Question1:

Fisher linear discriminant analysis (FDA) is one of the common parametric methods. It looks for directions that are efficient for discrimination. Projection onto one direction w, two-class problem is defined below:

(1) Samples: *n d*-dimensional vectors $x_1 \ldots\ldots x_n$, consisting of two subsets D_1, D_2

(2) Projected samples: $y = \omega^T x$, two subsets Y_1, Y_2

(3) Criterion: maximize the Fisher linear discriminant

$$J(\vec{\omega}) = \frac{\left|\tilde{m}_1 - \tilde{m}_2\right|^2}{\tilde{S}_1^{\,2} + \tilde{S}_2^{\,2}}$$

where $\tilde{m}_1 = \text{mean of } y \in Y_1, \tilde{m}_2 = \text{mean of } y \in Y_2, \tilde{S}_1^{\,2} = \text{scatter}$

$$\text{of } Y_1 = \sum_{y \in Y_1} (y - \tilde{m}_1)^2, \tilde{S}_2^{\,2} = \text{scatter of } Y_2 = \sum_{y \in Y_2} (y - \tilde{m}_2)^2$$

$$J(\vec{\omega}) = \frac{\left|\tilde{m}_1 - \tilde{m}_2\right|^2}{\tilde{S}_1^{\,2} + \tilde{S}_2^{\,2}} = \frac{\vec{\omega}^T S_B \vec{\omega}}{\vec{\omega}^T S_W \vec{\omega}}$$

$S_B = (\vec{m}_1 - \vec{m}_2)(\vec{m}_1^{\,T} - \vec{m}_2^{\,T})$ is the between-class scatter matrix

, and $S_w = S_1^{\,2} + S_2^{\,2}$ is the within-class scatter matrix, where $S_i = \sum_{x \in D_i} (x - \vec{m}_i)(x - \vec{m}_i)^t$

Maximizing the Fisher linear discriminant—the linear function generating the maximum ratio of between-class scatter to within-class scatter, yields the optimal projection (a projection that generates large separation between the projected means while reducing the scatter of the projected data). $\vec{\omega}^* = S_w^{-1}(\vec{m}_1 - \vec{m}_2)$

The main purpose of this question is to investigate the performance of the classifier using $\vec{\omega}^* = S_w^{-1}(\vec{m}_1 - \vec{m}_2)$ as the projection versus using $\vec{\omega}_I = (\vec{m}_1 - \vec{m}_2)$ (setting S_w as the identity matrix). The following steps were followed for programming:

(1) Produced correlated Gaussian data sets from class 1 and class 2. Data from class 1 and class 2 has the different means ($\vec{m}_2 = \vec{m}_1 + \Delta$ to generate separation between the classes. The mean of class is set as $\vec{m}_1 = \left\{\frac{1}{2}, 1, \frac{3}{2}, 2 \ldots, N/2\right\}$, but the covariance is the same (which means $\Sigma_1 = \Sigma_2$).

(2) Compute the optimal projection $\vec{\omega}^* = S_w^{-1}(\vec{m}_1 - \vec{m}_2)$ by using above equation and criteria.
(3) Apply the projection to the data: $\vec{\omega}^* \vec{y} = \vec{b}$.
(4) Find the threshold to classify the data: $\omega_0 = \frac{\tilde{m}_1 - \tilde{m}_2}{2}$ for Gaussian data from two classes with equal covariance matrices.
(5) Classify the i^{th} data point as from class 1 if $b_i > \omega_0$, else from class 2. Count number of true and false classifications.
(6) Apply the same procedure for $\vec{\omega}_I = (\vec{m}_1 - \vec{m}_2)$ on same data.
(7) Compare the result of classification using $\vec{\omega}^*$ versus using $\vec{\omega}_I$.

Three experiments were applied to evaluate the result of the two different methods ($\vec{\omega}^*$ versus $\vec{\omega}_I$): (1) performance as a function of separation between two classes, (2) performance as a function of feature vector dimension size, and (3) performance as a function of number of samples.

Experiment 1: Separation

From the above step, $\vec{m}_2 = \vec{m}_1 + \Delta$ has been set. To evaluate performance as a function of separation between classes, Δ was varied from 0 to 5 with interval 0.05, while the feature vector size is kept at N = 5 and the number of samples for each class is 10000. Figure 1 shows the performance as a function of data separation. $\vec{\omega}^*$ generates more correct classifications than using $\vec{\omega}_I$ (setting S_w as the identity matrix) at all separations except the situation when there is no separation between the means (Δ=0). As to be observed, at very small (say below 0.5) and very large separations (say above 4.5), the performance of both methods are somewhat similar – but at intermediate separations (say between 1.5 and 3), method $\vec{\omega}^*$ is better in accuracy to method $\vec{\omega}_I$ by around 10%. It can be expected, as when the data has no separation, no method will achieve more than chance accuracy, no matter how good the algorithm applies. And when there is a very large separation, any good method should produce great classification.

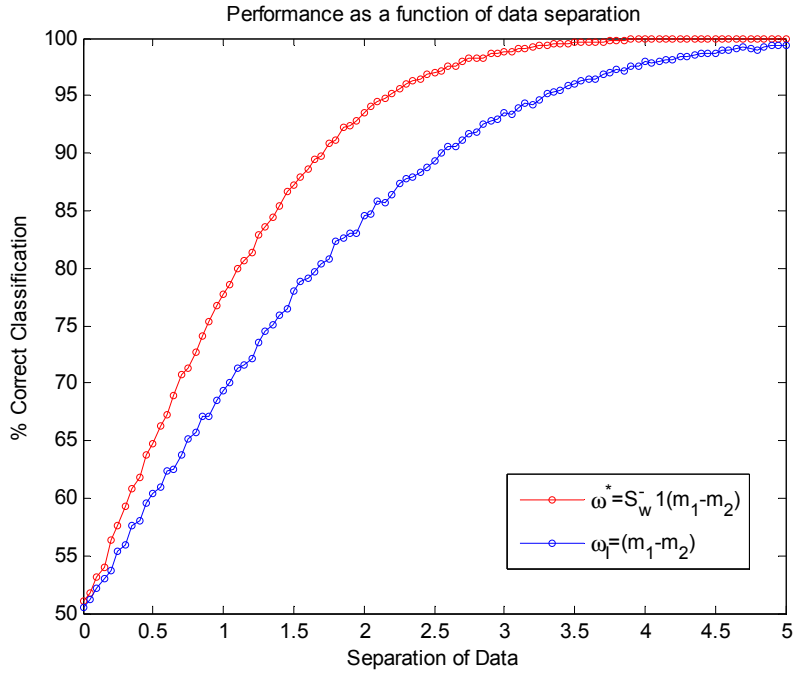

Figure 1: Performance as a function of data separation

Experiment 2: Feature vector dimension size (N)

Feature vector dimension size N was varied from 2 up to 100, while the separation is set at $\Delta = 1.5$ and the number of samples for each class is 10000. Figure 2 displays the results, showing that the performance of $\vec{\omega}^*$ is better to that of method $\vec{\omega}_I$ when the dimension of feature vector goes higher. The difference of the accuracy is up to 10% when the dimension of the feature vector is above 25. Also, an increase in dimension size when N is small ([2:2:40]) results in a big increase in accuracy for both algorithms. However, an additional increase in dimension size when N is large ([50:10:100]) results in insignificant improvement in accuracy for both algorithms

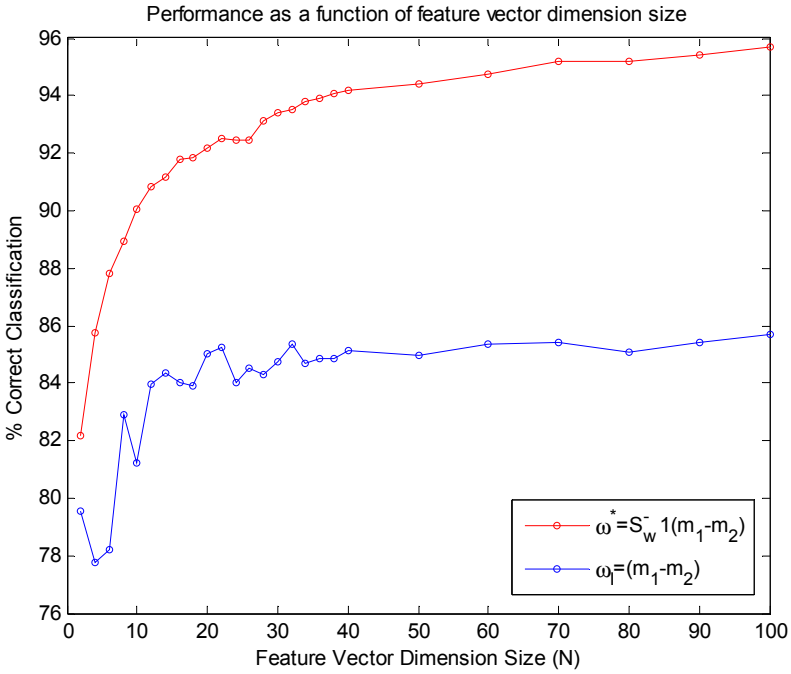

Figure 2: Performance as a function of feature vector dimension size (N)

Experiment 3: Data sample size (M)

To evaluate performance as a function of data sample size, number of samples M for each class was different (from 100 to 10000), while the separation is set at Δ =1.5, and the feature vector dimension size is 5. Figure 3 illustrates the results. As to be observed, while method $\vec{\omega}^*$ outtakes $\vec{\omega}_l$ at all sample sizes, neither method presented an increase in accuracy as sample size grows.

