Question 1

Parametric methods compute the parametric decision boundaries to approximate the true decision boundaries between classes. One common parametric method involves the use of the Fisher linear discriminant to find a projection such that the projected data can best be separated. The Fisher linear discriminant for two classes with d_1 samples from class 1 and the remaining samples from class 2 is defined as follows:

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

where $\tilde{m}_1 = \vec{w} \cdot \frac{1}{d_1} \sum_{i=1}^{d_1} y_i$, $\tilde{m}_2 = \vec{w} \cdot \frac{1}{d_2} \sum_{i=d_1+1}^{d} y_i$ are the means of the projected data

$\tilde{S}_1^{\,2} = \sum_{i=1}^{d_1} (\vec{w} \cdot y_i - \tilde{m}_1)$, $\tilde{S}_2^{\,2} = \sum_{i=d_1+1}^{d} (\vec{w} \cdot y_i - \tilde{m}_2)$ are the variances

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

and $S_w = \left(\sum_{i=1}^{d_1} \frac{\vec{w}}{|\vec{w}|} \cdot (y_i - \tilde{m}_1) \right)^2 + \left(\sum_{i=d_1+1}^{d_2} \frac{\vec{w}}{|\vec{w}|} \cdot (y_i - \tilde{m}_2) \right)^2$ is the within class scatter.

Maximizing the Fisher linear discriminant yields the optimal projection $\vec{w}^* = S_w^{-1}(\vec{m}_1 - \vec{m}_2)$. Intuitively, this optimization problem is computing the projection that maximizes the ratio between the separation of the means of the projected data and the within class scatter of the projected data – producing a projection that separates the projected means while reducing the scatter/variance of the projected data.

The goal of this question is to evaluate the performance of the classifier using $\vec{w}^*$ as the projection versus using $\vec{w}_I = (\vec{m}_1 - \vec{m}_2)$. The following procedure was followed in writing the Matlab algorithm:

- Generate correlated Gaussian data from class 1 and class 2. Data from class 1 and class 2 have the same covariance (i.e. $\Sigma_1 = \Sigma_2$), but the means are different (i.e. $\vec{m}_2 = \vec{m}_1 + \Delta$ to produce separation between the classes. The mean of class 1 is set as $\vec{m}_1 = \{0.5,\ 1,\ 1.5,\ \ldots,\ N/2\}$.
- Compute the optimal projection $\vec{w}^* = S_w^{-1}(\vec{m}_1 - \vec{m}_2)$.
- Apply the projection to the data: $\vec{w}^* \vec{y} = \vec{b}$.
- Find the threshold to classify the data: $w_0 = \frac{\tilde{m}_1 - \tilde{m}_2}{2}$ for Gaussian data from two classes with equal covariance matrices.
- Classify the i^{th} data point as from class 1 if $b_i > w_0$, else from class 2. Count number of true and false classifications.
- Repeat procedure for $\vec{w}_I = (\vec{m}_1 - \vec{m}_2)$ on same data.
- Compare the performance of classification using $\vec{w}^*$ versus using $\vec{w}_I$.

To evaluate the performance of the two different methods, several simulations were performed to assess: (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.

Case 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 4 while the feature vector size was kept at $N = 4$ and the number of samples for each class was 10000. Figure 1 shows the results, revealing that the method employing $\vec{w}^*$ outperforms the method using $\vec{w}_I$ at all separations. It is interesting to note that at very small and very large separations the performance of both methods are somewhat similar – but at intermediate separations, method $\vec{w}^*$ is superior in accuracy to method $\vec{w}_I$ by approximately 10%. This is to be expected, as when the data has no separation, no method will achieve more than chance accuracy, no matter how good the algorithm is; when there is a very large separation, any decent method should produce good classification.

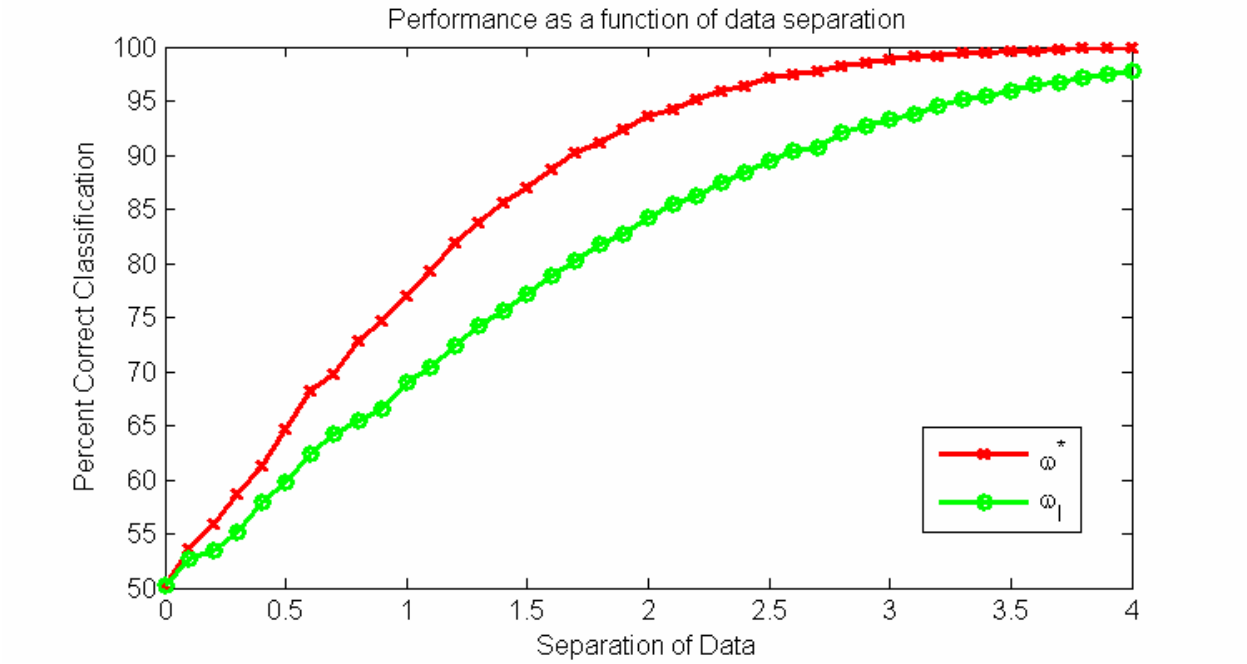

Figure 1: Performance as a function of data separation.

