);
   xvec = x(ii,:);
   tempmat = repmat(trainvec',[1,numsamp]) - repmat(xvec,[numtrainsamp,1]);
   distmat = distmat + tempmat.^2;
end
distmat = sqrt(distmat);

%**** Part(a).  Parzen window classifier.  Find how many training samples
%of each class fall within radius parzenrad of each data sample
A = distmat<parzenrad;

%Loop through samples to classify (could probably vectorize somehow, but I
%have decided not to)
parzenclassest = zeros(1,numsamp);
for(ii=1:numsamp)
   aii = A(:,ii);
   idxInWindow = find(aii);
   if(isempty(idxInWindow))
      %No points fall within the radius defining the circular Parzen
      %window.  Classify point as "unknown"
      parzenclassest(ii) = -1;
   else
      %Otherwise, classify as the class with the most training samples
      %within the Parzen window
      [est,freq,multestCell] = mode(traintrueclass(idxInWindow));

      %Check for any "tie votes"
      if(length(multestCell{1})>1)
         %There is a tie.  Break with a random selection between the
         %labels found to tie, with selection driven by the priors for
         %each class (priors calculated from training data).
         bins = trainpriors(multestCell{1}+1);
         cutoffs = cumsum(bins)/sum(bins);
         parzenclassest(ii) = sum(cutoffs<rand);
      else
         parzenclassest(ii) = est;
      end
   end
end
%Analyze Parzen window performance
ParzenPerr = sum(parzenclassest~=trueclass)/length(trueclass);
ParzenPunclass = sum(parzenclassest==-1)/length(trueclass);
ParzenPclassErr = sum((parzenclassest~=-1)&(parzenclassest~=trueclass))/length(trueclass);

if(d==2 && TWOCLASSES)
   misclass1idx = find(parzenclassest==0 & trueclass==1);
   misclass2idx = find(parzenclassest==1 & trueclass==0);
   noclassidx = find(parzenclassest==-1);
```

```
    %Plot data and estimates
    figure
    hold on
    %plot samples with colors representing true classes.  Circle misclassified
    %samples with the same color as their true class.  Circle unclassified
    %samples (those with no training samples within the parzen window) in
    %black.
    plot(x(1,class1samps),x(2,class1samps),'rx')
    plot(x(1,class2samps),x(2,class2samps),'bx')
    plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
    plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)
    plot(x(1,noclassidx),x(2,noclassidx),'ko','lineWidth',2)
    title(['Parzen window classifier yeilds Perr = ',num2str(ParzenPerr),...
          ' with ',num2str(ParzenPunclass),' being unclassified'])

    if(ShowTrainPlots)
       %Plot the misclassified points along with training samples
       %Extract training sample labels
       trainclass1samps = find(~traintrueclass); %Class 1 labeled as 0 in trueclass
       trainclass2samps = find(traintrueclass); %Class 2 labeled as 1 in trueclass
       figure
       hold on
       %Plot the misclassified points along with training samples
       plot(xtrain(1,trainclass1samps),xtrain(2,trainclass1samps),'rx')
       plot(xtrain(1,trainclass2samps),xtrain(2,trainclass2samps),'bx')
       plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
       plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)
       plot(x(1,noclassidx),x(2,noclassidx),'ko','lineWidth',2)
    end
end

%**** Part(b).  KNN classifier.  Count how many of each class is
%represented in the K nearest training samples to each of the data samples.
% Use the "majority vote" to classify each sample.
if(K>numtrainsamp)
    error('K exceeds number of training samples');
end

%Sort the distance matrix
[distmatSort,sortIdxMat] = sort(distmat);

%Classify data using KNN approach

if(KISODD & TWOCLASSES)
    %If K is odd and there are precisely two classes, there is no risk of
    %tie votes.  Take the estimate as the "majority vote" (mode) of the K
    %nearest training samples.
    KNNclassest = mode(traintrueclass(sortIdxMat(1:K,:)),1);
else
    %We have a risk of a "tie vote."  Use mode command to find candidate
    %labels, then check for any "ties."
    [est,freq,multestCell] = mode(traintrueclass(sortIdxMat(1:K,:)));
    for(ii=1:size(x,2))
       %Check for any "tie votes"
       if(length(multestCell{ii})>1)
          %There is a tie.  Break with a random selection between the
          %labels found to tie, with selection driven by the priors for
          %each class (priors calculated from training data).
          bins = trainpriors(multestCell{ii}+1);
          cutoffs = cumsum(bins)/sum(bins);
          KNNclassest(ii) = sum(cutoffs<rand);
       else
          KNNclassest(ii) = est(ii);
       end
```