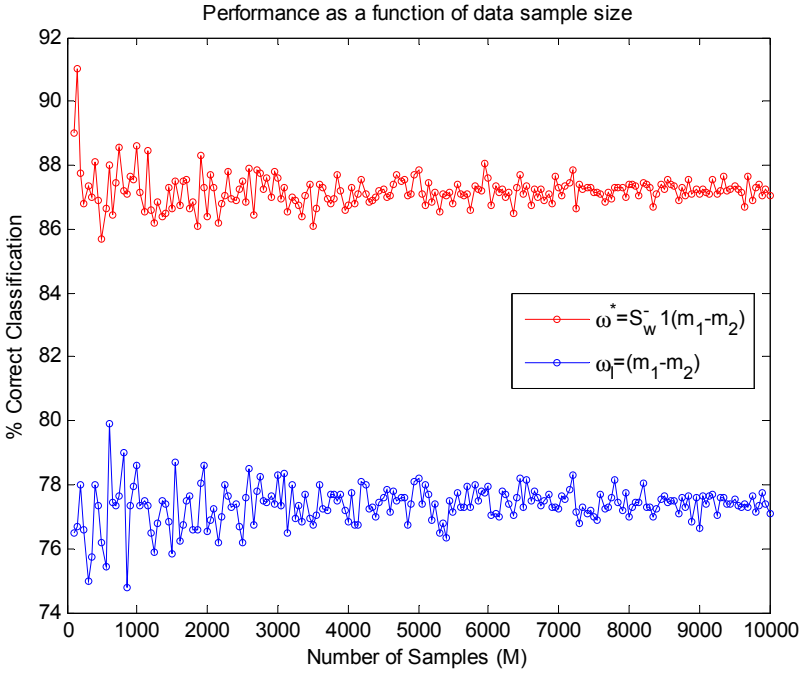

Figure 3: Performance as a function of data sample size (M)

Discussion of Results

In all the simulations discussed above, method $\vec{\omega}^*$ accomplished better performance over $\vec{\omega}_I$. While the simulations by no means cover all possible cases, the results do show a large degree of certainty that $\vec{\omega}^*$ is better. From the Fisher discriminant

$J(\vec{\omega}) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{\tilde{S}_1^{\,2} + \tilde{S}_2^{\,2}} = \frac{\vec{\omega}^T S_B \vec{\omega}}{\vec{\omega}^T S_W \vec{\omega}}$. Maximizing $J(\vec{\omega})$ finds the projection $\vec{\omega}^*$ that increases the separation between the means of the projected data as well as reducing the within-class scatter of the data to produce minimal overlap between classes. On the contrary, the projection $\vec{\omega}_I = (\vec{m}_1 - \vec{m}_2)$ does not account for within-class scatter – resulting in a sub-optimal projection that while separating the means of the projected data, does not look for optimizing the variance/scatter of the projected data. Thus, method $\vec{\omega}^*$ can be predicated to accomplish better performance than method $\vec{\omega}_I$.

Question2:

Neural Network:

Design a classifier using the neural network approach versus the support vector machine. The neural network method used in this report is part of the Neural Network Toolbox in Matlab (The Mathworks). Particularly, the following functionalities were used for neural network classification:

(1) *net = newpnn(P, T, Sread)*: Implements a probabilistic neural network (PNN; a kind of radial basis network suitable for classification problems) that is suitable for classification. The *newpnn* function creates a two layer network by calling on a variety of functions from the Neural Network Toolbox. The first layer has *radbas( )* neurons, and calculates the weighted inputs with *dist( )* and its net input with *netprod( )*. The second layer has *compet( )* neurons, and calculates its weighted input with *dotprod( )* and its net inputs with *netsum( )*. The main function *newpnn( )* accepts an input matrix *P* of input vectors and an input matrix *T* of target class vectors, and returns a new probabilistic network.

(2) *a = sim(net,P):* simulate a Simulink model. It uses the new neural network designed by *newpnn( )* to classify data matrix *P*.

To evaluate the performance of this probabilistic neural network, a set of training data was used to design a new neural data, and a separate set testing data was used to compute the accuracy of the algorithm.

Support Vector Machine:

The support vector machine (SVM) algorithm utilized in this report is authored by Mangasarian and Musicant in the department of computer sciences at the University of Wisconsin. The algorithm is named Lagrangian Support Vector Machines (LSVM) and is

freely available online at http://www.cs.wisc.edu/dmi/lsvm/. The technical report detailing LSVM is also available online from ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-06.ps. Mangasarian and Musicant developed an implicit Lagrangian for the dual of a simple reformulation of the standard quadratic program of a linear support vector machine, leading to the minimization of an unconstrained differentiable convex function with dimensionality equal to number of classified points. The minimization problem is solved by a linearly convergent Lagrangian support vector machine algorithm, requiring the inversion at the outset of a single matrix with the order of dimensionality equal to the original input space plus one. Classification was performed by training the LSVM – its accuracy was assessed with a separate set of testing data.

Three experiments were applied to evaluate and compare the performance of PNN and LSVM: (1) performance as a function of separation between two classes, (2) performance as a function of feature vector dimension size, and (3) performance as a function of number of samples.

Experiment 1: Separation

Recall that $\vec{m}_2 = \vec{m}_1 + \Delta$. To evaluate performance as a function of separation between classes, Δ was varied from 0 to 5 ([0:0.05:5]), while the feature vector size is kept at N = 5 and the number of samples for each class is 2500. Half the sample from each class was used as training data, and the other half utilized as testing data. Figure 4 shows the results of this simulation. As to be observed from Figure 4, LSVM achieves better accuracy with small separations (< 1.2), PNN overtakes LSVM in accuracy at approximately $\Delta = 1.2$. While PNN almost accomplishes perfect accuracy at $\Delta = 4$, LSVM only has an accuracy of less than 90%.

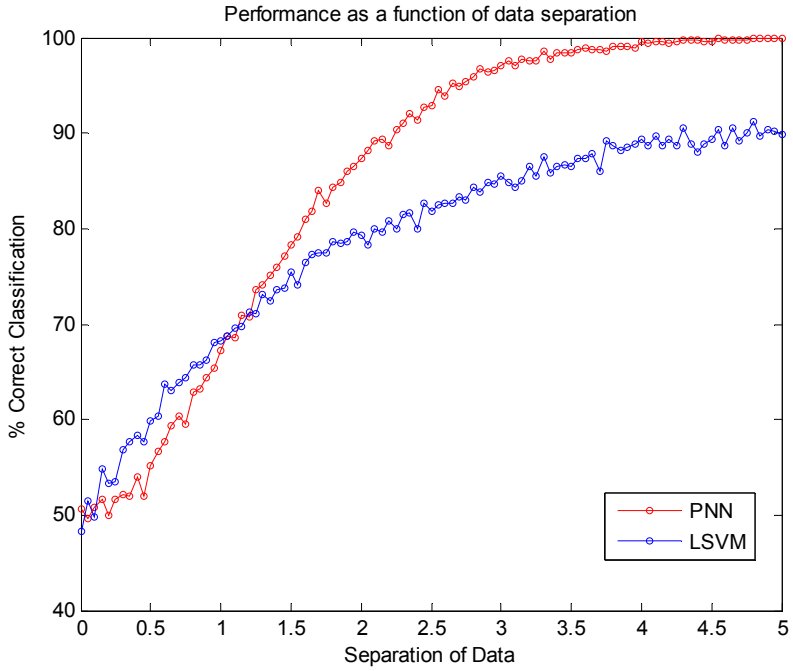

Figure 4: Performance as a function of data separation

Experiment 2: Feature vector dimension size (N)

In this experiment, feature vector dimension size N was varied from 1 to 40, while the separation is set at $\Delta = 1.5$ and the number of samples for each class is 2500 (half used as training data, remaining half used as testing data). Figure 5 is the results, showing that while PNN achieves better performance at low dimension sizes (< 12), but it fails at higher dimension sizes and only accomplishes chance accuracy. LSVM, on the contrary, gradually improves in accuracy as feature vector dimension size goes higher.