Case 2: Feature vector dimension size

To evaluate performance as a function of separation between classes feature vector dimension size N was varied while the separation was set at $\Delta = 1.5$ and the number of samples for each class was 10000. Figure 2 depicts the results, again showing that the performance of $\vec{w}^*$ is superior to that of method $\vec{w}_I$ by about 10% in accuracy at all feature vector dimension sizes. An increase in dimension size when N is small results in a large increase in accuracy for both algorithms. On the other hand, an additional increase in dimension size when N is already large only results in minimal improvement in accuracy for both algorithms.

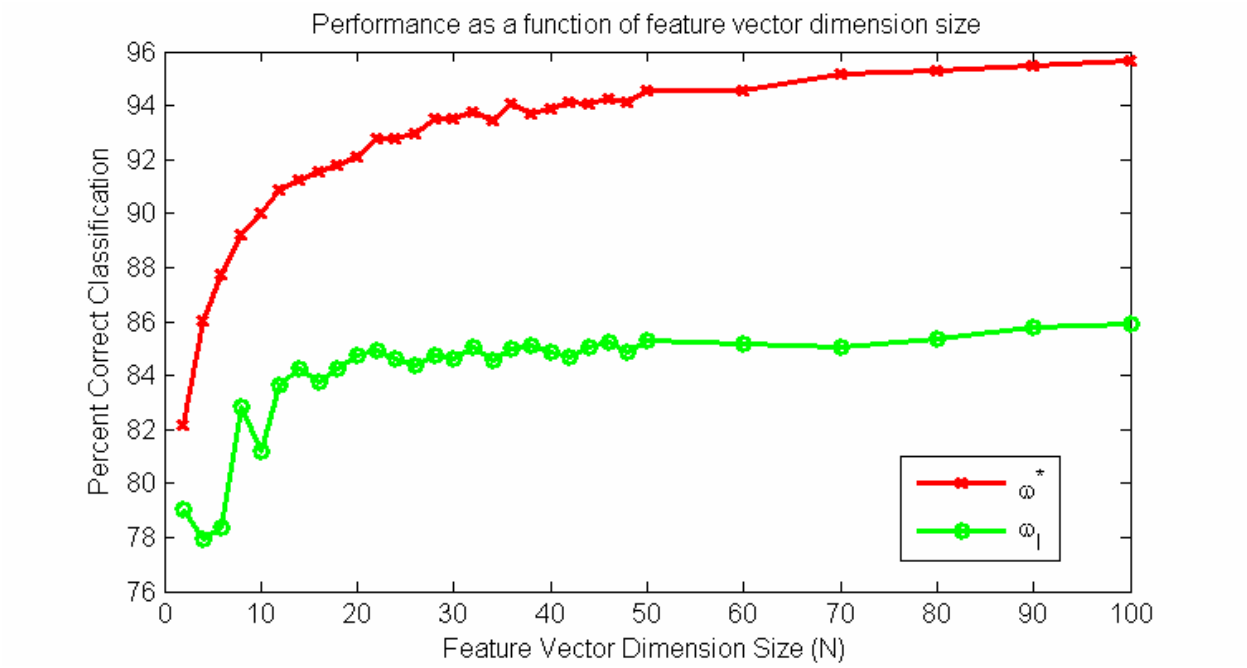

Figure 2: Performance as a function of feature vector dimension size.

Case 3: Data sample size

To evaluate performance as a function of data sample size, number of samples M for each class was varied with the separation set at $\Delta = 1.5$ and the feature vector dimension equal to 4. Figure 3 shows the results. It is interesting to note that while method $\vec{w}^*$ outperforms $\vec{w}_I$ at all sample sizes, neither method exhibited an increase in accuracy as sample size increases.

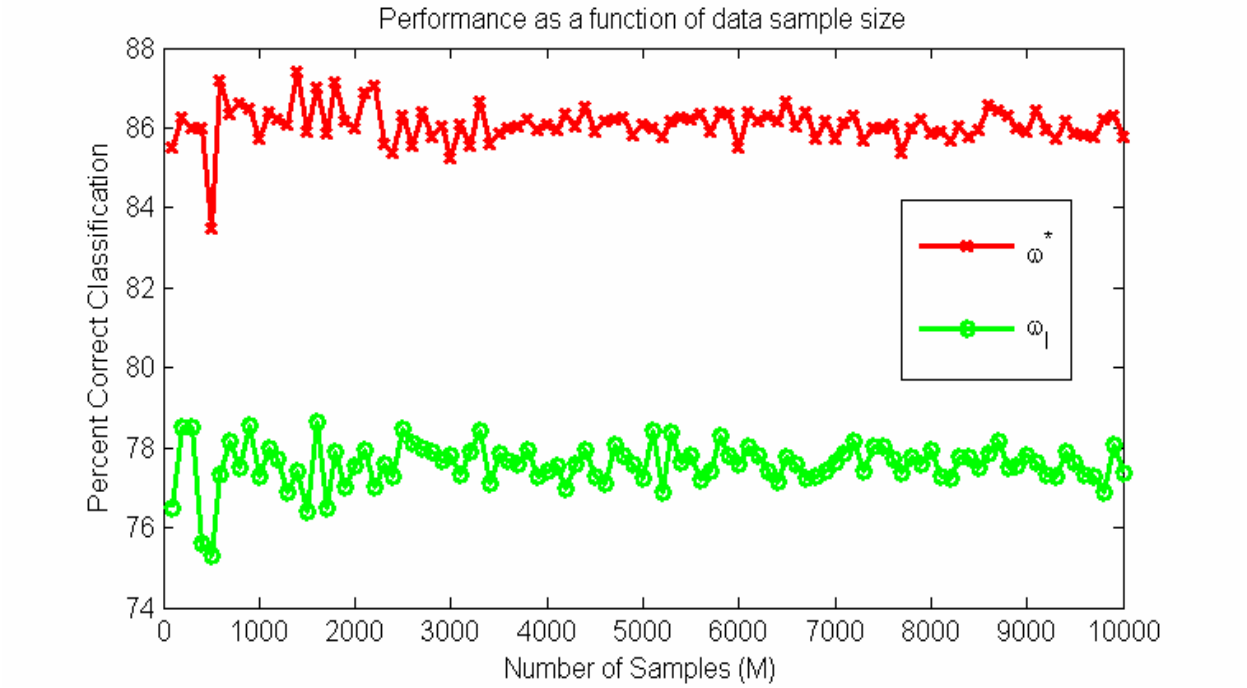

Figure 3: Performance as a function of data sample size.