```
    end
end
%Analyze KNN performance
KNNPerr = sum(KNNclassest~=trueclass)/length(trueclass);

if(d==2 && TWOCLASSES)
    misclass1idx = find(KNNclassest==0 & trueclass==1);
    misclass2idx = find(KNNclassest==1 & trueclass==0);
    %Plot data and estimates
    figure
    hold on
    %plot samples with colors representing true classes.  Circle misclassified
    %samples with the same color as their true class.
    plot(x(1,class1samps),x(2,class1samps),'rx')
    plot(x(1,class2samps),x(2,class2samps),'bx')
    plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
    plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)
    title(['K nearest neighbor classifier yeilds Perr = ',num2str(KNNPerr)])

    if(ShowTrainPlots)
        %Plot the misclassified points along with training samples
        %Extract training sample labels
        figure
        hold on
        %Plot the misclassified points along with training samples
        plot(xtrain(1,trainclass1samps),xtrain(2,trainclass1samps),'rx')
        plot(xtrain(1,trainclass2samps),xtrain(2,trainclass2samps),'bx')
        plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
        plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)
    end
end
%
%**** Part(c).  Nearest neighbor classifier.  Classify each data sample as
%the class of the nearest training sample.  This is a special case of KNN
%with K = 1.
NNclassest = traintrueclass(sortIdxMat(1,:));
NNPerr = sum(NNclassest~=trueclass)/length(trueclass);

if(d==2 && TWOCLASSES)
    misclass1idx = find(NNclassest==0 & trueclass==1);
    misclass2idx = find(NNclassest==1 & trueclass==0);
    %Plot data and estimates
    figure
    hold on
    %plot samples with colors representing true classes.  Circle misclassified
    %samples with the same color as their true class.
    plot(x(1,class1samps),x(2,class1samps),'rx')
    plot(x(1,class2samps),x(2,class2samps),'bx')
    plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
    plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)
    title(['Nearest neighbor classifier yeilds Perr = ',num2str(NNPerr)])

    if(ShowTrainPlots)
        %Plot the misclassified points along with training samples
        %Extract training sample labels
        figure
        hold on
        %Plot the misclassified points along with training samples
        plot(xtrain(1,trainclass1samps),xtrain(2,trainclass1samps),'rx')
        plot(xtrain(1,trainclass2samps),xtrain(2,trainclass2samps),'bx')
        plot(x(1,misclass1idx),x(2,misclass1idx),'bo','lineWidth',2)
        plot(x(1,misclass2idx),x(2,misclass2idx),'ro','lineWidth',2)
    end
end
```

genMultClassSamps.m

```
function [x,trueclass] = genMultClassSamps(means,covs,priors,numsamp)
%Utility function to generate mutlidimensional Gaussian data of various
%classes.  If dimension of data is d, and number of classes is N, then
%input means is dxN, input covs is dxdxN, priors is 1xN.  Numsamp specifies
%how many samples of data to generate.  Output x is dxnumsamp, trueclass is
%1xnumsamp, taking on integer values in [0,N-1]

%Check dimension of means input
[d,numclass] = size(means);
%Check dimension of covs input
[d1,d2,d3] = size(covs);
if(d1~=d2)
   error('Covariance matrices must be square');
elseif(d1~=d)
   error
elseif(d3~=numclass)
   error('Number of class means does not match number of class covariance matrices');
end
%Check dimension of priors input
[d1,d2] = size(priors);
if(d1~=1)
   error('Please enter a 1xnumclass vector for priors');
elseif(d2~=numclass)
   error('Number of class means does not match number of class priors');
end
%Normalize priors vector to sum to unity
priors = priors/sum(priors);
%Use priors vector to define cutoffs
cutoffs = cumsum(priors)';
cutoffmat = repmat(cutoffs,[1,numsamp]);
rmat = repmat(rand(1,numsamp),[numclass,1]); %random vector repeated to choose class labels
trueclass = sum(cutoffmat<rmat); %1xnumsamp vector taking on values from 0 to numclass-1;

x = randn(d,numsamp);
for(class=0:numclass-1)
   idx = class+1;
   classsamps = find(trueclass==class);
   x(:,classsamps) = repmat(means(:,idx),[1,size(classsamps,2)]) +...
                  chol(covs(:,:,idx))*x(:,classsamps);
end
```

visualizeClassEst.m

```
function visualizeANNresults(x,trueclass,estclass)
%Utility function to visualize classified data with a known true
%classification.  Various classes (up to 7) are distinguished by color and
%misclassified samples are circled in their own color

colorstr = 'rbgkcmy';
figure
hold on
ii = 1;
for(class=unique(trueclass))
    classidx = find(trueclass==class);
    plot(x(1,classidx),x(2,classidx),[colorstr(ii),'x'])
    misclassidx = find(trueclass==class & estclass~=class);
    plot(x(1,misclassidx),x(2,misclassidx),[colorstr(ii),'o'],'Linewidth',2)
    ii=ii+1;
end
```