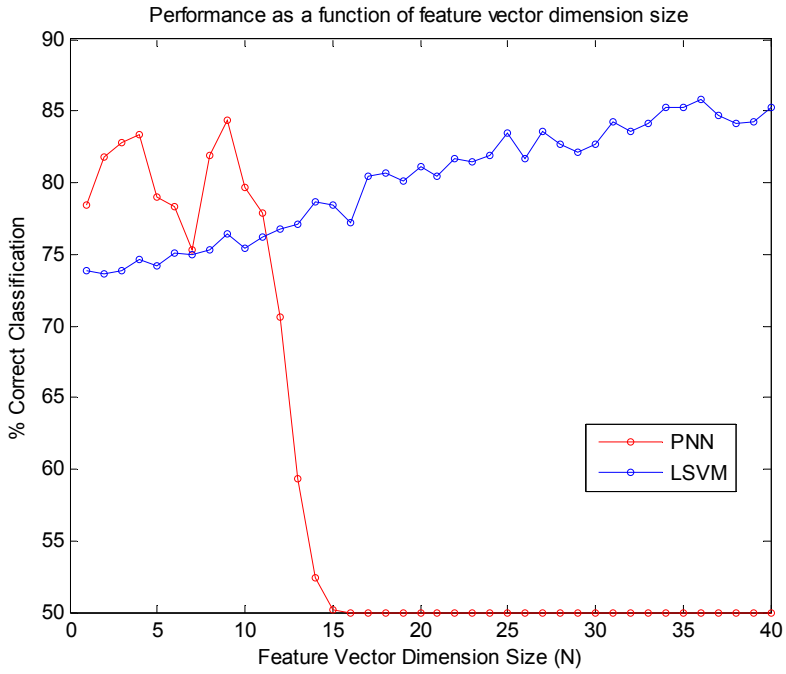

Figure 5: Performance as a function of feature vector dimension size (N)

Experiment 3: Data sample size (M)

In this experiment, the number of samples M for each class was varied from 100 to 2500 while the separation is set at $\Delta = 1.5$ and the feature vector dimension size (N) is 4. Half of the samples (M/2) were used for training and the rest for testing. Figure 6 shows that PNN consistently achieves better accuracy than LSVM at the given separation and dimension size 4, though increasing sample size past around 1000 does not increase accuracy for either method. Figure 7 shows the results, showing that PNN achieves better accuracy than LSVM in most of the given separation and dimension size 5, though increasing sample size past around 1000 does not increase accuracy for either method. It is consistent with the observation in experiment 2 which indicates LSVM gradually improves in accuracy as feature vector dimension size goes higher while PNN performs worse accuracy in higher feature vector dimension.

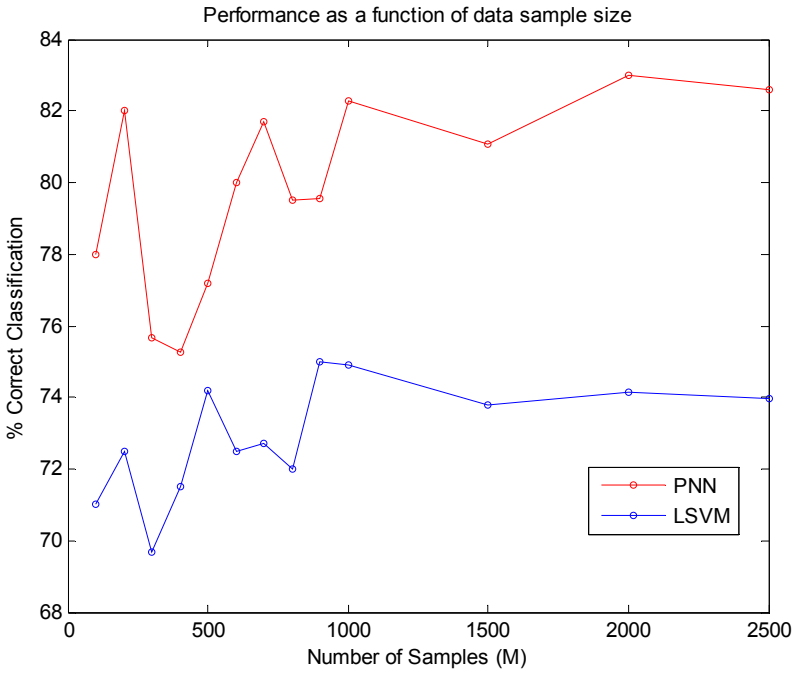

Figure 6: Performance as a function of data sample size (M) with N=4

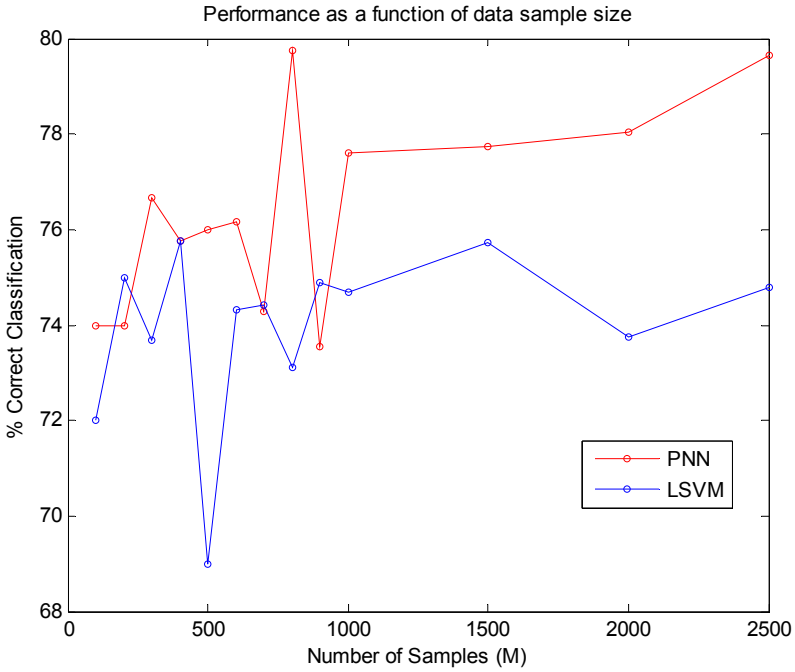

Figure 7: Performance as a function of data sample size (M) with N=5

Experiment 4: Distribution

To evaluate performance of LSVM and PNN for data with different distributions, Gaussian, Exponential, and Uniform random data were generated. Number of samples M for each class was 2500, the separation of the means was set at $\Delta = 1.5$, and feature vector dimension size was 5. Table 1 showed that PNN achieved better accuracy for the Gaussian and exponential data. Both methods achieved chance accuracy for the uniformly distributed data.

	Gaussian	Exponential	Uniform
PNN	77.44%	74.80%	48.16%
LSVM	74.24%	73.20%	48.88%

Table 1: Accuracy of PNN and LSVM with different distributions with N=5

From the simulations conducted above, both PNN and LSVM are shown to have some strengths and weaknesses. PNN outperforms LSVM when the separation is large and feature vector dimension size is small. PNN fails with large feature vector dimension sizes, while LSVM shows continued improvement in accuracy with increasingly large dimension sizes. PNN and LSVM are two specific algorithms examined here, so the conclusion should not be generalized to all support vector machines and neural networks. Accuracy is expected to vary depending on the robustness of each algorithm.

Question 3

This question investigates applying the methods of Parzen windows, K-nearest neighbors, and nearest neighbor to classification.

Parzen Windows

Parzen windows classification is a technique for nonparametric density estimation, which can also be used for classification. Using a given window function, this technique approximates the distribution of a given training set using a linear combination of window centered at a observed point $\vec{x}_0$ to compute and sum the contribution from each point of the training data set. Applied to classification, a test point is labeled by using the window function to compute a weighted mean of the contribution from the training points in each of the classes. The test point is labeled to be from the class with the maximum weighted mean. While the choice of a window function is important, the choice of a sensible window side-length is crucial to accurate density estimation and classification. A small window length may lead to sharp behavior in the density estimate while an excessive large window length may average out the details of the data's underlying distribution. To simulate the effect of window length on the density estimate, the following steps were taken:

- Generate M=5000 samples from a Uniform distribution $X \sim U(0,\ 1)$.
- Use a given Uniform window of length h
- Visualize the density estimate as a function of h and x using mesh( )
- Compute the mean squared error (MSE) of the density estimate for each window size h using the known underlying Uniform distribution
- Repeat procedure with data from a univariate Gaussian distribution $X \sim N(0.5,\ 1)$.