Discussion of Results

In all the simulations conducted above, method $\vec{w}^*$ achieved superior performance over $\vec{w}_I$. While the simulations by no means cover all possible cases, the results do reveal with a large degree of certainty that $\vec{w}^*$ is superior – possibly explainable by theory. Recall that the Fisher discriminant is $J(\vec{w}) = \frac{\left|\tilde{m}_1 - \tilde{m}_2\right|^2}{\tilde{S}_1^{\,2} + \tilde{S}_2^{\,2}} = \frac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}$. Maximizing $J(\vec{w})$ finds the projection $\vec{w}^*$ 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. The denominator of the Fisher discriminant not only serves to make $J(\vec{w})$ independent of $\|\vec{w}\|$ but also regulates the within-class scatter. On the other hand, the projection $\vec{w}_I = (\vec{m}_1 - \vec{m}_2)$ does not account for within-class scatter – resulting in a sub-optimal projection that while separating the means of the projected data, does not seek to optimize the variance/scatter of the projected data. Therefore, method $\vec{w}^*$ can be expected to achieve superior performance over method $\vec{w}_I$.

Question 2

The performance of a neural network method was examined and compared to the performance of a support vector machine method for classification.

The neural network method used in this report is part of the Neural Network Toolbox in Matlab (The Mathworks). Specifically, the following functionalities were used for classification.

- *net = newpnn(P, T, SPREAD):* Implements a probabilistic neural network (PNN; a type of radial basis network) 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.
- *a = sim(net,P):*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 net, and a separate set testing data was used to compute the accuracy of the algorithm.

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 the 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.

As before, to evaluate and compare the performance of LSVM and PNN, several experiments were performed: (1) performance as a function of separation between classes, (2) performance as a function of feature vector dimension size, (3) performance as a function of number of samples, and (4) performance with different distributions. Note that cases (1)-(3) utilized a Gaussian distribution.

Case 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 4 while the feature vector size was kept at $N = 4$ and the number of samples for each class was 2000. Note that half the samples from each class was utilized as training data, and the remaining half used as testing data. Figure 4 shows the results of this simulation, revealing that while LSVM achieves superior accuracy with small separations (< 0.6), PNN quickly overtakes LSVM in accuracy at approximately $\Delta = 0.6$. While PNN almost achieves perfect accuracy at $\Delta = 4$, LSVM only has an accuracy of less than 90%.

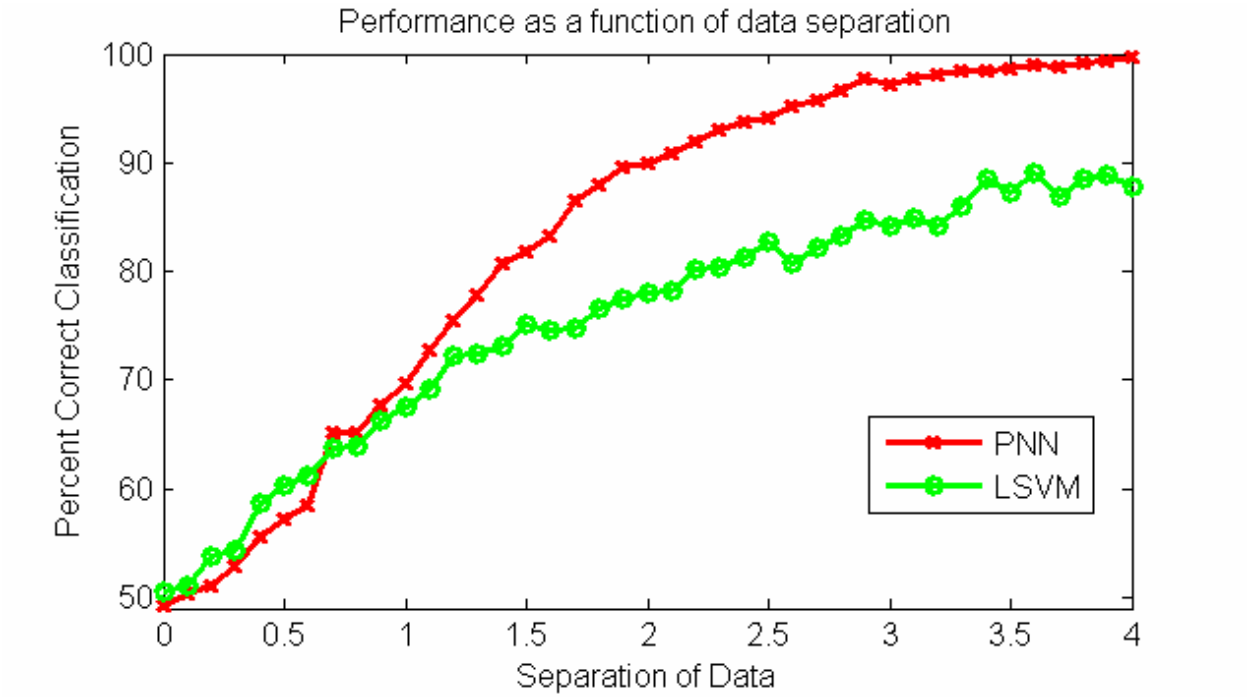

Figure 4: Performance as a function of data separation.

