Question 1

Aim:

To perform an experiment by devising a fisher discriminant classifier to classify different classes of data, and to make a modification in the design and verify the outcomes, and finally to draw conclusions from the same

Introduction:

The fisher discriminant is one of the popularly used techniques for pattern classification cum recognition. It deals with the projection of data with higher dimensionality into lower subspaces, so that faster and easier computation is facilitated. One should not that even though samples could form well-separated, compact clusters in *d*-space, projection onto an arbitrary line will usually produce a confused mixture of samples from all of the classes, and thus poor recognition performance. However, by moving the line around, we might be able to find an orientation for which the projected samples are well separated. This is exactly the goal of classical discriminant analysis.

The equation of the Fisher linear discriminant function is given [1] as follows:

$$J(w) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{|\tilde{s}_1^2 - \tilde{s}_2^2|} \quad (1)$$

which on simplification yields the Fisher's linear discriminant - the linear function yielding the maximum ratio of between-class scatter to within-class scatter, given by

$w = S_W^{-1}(m_1 - m_2)$, where $S_W = S_1 + S_2$, S_1, S_2 being the scatter matrices and S_W is called the *within class scatter matrix*. Our aim hence would be to maximize w so as to obtain the best possible discriminating surface.

Procedure:

Let us run through the steps that were followed during the experimentation process. Firstly, we chose to use artificially generated data so as to enhance flexibility in the experimentation process. This was done using the '*mvnrnd'* command in Matlab®. Two sets of data were generated using this, for varying degrees of overlapping and scattering. This was first tested

using the Fisher linear discriminant, followed by what is called the modified fisher discriminant, which was obtained by modifying equation (1), given by $w = (m_1 - m_2)$. Accuracies of both were compared and tabulated.

We shall now describe the method that was used in finding out the line of separation in 2-D or the plane of separation in 3-D. We may recall from [1] that $w^T x + w_0 = 0$, where w_0 is a constant that involves w and the prior probabilities. After obtaining w_0, we can compute the separating line with the knowledge that it is of the form $-bx + ay = k$, given that w is of the form $ax + by = c$. In case of 3-D, if w be the plane of projection, we first project the data (x) to a plane w_n, which is a plane perpendicular to w. This projection of x is then subtracted from x, which will give us the equation of the plane of projection.

Results and Discussion:

Simulations were carried out in MATLAB®, and the results were recorded. We shall now discuss results so obtained for several different cases, with respect to the distribution of data. The discussion that follows, will deal with both the Fisher linear discriminant and its modified version.

Case 1

Here we deal with 2-D data that are well separated in space.

	Class I	Class II
Mean	[1 2]	[-2 -4]
Variance	$\begin{pmatrix} 0.4 & 0 \\ 0 & 0.4 \end{pmatrix}$	$\begin{pmatrix} 0.4 & 0 \\ 0 & 0.4 \end{pmatrix}$

Method	Accuracy %
Fisher	100
Modified fisher	100

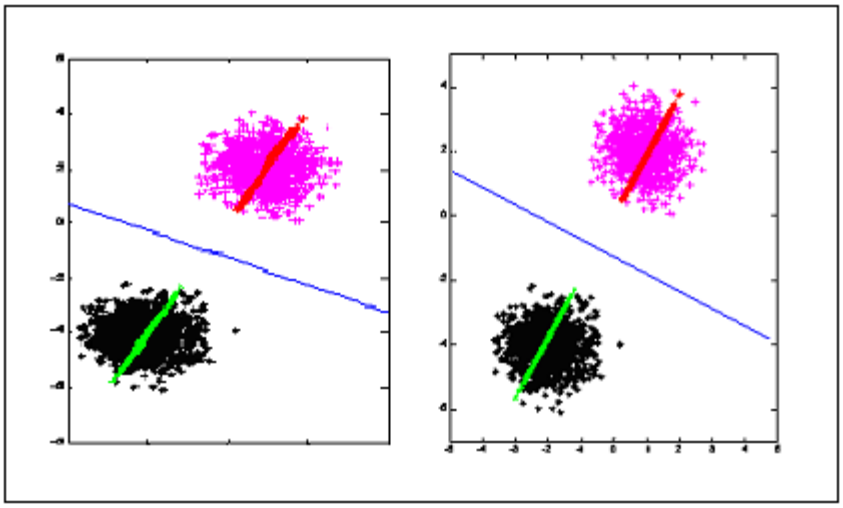

Fig1: Results & Discussion, Case 1

Figure 1 shows the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. As we can see, the modified fisher discriminant produces a separation surface that is perpendicular to the line joining the means of the two classes of data.