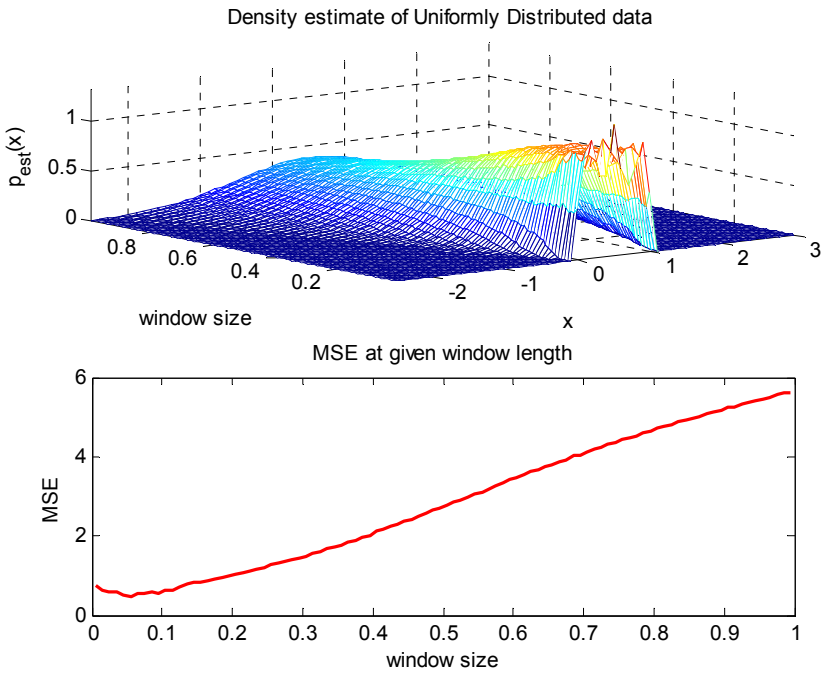

Figure 8: Density estimate as a function of window length for Uniform data

Figure 8 shows the density estimate of a data set generated from the uniform distribution. As can be observed, with a large window size, the density estimate starts to appear Gaussian (partly because the window function is Gaussian). The minimum MSE occurs when h=0.065. If the underlying distribution is known, perhaps a window function of the shape of that particular probability density function with an appropriate window size would generate the best density estimate. However, in reality, the underlying distribution is complex and barely known, and thus a generic Gaussian window function is generally used.

As can be observed from Figure 9, small window length results in sharp behavior of the density estimate while a large window size produces excessive averaging. The subplot with the MSE in fact confirms that optimal density estimation for this Gaussian data is achieved when h=0.295.

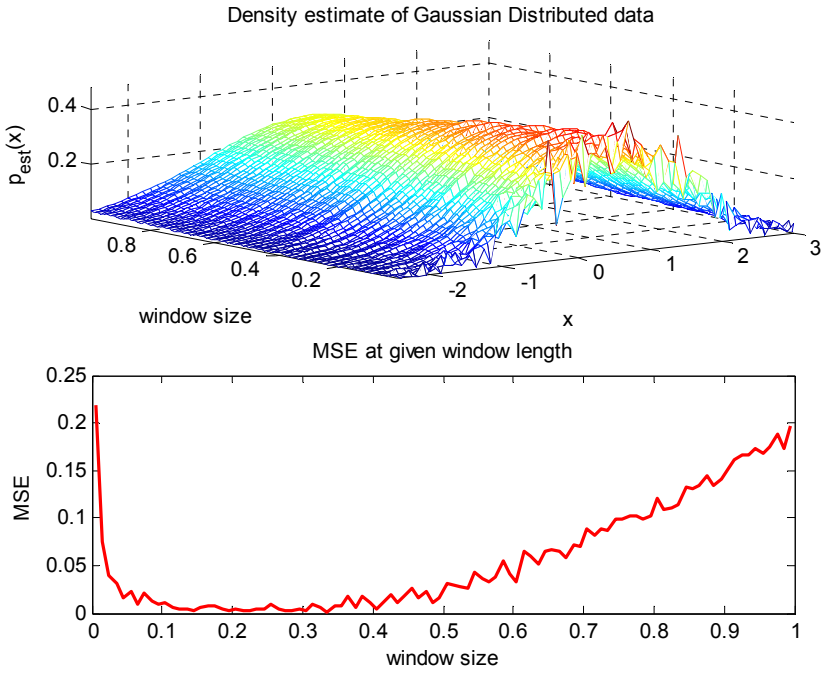

Figure 9: Density estimate as a function of window length for Gaussian distributed data

Nearest Neighbor and K-Nearest Neighbor

Nearest neighbor and K-nearest neighbor are simple methods used for classification. Using a set of training data, the nearest neighbor approach classifies a test point to be from the class of the nearest training data point. The K-nearest neighbor approach examines K training data neighbors surrounding a test point, and classifies the test point to be from the class who has more points closer to the testing point. Figure 10 shows an example of the nearest neighbor rule. The circle represents the unknown testing sample and its nearest neighbor comes from the red class, it is labeled as red class.

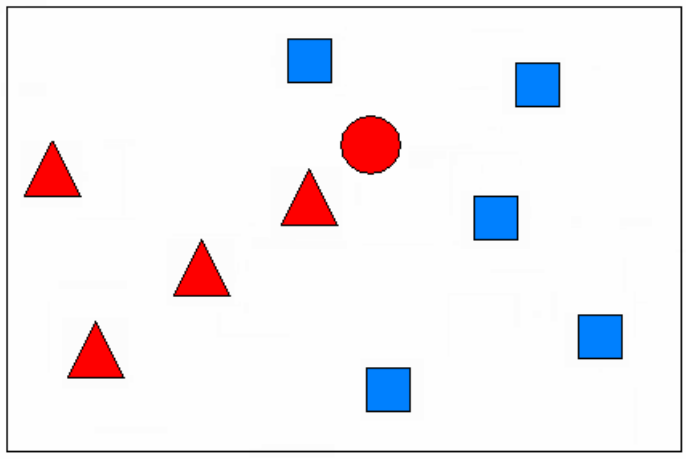

Figure 10: The NN rule

Figure 11 illustrates the K-nearest neighbor rule. Of the three training points closest to the circle testing point, two are from the blue class and one from the red class, therefore the testing point is labeled to be from the blue class.

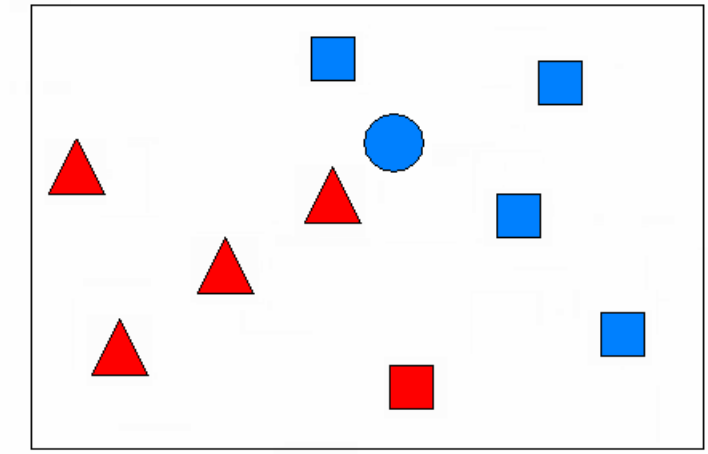

Figure 11: The KNN rule with k=3

As before, to evaluate and compare the performance of Parzen windows, nearest neighbors and K-nearest neighbors, several simulations were performed: (1) performance as a function of separation between classes, (2) performance as a function of feature vector dimension size, and (3) performance as a function of number of samples.

Experiment 1: Separation

Recall that $\bar{m}_2 = \bar{m}_1 + \Delta$. To evaluate performance as a function of separation between classes, Δ was varied from 0 to 5, while the feature vector size is kept at N = 5 and the number of samples for each class is 1000. The window side-length for Parzen windows is h = 0.3. The number of neighbors for K-nearest neighbors is set at K = 7.