Case 2: Feature vector dimension size

To evaluate performance as a function of separation between classes, feature vector dimension size N was varied while the separation was set at $\Delta = 1.5$ and the number of samples for each class was 2000 (half used as training data, remaining half used as testing data). Figure 5 depicts the results, showing that while PNN achieves superior performance at low dimension sizes (< 12), it fails disastrously at higher dimension sizes and only achieves chance accuracy. LSVM, on the other hand, steadily improves in accuracy as feature vector dimension size increases.

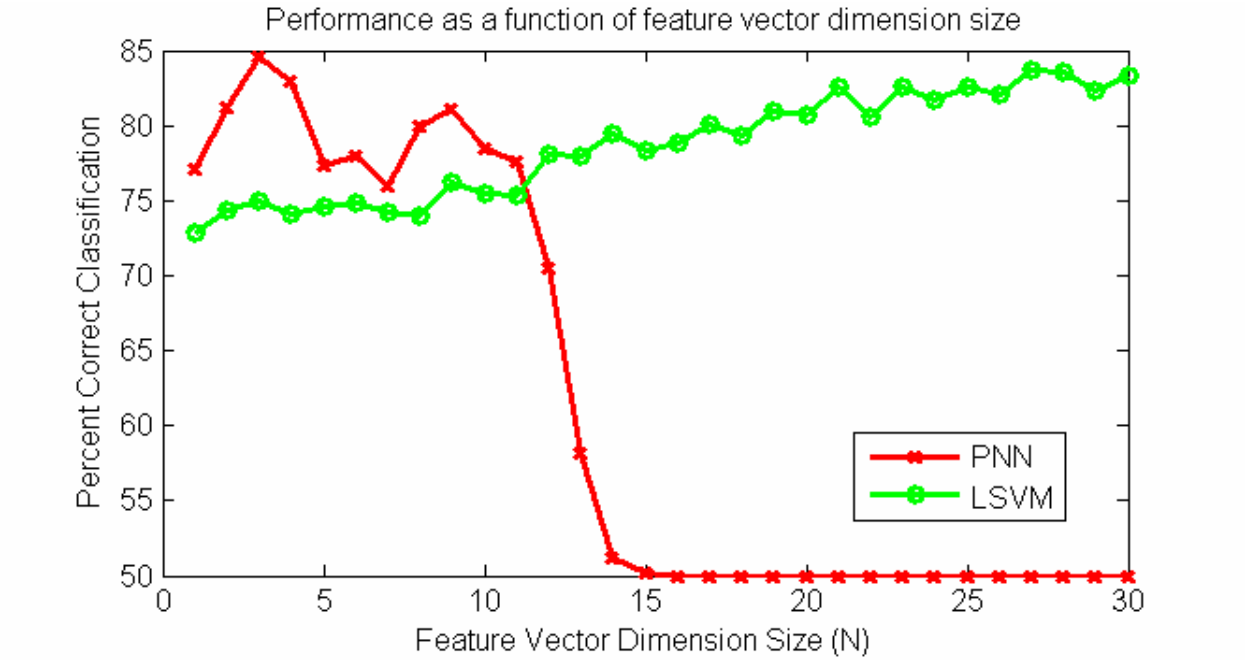

Figure 5: Performance as a function of feature vector dimension size.

Case 3: Data sample size

To evaluate performance as a function of data sample size, the number of samples M for each class was varied while the separation was set at $\Delta = 1.5$ and the feature vector dimension size was 4. Note that half of the samples was used for training and the rest for testing. Figure 3 shows the results, revealing that PNN consistently achieves better accuracy than LSVM at the given separation and dimension size for all sample sizes, although increasing the sample size past 1000 does not improve accuracy for either method.

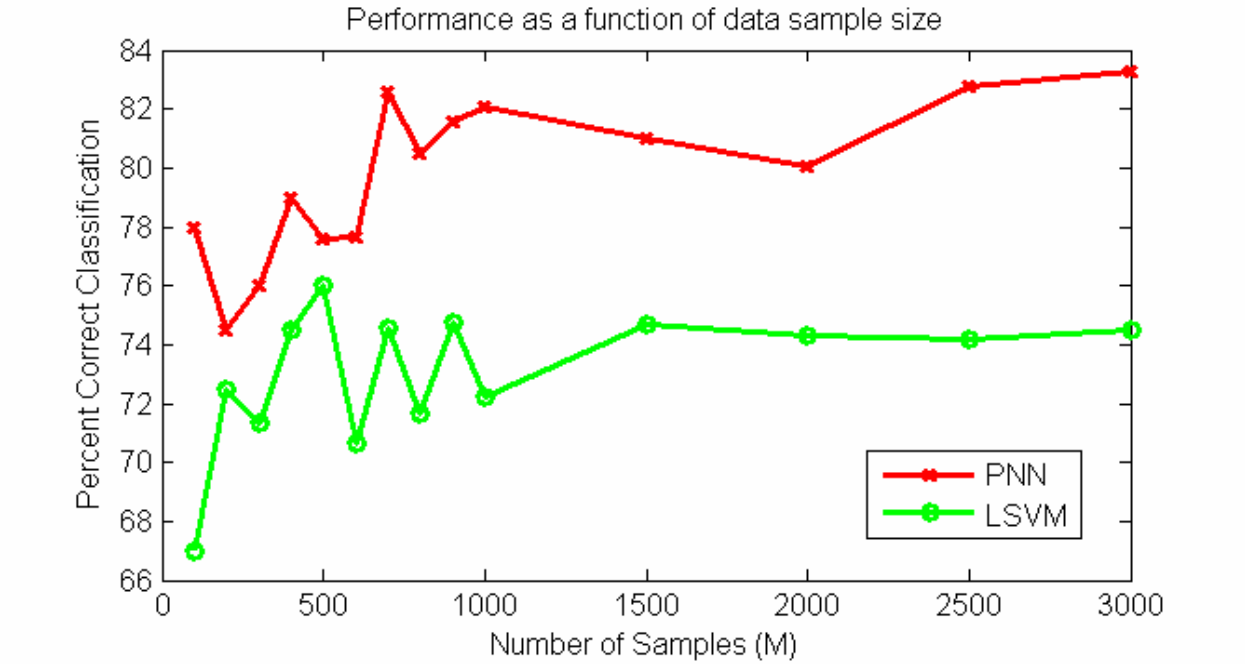

Figure 6: Performance as a function of data sample size.