This is a typical case wherein the error is zero in spite of *w* being different. We get cent percent accuracy rates in both cases because the classes are well separated out in space. Hence it is not possible to investigate the better classifier of these two, for this case.

Case 2

Here we shall examine the case of 2-D separated data which lead to an error in classification

	Class I	Class II
Mean	[5 6]	[2 6]
Variance	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}$

Method	Accuracy %
Fisher	100
Modified fisher	96.15

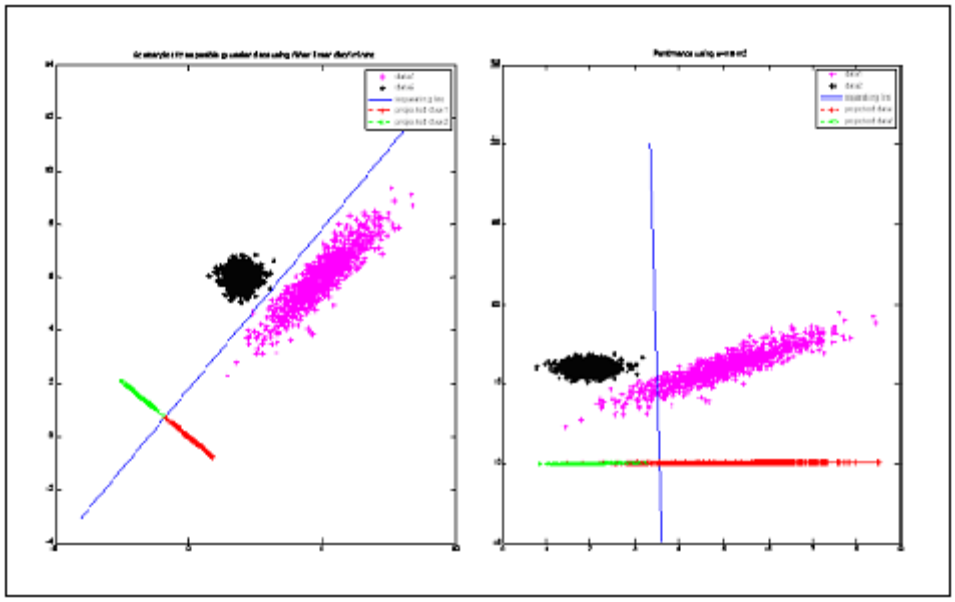

Fig2: Results & Discussion, Case 2

As shown in figure 2, the modified Fisher discriminant underperforms as compared to the fisher discriminant, as it draws a line of separation perpendicular to the one joining the means where that is not supposed to be the case. Hence this case exposes the lacuna present in the modified Fisher classifier technique.

Case 3

We now examine the case where we have 2-D OVERLAPPING data, and study the performances of both the classifiers.

	Class I	Class II
Mean	[2 8]	[6,5]
Variance	$\begin{pmatrix} 8.48 & -0.05 \\ -0.05 & 7.6 \end{pmatrix}$	$\begin{pmatrix} 8.64 & 0.15 \\ 0.15 & 8.53 \end{pmatrix}$