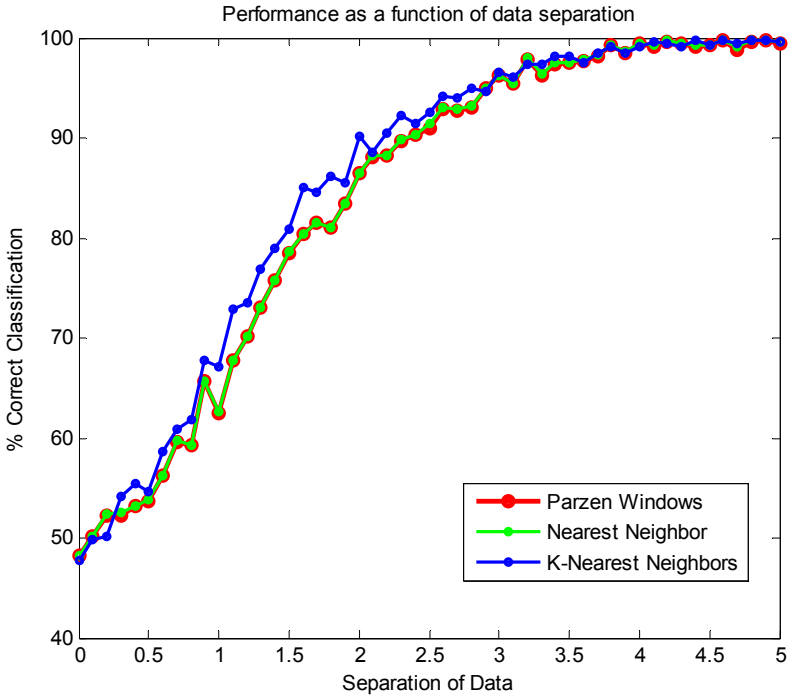

Figure 12: Performance as a function of data separation

From Figure 12, as to be observed, the curve for Parzen windows almost completely overlaps the curve for nearest neighbors. K-nearest neighbor performances continually better accuracy at all separations, though the improvement is not very significant.

Experiment 2: Feature vector dimension size (N)

Feature vector dimension size N was varied from 1 to 40 to evaluate performance as a function of separation between classes, while the separation is set at $\Delta = 1.5$ and the number of samples for each class is 1000 (half used as training data, the other half used as testing data). The window side-length for Parzen windows is h = 0.3. The number of neighbors for K-nearest neighbors is set at K = 7. Repeatedly, K-nearest neighbors has continually higher accuracy as feature vector dimension size goes higher. As to be observed form Figure 13, increasing the dimension size does not leads to a significant increase in accuracy for either nearest neighbor or K-nearest neighbor. On the contrary, increasing the dimension size makes a decrease in accuracy for the Parzen windows methods at N > 10, accuracy declines to zero percent. This can be explained by looking at the Gaussian window function:

$$\varphi(\vec{u}) = \frac{1}{(2\pi)^{N/2}} e^{\frac{-\|\vec{u}\|^2}{2}}$$

If N is very large, the constant in front of the exponential becomes too small to be represented as a type *double* in Matlab – it is rounded to zero. Therefore $\varphi(\vec{u})$ from class 1 is equal to $\varphi(\vec{u})$ from class 2 which means all are equal to zero. The script is coded to not assign a class when this occurs, resulting in zero accuracy. This suggests a crucial point when working with

Parzen windows: with very large feature vector dimension sizes, rounding error becomes a key issue.

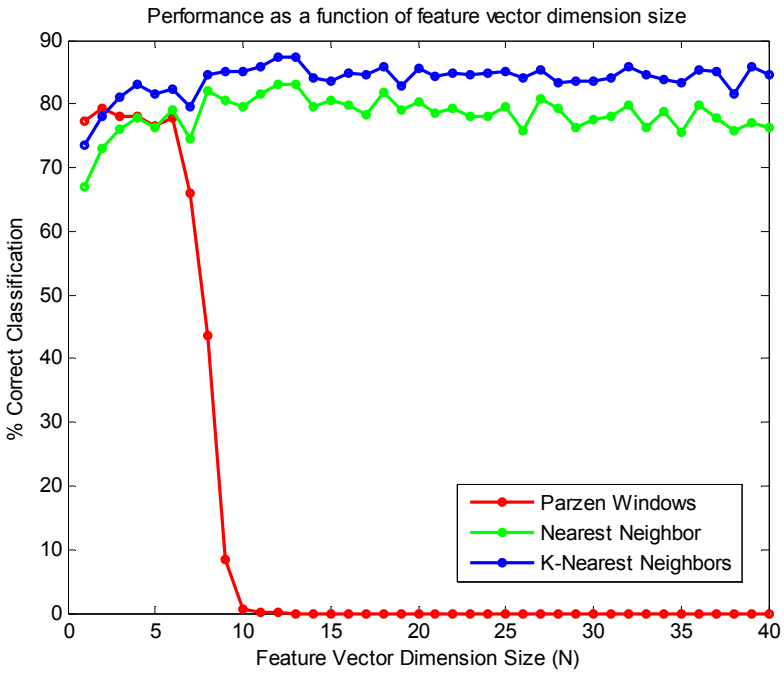

Figure 13: Performance as a function of feature vector dimension size (N)

Experiment 3: Data sample size (M)

To evaluate performance as a function of data sample size, the number of samples M for each class was varied from 100 to 2500, while the separation is set at $\Delta = 1.5$ and the feature vector dimension size is 5. Note that half of the samples (M/2) is used for training and the rest for testing. The window side-length for Parzen windows is $h = 0.3$. The number of neighbors for K-nearest neighbors is set at $K = 7$. Again, K-nearest neighbors performs better accuracy than the two other methods. The accuracy curve for Parzen windows almost completely overlaps the curve for nearest neighbors. As to be observed from Figure 14 below, increasing sample size does not appear to increase performance for any of these three methods.

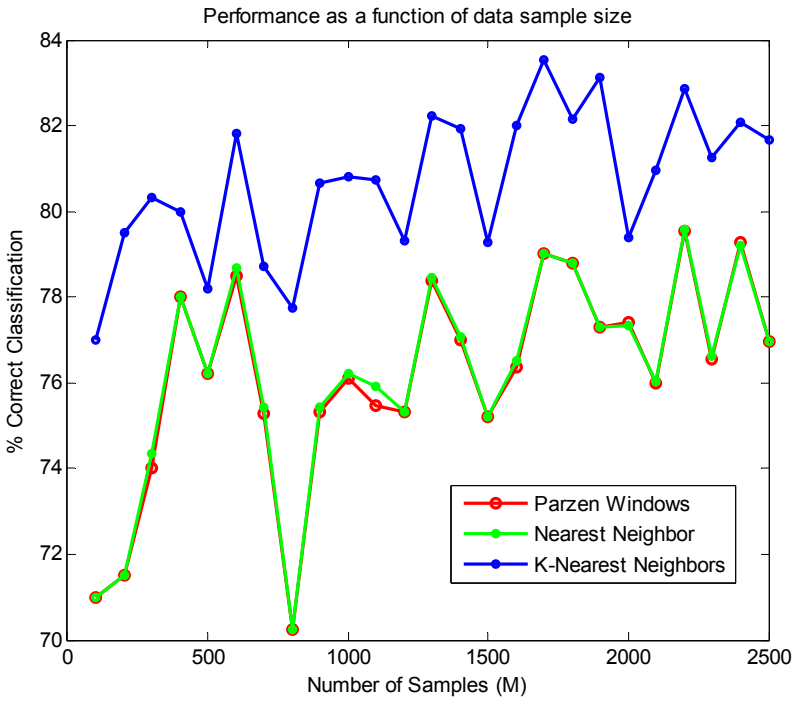

Figure 14: Performance as a function of data sample size (M)