Case 4: Distribution

To evaluate performance of LSVM and PNN for data with different distributions, correlated Gaussian, exponential, and uniform random data were generated. Number of samples M for each class was 2000; the separation of the means was set at $\Delta = 1.5$; and feature vector dimension size was 4. Table 1 tallies the results, showing that PNN achieved superior accuracy for the Gaussian and exponential data. Both methods achieved chance accuracy for the uniformly distributed data, due to the lack of separation.

	Gaussian	**Exponential**	**Uniform**
PNN	81.45%	78.30%	50.15%
LSVM	74.35%	72.55%	49.85%

Table 1: Accuracy of PNN and LSVM with different distributions

From the simulations conducted above, both PNN and LSVM are shown to have particular 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 exhibits 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 is a nonparametric density estimation technique that can be applied to classification. Using a given window function, this technique approximates the distribution of a training set using a window centered at a desired point $\bar{x}_0$ 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 producing the maximum weighted mean. While the choice of a window function is important, the choice of a sensible window size/side-length is crucial to accurate density estimation and classification. A small window length may lead to spiky behavior in the density estimate while an excessively 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 approach was taken:

- Generate *M*=1000 samples from a univariate Gaussian distribution $X \sim N(0.5,\ 1)$
- Use a given Gaussian 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 Gaussian distribution
- Repeat procedure with data from a uniform distribution $X \sim U(0,\ 1)$

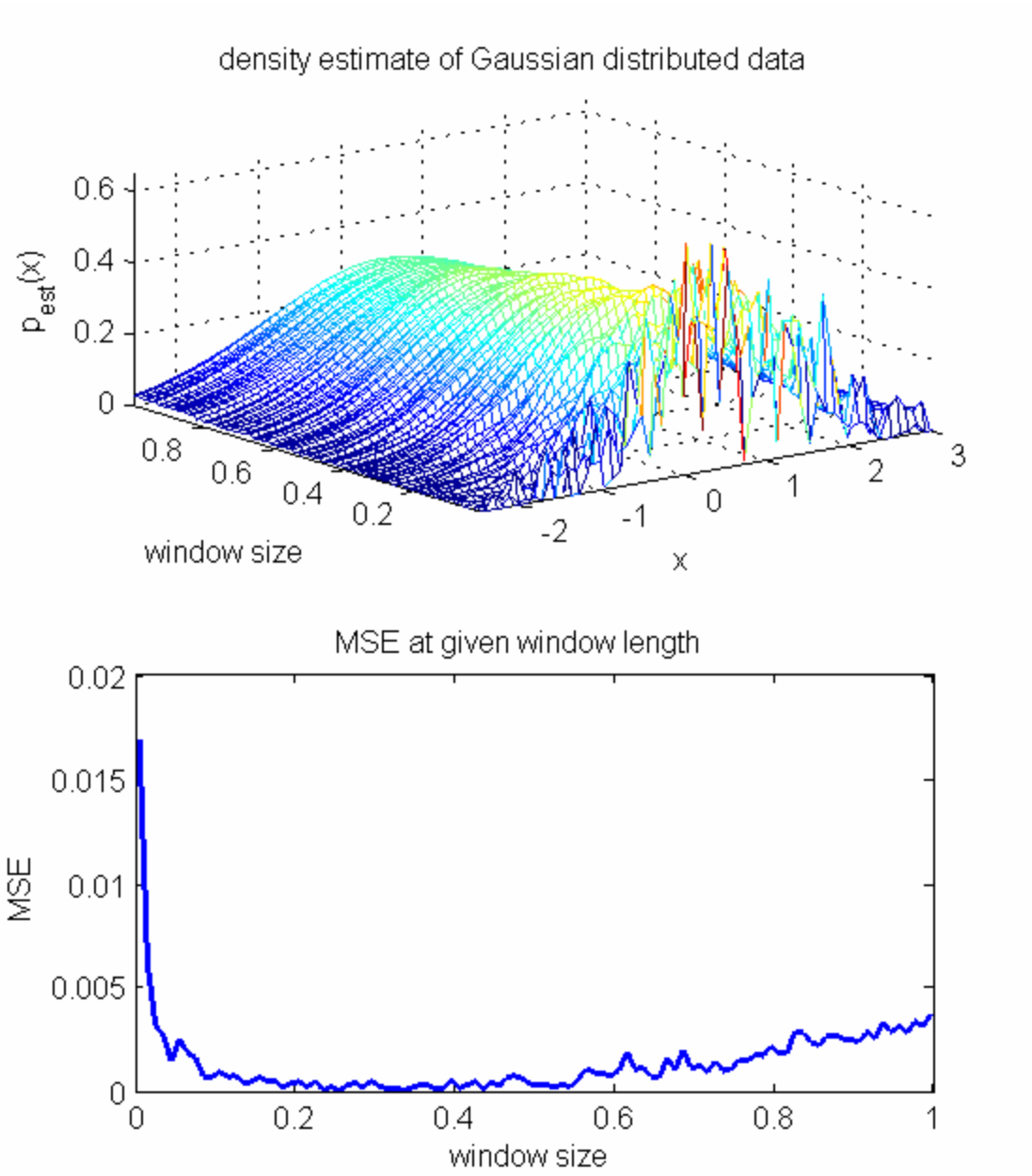

Figure 7: Density estimate as a function of window length for Gaussian data.

As can be observed from the first subplot, tiny window length results in spiky 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 with window size h=0.295.

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 with window size h=0.065. It is interesting to note that perhaps using a rectangular window function would result

in a better density estimate for the uniform case. If the underlying distribution is known, perhaps a window function of the same shape as the true underlying probability density function would produce the best density estimate (with an appropriate window size). But in reality, the underlying distribution is complicated and rarely known, thus a generic Gaussian window function is commonly used.