Method	Accuracy %
Fisher	83.25
Modified fisher	83.15

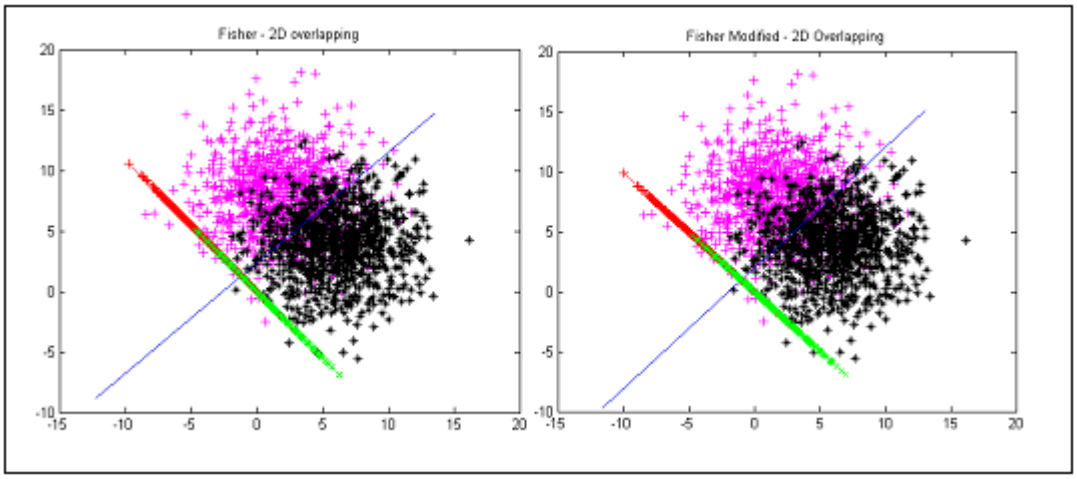

Fig3: Results & Discussion, Case 3

As shown in figure 3, the fisher and the modified fisher methods perform similarly on these overlapping data sets, with the accuracy of classification very close to each other. This is also partially due to the fact that the data here is such that the direction of minimum variance is the same as that joining the means of the data.

Case 4

All the previously dealt with cases were ones that used Gaussian data. We shall now experiment with 2-D uniform data. The following are the results obtained:

	Class I	Class II
Mean	[25 0]	[-25 0]
Variance	$\begin{pmatrix} 1026.9 & 6.3 \\ 6.3 & 16.9 \end{pmatrix}$	$\begin{pmatrix} 1027.1 & 4.4 \\ 4.4 & 17.1 \end{pmatrix}$

Method	Accuracy %
Fisher	83.97
Modified fisher	100

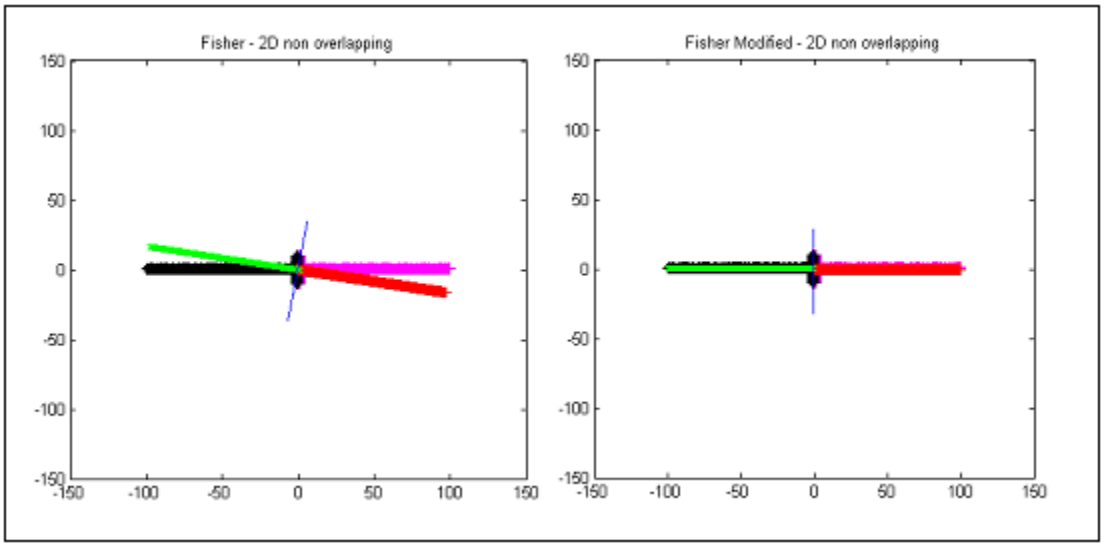

Fig4: Results & Discussion, Case 4

The data and its projection (enlarged view for the modified Fisher's method)