Discussion of Results

For the experiments conducted to examine Parzen windows, nearest neighbor, and K-nearest neighbors, K-nearest neighbors has been proved to have the highest accuracy. Parzen window's performance is almost exactly the same as that of nearest neighbor in almost all cases examined here. Though this may be the case here, it is expected that window size and window length will significantly change the performance of Parzen windows classification depending on the circumstance. Similarly, the value for K may also be expected to change the accuracy depending on the situation.

```
% Question1
clc; close all;
clear all;
difvec=0:0.05:5;
% Nvec=[2:2:40 50:10:100];
% Mvec=[100:50:10000];
ite=length(difvec);
% ite=length(Nvec);
% ite=length(Mvec);

C1=zeros(1,ite);
C2=C1;
for iii=1:ite
    dif=difvec(iii); N=5; M=10000;
    %N=Nvec(iii); dif=1.5; M=10000;
    %M=Mvec(iii); dif=1.5; N=5;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % USER DEFINED MEAN AND VARIANCE %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % DATA IS IID HERE, MUST CORRELATE LATER
    mu1=[1:N]/2;
    sigma1=sqrt([1:N]);
    % Separate data by adjusting mean and variance
    mu2=mu1+dif;
    sigma2=sigma1;

    % diagonal of covariance of X from class 1
    covX1=diag([sigma1.^2]');
    % diagonal of covariance of X from class 2
    covX2=diag([sigma2.^2]');

    x1=zeros(M,N); x2=zeros(M,N);
    for ii=1:N
        x1(:,ii)=random('normal',mu1(ii),sigma1(ii),[M 1]);
        x2(:,ii)=random('normal',mu2(ii),sigma2(ii),[M 1]);
    end

    % GENERATING POSITIVE DEFINITE MATRIX TO CORRELATE DATA
    c=zeros(1,N); c(1)=1;
    r=ones(1,N);
```

```
    P=toeplitz(c,r);
    Porth=orth(P);      % orthogonalize P
    D=diag([1:N]);      % eigenvalues along the diagonal
    E=inv(Porth)*D*Porth;

    % generating correlated feature vectors
    y1=x1*E;
    y2=x2*E;
    covY1=E*covX1*E';
    covY2=E*covX2*E';
    % Calculating means of Y1, Y2, correlated data for classes 1 and 2
    muY1=mean(y1);
    muY2=mean(y2);
    Sw=(y1-repmat(muY1,M,1))'*(y1-repmat(muY1,M,1))+(y2-
repmat(muY2,M,1))'*(y2-repmat(muY2,M,1));

    %%%%%%%%%%
    % with Sw %
    %%%%%%%%%%

    w=inv(Sw)*(([muY1-muY2])');
    w_s=w;

    y=[y1;y2];
    %b=repmat(w',2*M,1).*y;
    b=y*w;

    % For univariate Gaussian with same E, threshold w0 is midway between
    % the two projected means
    w0=(w'*muY1'+w'*muY2')/2;

    ind1=find(b(1:M)>w0);        % class 1
    ind2=find(b(M+1:end)<w0);    % class 2

    TC=length(ind1)+length(ind2);   % true classifications
    FC=2*M-TC;

    TD_FD_inv=[TC FC]
    C1(iii)=[TC/2/M*100];
```

```
    %%%%%%%%%%%%%
    % without Sw %
    %%%%%%%%%%%%%
    Sw=eye(N);
    w=inv(Sw)*(([muY1-muY2])');

    y=[y1;y2];
    %b=repmat(w',2*M,1).*y;
    b=y*w;

    % For univariate Gaussian with same E, threshold w0 is midway between
    % the two projected means
    w0=(w'*muY1'+w'*muY2')/2;

    ind1=find(b(1:M)>w0);        % class 1
    ind2=find(b(M+1:end)<w0);    % class 2

    TC=length(ind1)+length(ind2);   % true classifications
    FC=2*M-TC;

    TD_FD_iden=[TC FC]
    C2(iii)=[TC/2/M*100];
end

figure
plot(difvec,C1,'ro-','linewidth',1,'MarkerSize',4); hold on
plot(difvec,C2,'bo-','linewidth',1,'MarkerSize',4);
ylabel('% Correct Classification');
xlabel('Separation of Data');
title('Performance as a function of data separation');
legend('\omega^*=S_w^-1(m_1-m_2)','\omega_I=(m_1-m_2)')

% figure
% plot(Nvec,C1,'ro-','linewidth',1,'MarkerSize',4); hold on
% plot(Nvec,C2,'bo-','linewidth',1,'MarkerSize',4);
% ylabel('% Correct Classification');
% xlabel('Feature Vector Dimension Size (N)');
% title('Performance as a function of feature vector dimension size');
% legend('\omega^*=S_w^-1(m_1-m_2)','\omega_I=(m_1-m_2)')
```

```
% figure
% plot(Mvec,C1,'ro-','linewidth',1,'MarkerSize',4); hold on
% plot(Mvec,C2,'bo-','linewidth',1,'MarkerSize',4);
% ylabel('% Correct Classification');
% xlabel('Number of Samples (M)');
% title('Performance as a function of data sample size');
% legend('\omega^*=S_w^-1(m_1-m_2)','\omega_I=(m_1-m_2)')

% Question2
clear all;close all;clc

clc; close all;
clear all;

%difvec=0:0.05:5;
%Nvec=[1:1:40];
Mvec=[100:100:1000 1500:500:2500];

% ite=length(difvec);
% ite=length(Nvec);
ite=length(Mvec);

C1=zeros(1,ite);
C2=C1;

for iii=1:ite

    %dif=difvec(iii); N=5; M=2500;
    %N=Nvec(iii); dif=1.5; M=2500;
    M=Mvec(iii); dif=1.5; N=5;

    [y1 y2 muY1 muY2 covY1 covY2]=gen_Ngaussian(N,M,dif);

    % half of the data is for training, rest is for testing
    tr1=y1(1:M/2,:); tr2=y2(1:M/2,:);
    te1=y1(M/2+1:end,:); te2=y2(M/2+1:end,:);

    %%%%%%%
    % SVM %
    %%%%%%%
```

```
A=[tr1; tr2];
len=length(A);
D=eye(len);
D(len/2:end,:)=D(len/2:end,:)*-1;
nu = 1/size(A,1); tol = 1e-5; maxIter = 100; alpha = 1.9/nu;
perturb = 0; normalize = 0;
[iter, optCond, time, w, gamma] = lsvm(A,D,nu,tol,maxIter,alpha, ...
    perturb,normalize);
% w and gamma are used to classify data points
res_tr=D*(A*w-gamma)>0;    % testing on training data
TD_FD_svmTr=[sum(res_tr) len-sum(res_tr)];

% testing on non-training data
A=[te1; te2];
res_te=D*(A*w-gamma)>0;    % testing on training data
TD_FD_svmTe=[sum(res_te) len-sum(res_te)];

%%%%%%%
% ANN %
%%%%%%%
Ptr = [tr1; tr2]';
len=length(tr1);
Tc = [ones(1,len) 2*ones(1,len)];

T = ind2vec(Tc);
spread = 1;
net = newpnn(Ptr,T,spread);

% testing on training data
A = sim(net,Ptr);
Ac_tr = vec2ind(A);
TD=[sum(Ac_tr(1:len)==1)+sum(Ac_tr(len+1:end)==2)];
TD_FD_annTr=[TD 2*len-TD];

% testing on data
Pte = [te1; te2]';
A = sim(net,Pte);
Ac_te = vec2ind(A);
TD=[sum(Ac_te(1:len)==1)+sum(Ac_te(len+1:end)==2)];
```

```
    TD_FD_annTe=[TD 2*len-TD];

    C1(iii)=TD_FD_svmTe(1);
    C2(iii)=TD_FD_annTe(1);

    C1(iii)=C1(iii)/M*100;
    C2(iii)=C2(iii)/M*100;
end

% figure
% plot(difvec,C2,'ro-','linewidth',1,'MarkerSize',4); hold on
% plot(difvec,C1,'bo-','linewidth',1,'MarkerSize',4);
% ylabel('% Correct Classification');
% xlabel('Separation of Data');
% title('Performance as a function of data separation');
% legend('PNN','LSVM')

% figure
% plot(Nvec,C2,'ro-','linewidth',1,'MarkerSize',4); hold on
% plot(Nvec,C1,'bo-','linewidth',1,'MarkerSize',4);
% ylabel('% Correct Classification');
% xlabel('Feature Vector Dimension Size (N)');
% title('Performance as a function of feature vector dimension size');
% legend('PNN','LSVM')

figure
plot(Mvec,C2,'ro-','linewidth',1,'MarkerSize',4); hold on
plot(Mvec,C1,'bo-','linewidth',1,'MarkerSize',4);
ylabel('% Correct Classification');
xlabel('Number of Samples (M)');
title('Performance as a function of data sample size');
legend('PNN','LSVM')
```