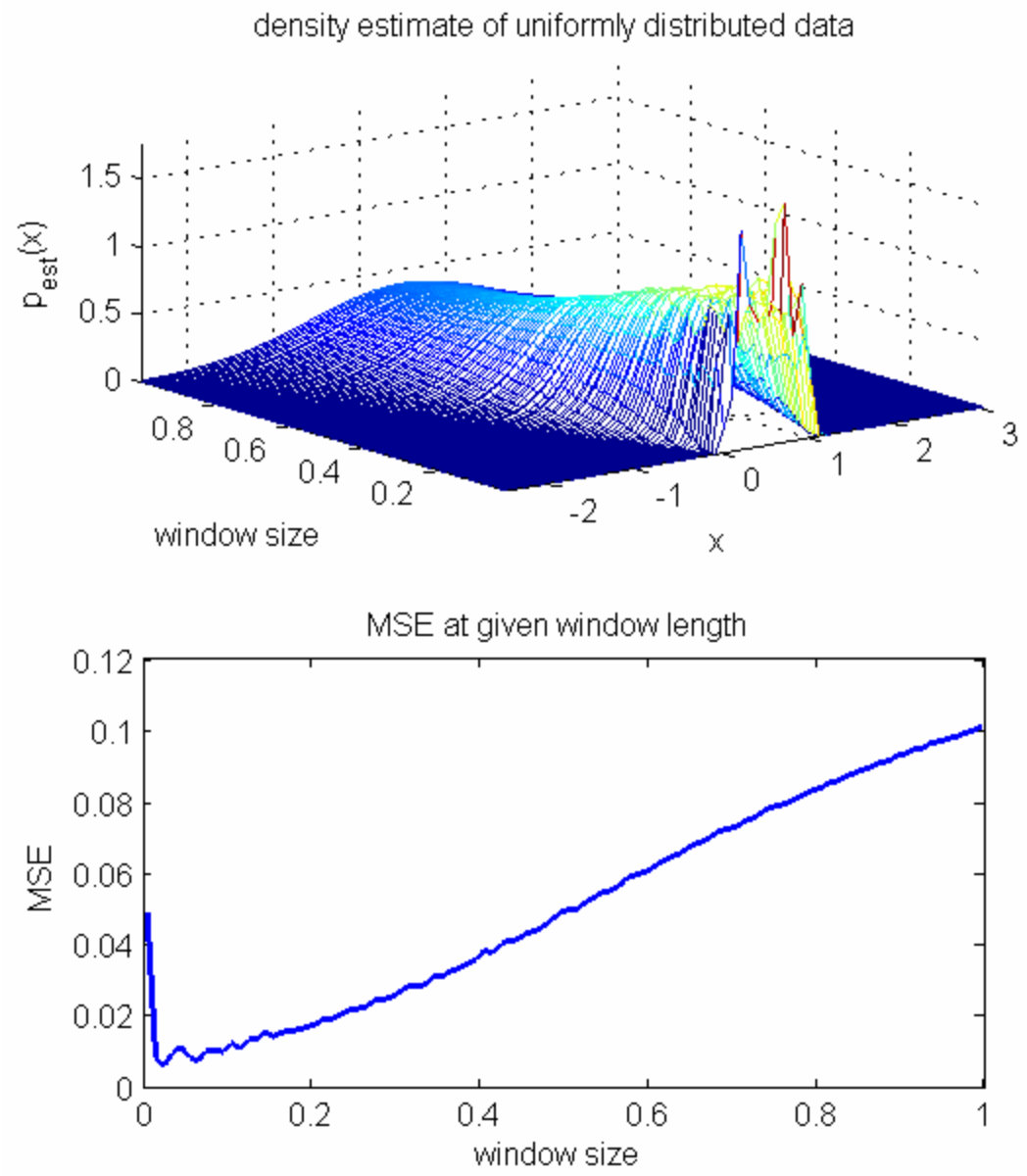

Figure 8: Density estimate as a function of window length for uniformly distributed data.

Nearest Neighbor and K-Nearest Neighbor

Nearest neighbor and K-nearest neighbor approaches are relatively 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 9 illustrates the K-nearest neighbor technique. Of the five training points closest to the square testing point, three are from the blue class and two from the red class, therefore the testing point is labeled to be from the blue class.

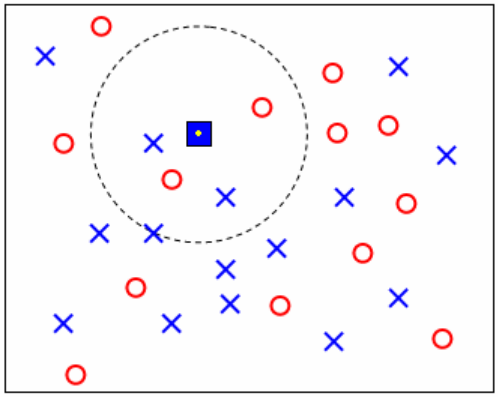

Figure 9: Illustration of K-Nearest Neighbor with K = 5

As before, to evaluate and compare the performance of Parzen windows, nearest neighbor and K-nearest neighbors, several simulations were performed to assess: (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.

Case 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 4 while the feature vector size was kept at $N = 2$ and the number of samples for each class was 1000. The window side-length for Parzen windows is $h = 0.3$. The number of neighbors for K-nearest neighbors is set at $K = 7$.