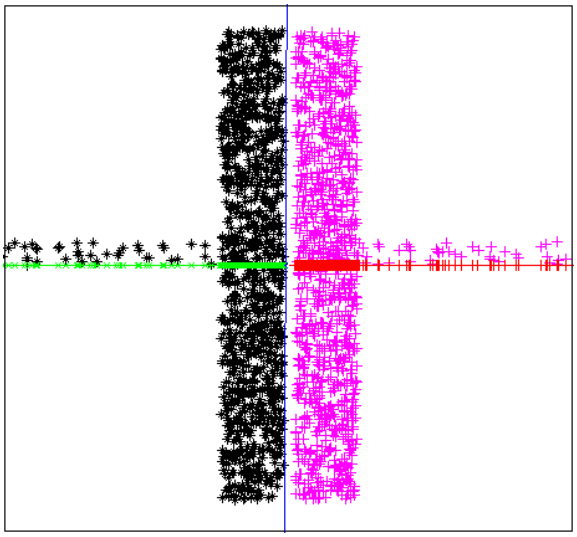

Fig5: Results & Discussion, Case 4

This is rather a peculiar case, the motivations for which arose from the fact that there would exist a particular case in which the modified fisher method would outperform the Fisher method. Hence two uniformly distributed data sets as shown in figure 5 were generated. These were

tested using both classifiers and the modified fisher discriminant outperformed its predecessor. This is because this case is tailor made for the modified Fisher method, in that, the line of separation is exactly perpendicular to the line joining the means of the data. This case however is of interest from the academic perspective alone, and rarely does one come across such data in real time.

Case 5

We shall now switch to 3-D, and record the results. We here deal with well separated data in which both yield similar results

	Class I	Class II
Mean	[1.0285 1.9437 3.0294]	[-0.9586 -2.0054 -2.9299]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Method	Accuracy %
Fisher	100
Modified fisher	100

Plotted below are the figures of both the discriminators. As usual, the fisher method is shown in the left and the modified method is on to the right. We see that both perform extremely well, as the data here is very well separated, as was in case 1.

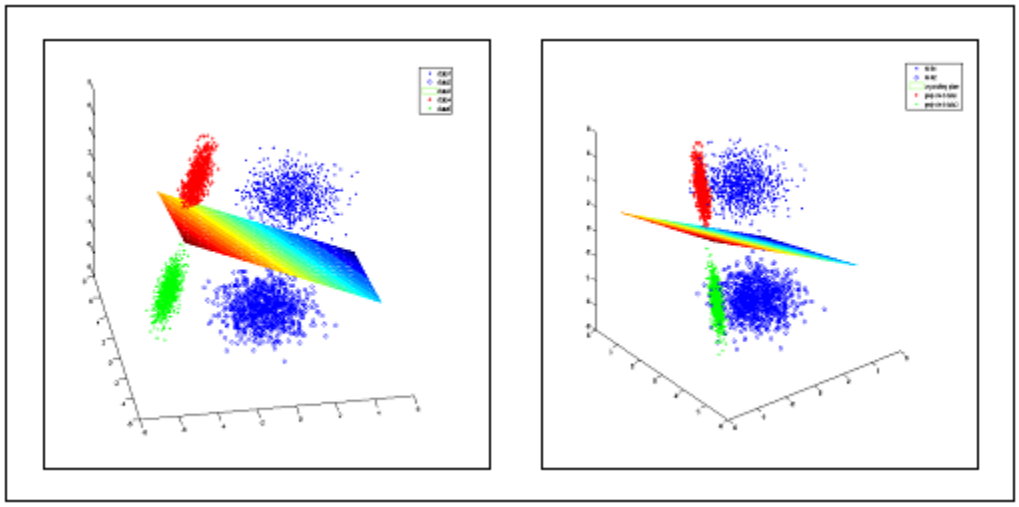

Fig6: Results & Discussion, Case 5

Case 6

We shall now examine cases in which we see that the Fisher method clearly outperforms the modified Fisher method.