LSVM algorithm

```
function [iter, optCond, time, w, gamma] =
lsvm(A,D,nu,tol,maxIter,alpha, ...
                          perturb,normalize);
% LSVM Langrangian Support Vector Machine algorithm
%   LSVM solves a support vector machine problem using an iterative
%   algorithm inspired by an augmented Lagrangian formulation.
```

```
%
%   iters = lsvm(A,D)
%
%   where A is the data matrix, D is a diagonal matrix of 1s and -1s
%   indicating which class the points are in, and 'iters' is the number
%   of iterations the algorithm used.
%
%   All the following additional arguments are optional:
%
%   [iters, optCond, time, w, gamma] = ...
%     lsvm(A,D,nu,tol,maxIter,alpha,perturb,normalize)
%
%   optCond is the value of the optimality condition at termination.
%   time is the amount of time the algorithm took to terminate.
%   w is the vector of coefficients for the separating hyperplane.
%   gamma is the threshold scalar for the separating hyperplane.
%
%   On the right hand side, A and D are required. If the rest are not
%   specified, the following defaults will be used:
%     nu = 1/size(A,1), tol = 1e-5, maxIter = 100, alpha = 1.9/nu,
%       perturb = 0, normalize = 0
%
%   perturb indicates that all the data should be perturbed by a random
%   amount between 0 and the value given. perturb is recommended only
%   for highly degenerate cases such as the exclusive or.
%
%   normalize should be set to 1 if the data should be normalized before
%   training.
%
%   The value -1 can be used for any of the entries (except A and D) to
%   specify that default values should be used.
%
%   Copyright (C) 2000 Olvi L. Mangasarian and David R. Musicant.
%   Version 1.0 Beta 1
%   This software is free for academic and research use only.
%   For commercial use, contact musicant@cs.wisc.edu.

  % If D is a vector, convert it to a diagonal matrix.
  [k,n] = size(D);
  if k==1 | n==1
```

```
  D=diag(D);
end;

% Check all components of D and verify that they are +-1
checkall = diag(D)==1 | diag(D)==-1;
if any(checkall==0)
  error('Error in D: classes must be all 1 or -1.');
end;

m = size(A,1);

if ~exist('nu') | nu==-1
  nu = 1/m;
end;
if ~exist('tol') | tol==-1
  tol = 1e-5;
end;
if ~exist('maxIter') | maxIter==-1
  maxIter = 100;
end;
if ~exist('alpha') | alpha==-1
  alpha = 1.9/nu;
end;
if ~exist('normalize') | normalize==-1
  normalize = 0;
end;
if ~exist('perturb') | perturb==-1
  perturb = 0;
end;

% Do a sanity check on alpha
if alpha > 2/nu,
  disp(sprintf('Alpha is larger than 2/nu. Algorithm may not converge.'));
end;

% Perturb if appropriate
rand('seed',22);
if perturb,
  A = A + rand(size(A))*perturb;
end;
```

```
  % Normalize if appropriate
  if normalize,
    avg = mean(A);
    dev = std(A);
    if (isempty(find(dev==0)))
      A = (A - avg(ones(m,1),:))./dev(ones(m,1),:);
    else
      warning('Could not normalize matrix: at least one column is
constant.');
    end;
  end;

  % Create matrix H
  [m,n] = size(A);
  e = ones(m,1);
  H = D*[A -e];
  iter = 0;
  time = cputime;

  % "K" is an intermediate matrix used often in SMW calclulations
  K = H*inv((speye(n+1)/nu+H'*H));

  % Choose initial value for x
  x = nu*(1-K*(H'*e));

  % y is the old value for x, used to check for termination
  y = x + 1;

  while iter < maxIter & norm(y-x)>tol
    % Intermediate calculation which is used twice
    z = (1+pl(((x/nu+H*(H'*x))-alpha*x)-1));
    y = x;
    % Calculate new value of x
    x=nu*(z-K*(H'*z));
    iter = iter + 1;
  end;

  % Determine outputs
  time = cputime - time;
```

```
  optCond = norm(x-y);
  w = A'*D*x;
  gamma = -e'*D*x;
  disp(sprintf('Running time (CPU secs) = %g',time));
  disp(sprintf('Number of iterations = %d',iter));
  disp(sprintf('Training accuracy = %g',sum(D*(A*w-gamma)>0)/m));

  return;

function pl = pl(x);
  %PLUS function : max{x,0}
  pl = (x+abs(x))/2;
  return;

% Question2_distribution
clear all;close all;clc

clc; close all;
clear all;

dif=1.5;
N=[5];
M=[2500];

ite=3;

C1=zeros(1,ite);
C2=C1;

for iii=1:ite

    if iii==1
        [y1 y2 muY1 muY2]=gen_Random(N,M,dif,'normal');
    elseif iii==2
        [y1 y2 muY1 muY2]=gen_Random(N,M,dif,'exponential');
    else
        [y1 y2 muY1 muY2]=gen_Random(N,M,dif,'uniform');
    end
```

```
% half of the data is for training, rest is for testing
tr1=y1(1:M/2,:); tr2=y2(1:M/2,:);
te1=y1(M/2+1:end,:); te2=y2(M/2+1:end,:);

%%%%%%
% SVM %
%%%%%%

A=[tr1; tr2];
len=length(A);
D=eye(len);
D(len/2:end,:)=D(len/2:end,:)*-1;
nu = 1/size(A,1); tol = 1e-5; maxIter = 100; alpha = 1.9/nu;
perturb = 0; normalize = 0;
[iter, optCond, time, w, gamma] = lsvm(A,D,nu,tol,maxIter,alpha, ...
    perturb,normalize);
% w and gamma are used to classify data points
res_tr=D*(A*w-gamma)>0;    % testing on training data
TD_FD_svmTr=[sum(res_tr) len-sum(res_tr)];

% testing on non-training data
A=[te1; te2];
res_te=D*(A*w-gamma)>0;    % testing on training data
TD_FD_svmTe=[sum(res_te) len-sum(res_te)];

%%%%%%
% ANN %
%%%%%%
Ptr = [tr1; tr2]';
len=length(tr1);
Tc = [ones(1,len) 2*ones(1,len)];

T = ind2vec(Tc);
spread = 1;
net = newpnn(Ptr,T,spread);

% testing on training data
A = sim(net,Ptr);
Ac_tr = vec2ind(A);
TD=[sum(Ac_tr(1:len)==1)+sum(Ac_tr(len+1:end)==2)];
```

```
    TD_FD_annTr=[TD 2*len-TD];

    % testing on data
    Pte = [te1; te2]';
    A = sim(net,Pte);
    Ac_te = vec2ind(A);
    TD=[sum(Ac_te(1:len)==1)+sum(Ac_te(len+1:end)==2)];
    TD_FD_annTe=[TD 2*len-TD];

    C1(iii)=TD_FD_annTe(1);
    C2(iii)=TD_FD_svmTe(1);

    C1(iii)=C1(iii)/M*100;
    C2(iii)=C2(iii)/M*100;
end
C1
C2

% figure
% plot(difvec,C2,'rx-','linewidth',2,'MarkerSize',6); hold on
% plot(difvec,C1,'go-','linewidth',2,'MarkerSize',6);
% ylabel('Percent Correct Classification');
% xlabel('Separation of Data');
% title('Performance as a function of data separation');
% legend('PNN','LSVM')

% figure
% plot(Nvec,C2,'rx-','linewidth',2,'MarkerSize',6); hold on
% plot(Nvec,C1,'go-','linewidth',2,'MarkerSize',6);
% ylabel('Percent Correct Classification');
% xlabel('Feature Vector Dimension Size (N)');
% title('Performance as a function of feature vector dimension size');
% legend('PNN','LSVM')

% figure
% plot(Mvec,C2,'rx-','linewidth',2,'MarkerSize',6); hold on
% plot(Mvec,C1,'go-','linewidth',2,'MarkerSize',6);
% ylabel('Percent Correct Classification');
% xlabel('Number of Samples (M)');
% title('Performance as a function of data sample size');
```

```
% legend('PNN','LSVM')

% Question 3
% classification using PW, NN, KNN

clear all;close all;clc

difvec=0:0.1:5;
Nvec=[1:1:40];
Mvec=[100:100:2500];

ite=length(difvec);
% ite=length(Nvec);
% ite=length(Mvec);

C1=zeros(1,ite);
C2=C1;
C3=C1;
hi=0.3;
k=9;

for iii=1:ite

    dif=difvec(iii); N=5; M=1000;
    %N=Nvec(iii); dif=1.5; M=1000;
    %M=Mvec(iii); dif=1.5; N=5;

    [y1 y2 muY1 muY2 covY1 covY2]=gen_Ngaussian2(N,M,dif);

    % half of the data is for training, rest is for testing
    tr1=y1(1:M/2,:); tr2=y2(1:M/2,:);
    te1=y1(M/2+1:end,:); te2=y2(M/2+1:end,:);

    len=M/2;

    %%%%%%%%%%%%%%%%%
    % Parzen Windows %
    %%%%%%%%%%%%%%%%%

    class1=zeros(len,1);
```

```
    class2=class1;

    for ii=1:len
        % for testing data from class 1
        post1=1/((M/2)*(hi^N))*sum(GaussianWin(N,(repmat(te1(ii,:),M/2,1)-