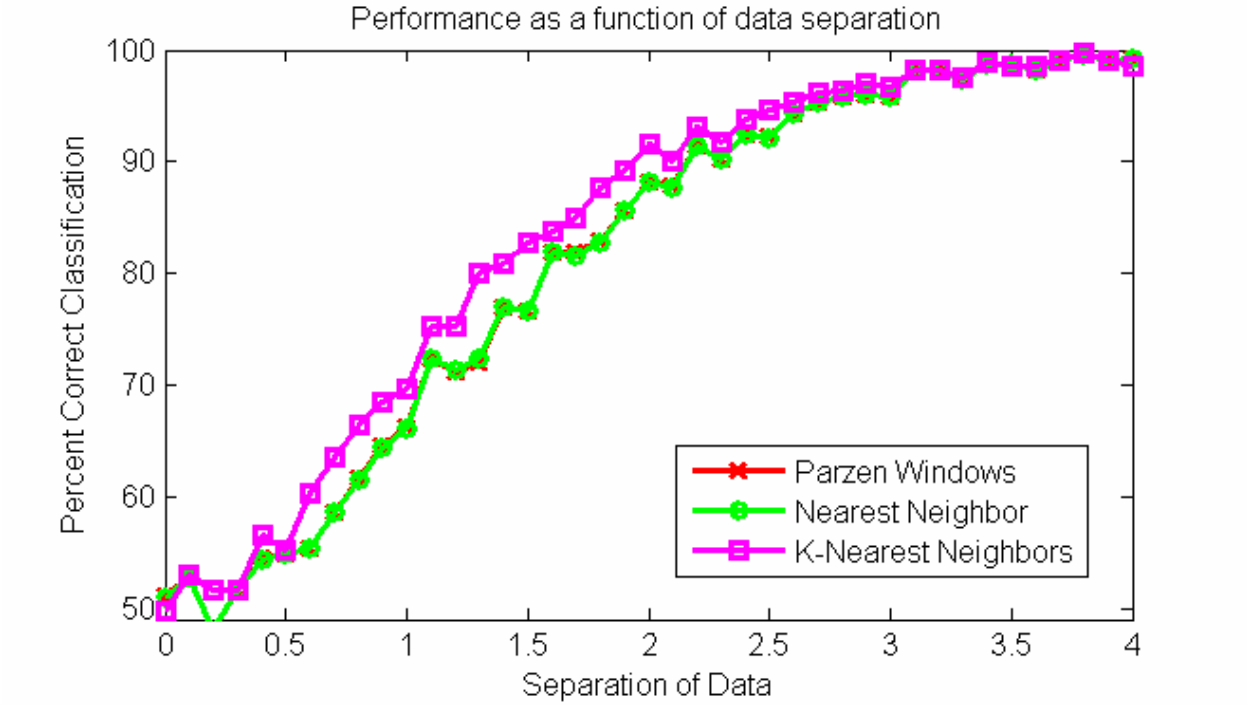

Figure 10: Performance as a function of data separation.

From Figure 10, it can be observed the accuracy curve for Parzen windows almost exactly overlaps the curve for nearest neighbors. K-nearest neighbors method produces consistently better accuracy at all separations.

Case 2: Feature vector dimension size

To evaluate performance as a function of separation between classes, feature vector dimension size N was varied, while the separation was set at $\Delta = 1.5$ and the number of samples for each class was 1000 (half used as training data, remaining 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 $K = 7$. Again, K-nearest neighbors has consistently higher accuracy as feature vector dimension size increases. Note that increasing the dimension size does not result in a significant increase in accuracy for nearest neighbor or K-nearest neighbor. On the other hand, increasing the dimension size results in a decrease in accuracy for the Parzen windows methods – at $N > 11$, accuracy plummets to zero percent. This can be explained by looking at the Gaussian window function:

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

For N 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 $j(\vec{u})$ from class 1 is equal to $j(\vec{u})$ from class 2 – all are equal to zero. The script for Parzen Windows 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.

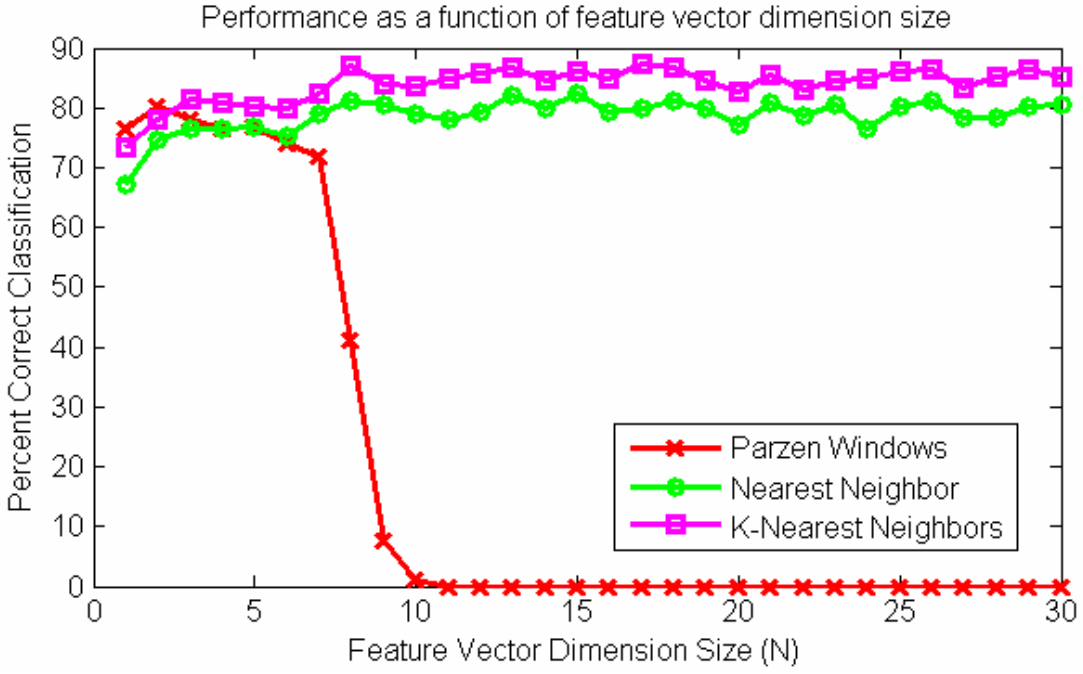

Figure 11: Performance as a function of feature vector dimension size.

Case 3: Data sample size

To evaluate performance as a function of data sample size, the number of samples M for each class was varied while the separation was set at $\Delta = 1.5$ and the feature vector dimension size was 4. Note that half of the samples 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 outperforms the two other methods which overlap almost exactly with each other. Increasing sample size does not appear to increase performance for any of these three methods.

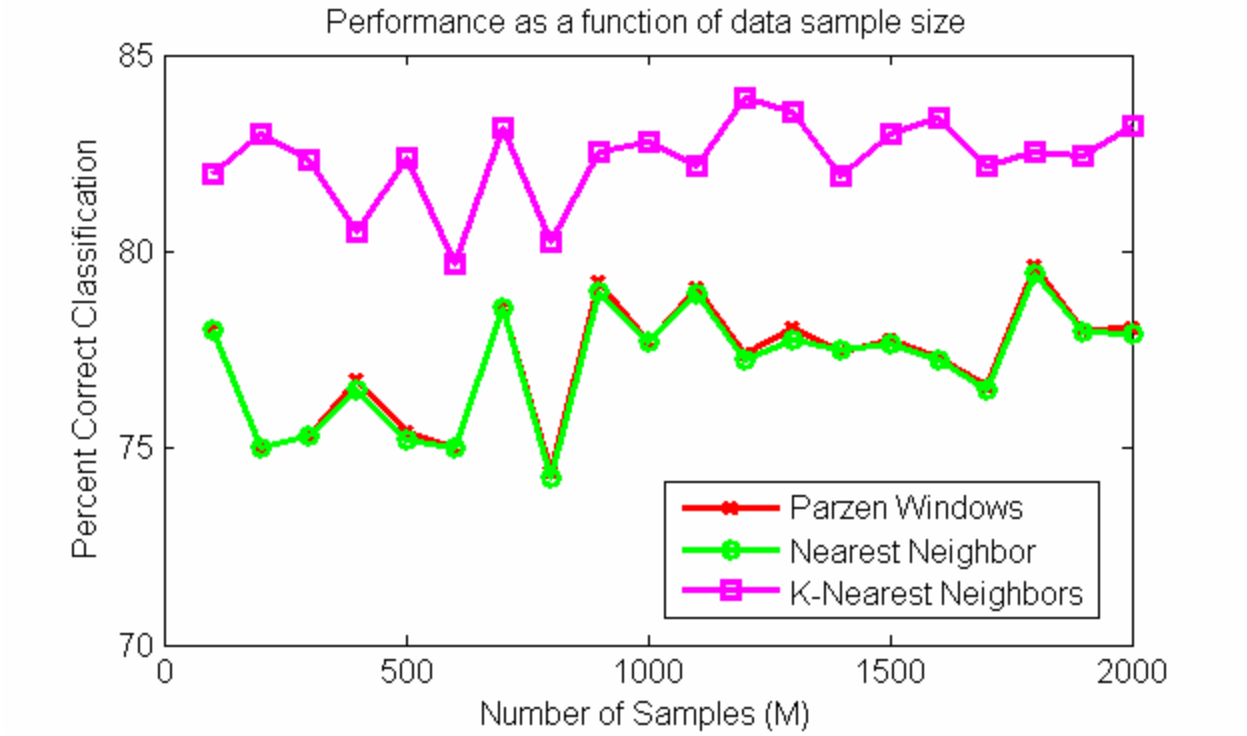

Figure 12: Performance as a function of data sample size.

For the simulations conducted to examine Parzen windows, nearest neighbor, and K-nearest neighbors, K-nearest neighbors method has been shown to possess the highest accuracy. The performance of Parzen windows is almost exactly the same as that of nearest neighbor in almost all cases examined 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 alter the accuracy of K-nearest neighbors method depending on the situation.

```
%%%%%%%%%%%%%%
% Question 1 %
%%%%%%%%%%%%%%
clc; close all;
clear all;

% difvec=0:0.1:4;
% Nvec=[2:2:50 60:10:100];
Mvec=[100:100:10000];

ite=length(Mvec);

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

for iii=1:ite

    %dif=difvec(iii); N=4; M=10000;
    %N=Nvec(iii); dif=1.5; M=10000;
    M=Mvec(iii); dif=1.5; N=4;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %  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,'rx-','linewidth',2,'MarkerSize',6); hold on
% plot(difvec,C2,'go-','linewidth',2,'MarkerSize',6);
% ylabel('Percent Correct Classification');
% xlabel('Separation of Data');
% title('Performance as a function of data separation');
% legend('\omega^*','\omega_I')

% figure
% plot(Nvec,C1,'rx-','linewidth',2,'MarkerSize',6); hold on
% plot(Nvec,C2,'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('\omega^*','\omega_I')

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


%%%%%%%%%%%%%%
% Question 2 %
%%%%%%%%%%%%%%

clear all;close all;clc

%difvec=0:0.1:4;
Nvec=[1:1:30];
%Mvec=[100:100:1000 1500:500:3000];
ite=length(Nvec);

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

for iii=1:ite

    %dif=difvec(iii); N=4; M=2000;
    N=Nvec(iii); dif=1.5; M=2000;
    %M=Mvec(iii); dif=1.5; N=4;

    [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,'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=0:0.1:4;
Nvec=[1:1:30];
Mvec=[100:100:2000];

ite=length(Mvec);

C1=zeros(1,ite);
C2=C1;
C3=C1;
hi=0.3;
k=7;

for iii=1:ite

    %dif=difvec(iii); N=4; M=1000;
    %N=Nvec(iii); dif=1.5; M=1000;
    M=Mvec(iii); dif=1.5; N=4;

    [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,'rx-','linewidth',2,'MarkerSize',8); hold on
% plot(difvec,C2,'go-','linewidth',2,'MarkerSize',6);
% plot(difvec,C3,'ms-','linewidth',2,'MarkerSize',6);
% ylabel('Percent 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,'rx-','linewidth',2,'MarkerSize',8); hold on
% plot(Nvec,C2,'go-','linewidth',2,'MarkerSize',6);
% plot(Nvec,C3,'ms-','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('Parzen Windows','Nearest Neighbor','K-Nearest Neighbors')

figure
plot(Mvec,C1,'rx-','linewidth',2,'MarkerSize',6); hold on
plot(Mvec,C2,'go-','linewidth',2,'MarkerSize',6);
plot(Mvec,C3,'ms-','linewidth',2,'MarkerSize',6);
ylabel('Percent Correct Classification');
xlabel('Number of Samples (M)');
title('Performance as a function of data sample size');
legend('Parzen Windows','Nearest Neighbor','K-Nearest Neighbors')
```