tr1)/hi));
        post2=1/((M/2)*(hi^N))*sum(GaussianWin(N,(repmat(te1(ii,:),M/2,1)-
tr2)/hi));
        if post1>post2      % p(w1|x)>p(w2|x)
            class1(ii)=1;
        end

        % for testing data from class 2
        post1=1/((M/2)*(hi^N))*sum(GaussianWin(N,(repmat(te2(ii,:),M/2,1)-
tr1)/hi));
        post2=1/((M/2)*(hi^N))*sum(GaussianWin(N,(repmat(te2(ii,:),M/2,1)-
tr2)/hi));
        if post1<post2      % p(w1|x)<p(w2|x)
            class2(ii)=1;
        end
    end

    TD=sum(class1)+sum(class2);
    FD=M-TD;
    TD_FD=[TD FD]
    C1(iii)=TD_FD(1)/M*100;

    %%%%%%%%%%%%%%%%%%%%
    % Nearest Neighbor %
    %%%%%%%%%%%%%%%%%%%%

    class1=zeros(M/2,1);
    class2=class1;

    for ii=1:M/2    % for each testing data point

        s=te1(ii,:);
        d1=sqrt(sum((tr1-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr1
```

```
        d2=sqrt(sum((tr2-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr2

        d1min=min(d1);   % closest point in tr1
        d2min=min(d2);   % closest point in tr2

        % if closest to a point in tr1, then classify as from class 1
        % 0 means belonging to other class 1, 1 means belonging to correct
class
        if d1min<d2min
            class1(ii)=1;
        else
            class1(ii)=0;
        end

        % repeat for each testing data point in class 2
        s=te2(ii,:);
        d1=sqrt(sum((tr1-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr1
        d2=sqrt(sum((tr2-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr2

        d1min=min(d1);   % closest point in tr1
        d2min=min(d2);   % closest point in tr2

        % if closest to a point in tr1, then classify as from class 1
        if d1min<d2min
            class2(ii)=0;
        else
            class2(ii)=1;
        end

    end

    TD=sum(class1)+sum(class2);
    FD=M-TD;

    [true_false_nn]=[TD FD]
    C2(iii)=true_false_nn(1)/M*100;
```

```
    %%%%%%%%%%%%%%%%%%%%
    %K-Nearest Neighbor %
    %%%%%%%%%%%%%%%%%%%%
    class1=zeros(M/2,1);
    class2=class1;

    for ii=1:M/2    % for each testing data point

        s=te1(ii,:);
        d1=sqrt(sum((tr1-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr1
        d2=sqrt(sum((tr2-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr2

        d1=[d1 ones(M/2,1)];         % assigning 1 to denote from class 1
        d2=[d2 -ones(M/2,1)];        % assigning -1 to denote from class 2
        % sorting distances
        d=[d1;d2];
        ds=sortrows(d);             % sortrows only sorts the first column

        dsk=ds(1:k,2);

        val=sum(dsk);   % positive means class1 has more contribution, neg
means other
        if val>0
            class1(ii)=1;
        else
            class1(ii)=0;
        end

        s=te2(ii,:);
        d1=sqrt(sum((tr1-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr1
        d2=sqrt(sum((tr2-repmat(s,M/2,1)).^2,2));   % distance from each
point in tr2

        d1=[d1 ones(M/2,1)];         % assigning 1 to denote from class 1
        d2=[d2 -ones(M/2,1)];        % assigning -1 to denote from class 2
        % sorting distances
        d=[d1;d2];
```

```
        ds=sortrows(d);             % sortrows only sorts the first column

        dsk=ds(1:k,2);

        val=sum(dsk);   % positive means class1 has more contribution, neg
means other
        if val>0
            class2(ii)=0;
        else
            class2(ii)=1;
        end

    end

    TD=sum(class1)+sum(class2);
    FD=M-TD;

    [true_false_knn]=[TD FD]
    C3(iii)=true_false_knn(1)/M*100;

end

figure
plot(difvec,C1,'ro-','linewidth',1.5,'MarkerSize',4); hold on
plot(difvec,C2,'go-','linewidth',1,'MarkerSize',4);
plot(difvec,C3,'bo-','linewidth',1,'MarkerSize',4);
ylabel('% Correct Classification');
xlabel('Separation of Data');
title('Performance as a function of data separation');
legend('Parzen Windows','Nearest Neighbor','K-Nearest Neighbors')

% figure
% plot(Nvec,C1,'ro-','linewidth',2,'MarkerSize',5); hold on
% plot(Nvec,C2,'go-','linewidth',1.5,'MarkerSize',4);
% plot(Nvec,C3,'bo-','linewidth',1.5,'MarkerSize',4);
% ylabel('% Correct Classification');
% xlabel('Feature Vector Dimension Size (N)');
% title('Performance as a function of feature vector dimension size');
% legend('Parzen Windows','Nearest Neighbor','K-Nearest Neighbors')
```

```
% figure
% plot(Mvec,C1,'ro-','linewidth',2,'MarkerSize',5); hold on
% plot(Mvec,C2,'go-','linewidth',1.5,'MarkerSize',4);
% plot(Mvec,C3,'bo-','linewidth',1.5,'MarkerSize',4);
% ylabel('% Correct Classification');
% xlabel('Number of Samples (M)');
% title('Performance as a function of data sample size');
% legend('Parzen Windows','Nearest Neighbor','K-Nearest Neighbors')

% Parzen window density estimation for 1 or 2 dimensions!

clear all;close all;clc

% density estimation
N=1;                % dimension of feature vector
M=10000;               % number of samples for class 1 and class 2
dif=0.6;            % difference between two means
hi=0.2;             % defines window width

[y1 y2 muY1 muY2 covY1 covY2]=gen_Ngaussian2(N,M,dif);

% half of the data is for training, rest is for testing
tr1=y1(1:M/2,:); tr2=y2(1:M/2,:);
te1=y1(M/2+1:end,:); te2=y2(M/2+1:end,:);

x0=-3:0.1:4;  % user defined range for density estimate
len=length(x0);

if N==1
    p=zeros(1,len);
    for ii=1:len
        p1(ii)=1/((M/2)*(hi^N))*sum(GaussianWin(N,(repmat(x0(ii),M/2,1)-
tr1)/hi));
    end
    figure
    pgss=normpdf(x0,muY1,sqrt(covY1));
    subplot(2,1,1);
    plot(x0,p1,'r','linewidth',2);
    hold on
    plot(x0,pgss,'b','linewidth',2);
```

```
    subplot(2,1,2);
    [d c]=hist(tr1,len);
    delta=abs(c(1)-c(2));
    bar(c,(d/(M/2))/delta)
    xlim([min(x0) max(x0)]);

elseif N==2
    p=zeros(len,len);
    for ii=1:len
        for iii=1:len
            p1(ii,iii)=1/((M/2)*(hi^N))*sum(GaussianWin(N,(repmat([x0(ii)
x0(iii)],M/2,1)-tr1)/hi));
        end
    end
    figure
    mesh(x0,x0,p1); axis tight
end

% Parzen window density estimation for 1 or 2 dimensions!
% examines effect of window size on density estimation

clear all;close all;clc

% density estimation
N=1;                % dimension of feature vector
M=5000;               % number of samples for class 1 and class 2
dif=0.8;            % difference between two means

hivec=0.005:0.01:1;             % defines window width
lenh=length(hivec);

x0=-2.5:0.1:3;  % user defined range for density estimate
len=length(x0);

p1=zeros(lenh,len);

for iii=1:lenh
    hi=hivec(iii);

    %Gaussian distribution
```

```
%     [y1 y2 muY1 muY2 covY1 covY2]=gen_Ngaussian2(N,M,dif);
%     %Uniform distribution
    y1=rand(M,N);
    % half of the data is for training, rest is for testing
    tr1=y1(1:M/2,:);
    te1=y1(M/2+1:end,:);
    p=zeros(1,len);
    for ii=1:len

p1(iii,ii)=1/((M/2)*(hi^N))*sum(GaussianWin(N,(repmat(x0(ii),M/2,1)-
tr1)/hi));
    end
 end
figure
subplot(2,1,1)
mesh(x0,hivec,p1)
xlabel('x');
ylabel('window size');
zlabel('p_e_s_t(x)');
title('Density estimate of Gaussian Distributed data');
axis tight

% pdf=normpdf(x0,muY1,sqrt(covY1));
pdf=unifpdf(x0,0,1);
mse=sum((p1-repmat(pdf,lenh,1)).^2,2);
subplot(2,1,2);
plot(hivec,mse,'r-','linewidth',1.5);
xlabel('window size'); ylabel('MSE');
title('MSE at given window length');
```