	Class I	Class II
Mean	[0.9912 2.0148 3.0280]	[-0.9889 0.9819 1.9961]
Variance	$\begin{pmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.4 \\ 0.1 & 0.4 & 0.9 \end{pmatrix}$	$\begin{pmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.4 \\ 0.1 & 0.4 & 0.9 \end{pmatrix}$

Method	Accuracy %
Fisher	89.35
Modified Fisher	88.15

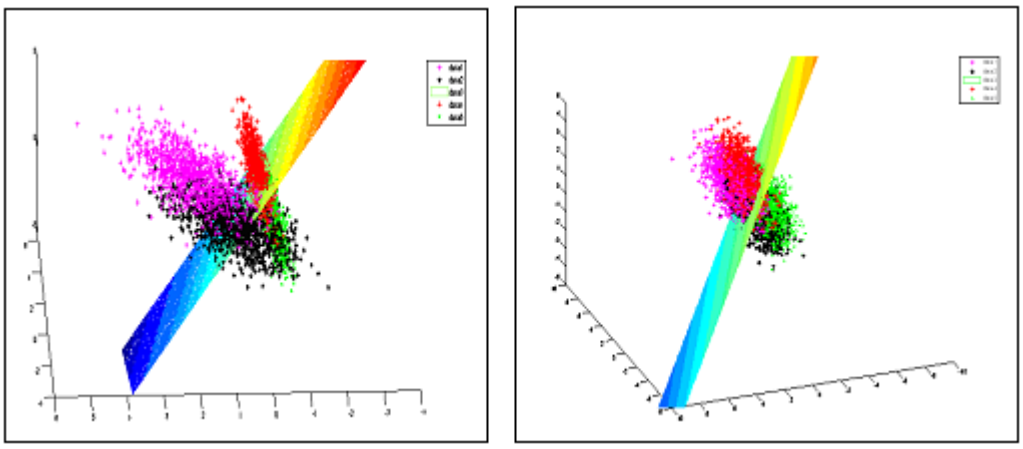

Fig7: Results & Discussion, Case 6

The above figures show the performances of the two methods for overlapping data sets. The next example will display the superiority in the performance of Fisher, for non-overlapping data as well.

	Data 1	Data 2
Mean	[0 0 0]	[2 2 0]
Variance	$\begin{pmatrix} 7 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.08 \end{pmatrix}$	$\begin{pmatrix} 0.08 & 0 & 0 \\ 0 & 0.08 & 0 \\ 0 & 0 & 0.08 \end{pmatrix}$

Method	Accuracy %
Fisher	100
Modified Fisher	89

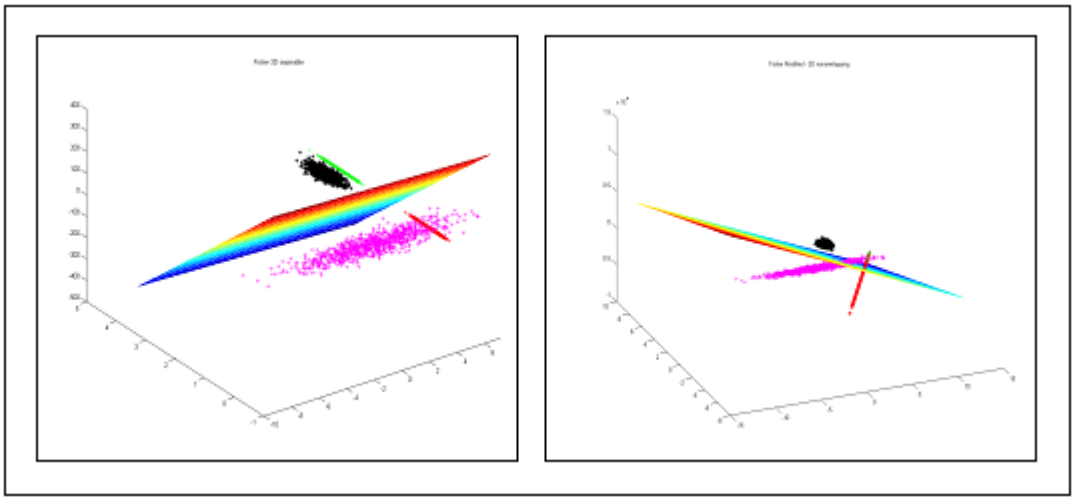

Fig8: Results & Discussion, Case 6

The above figures show the performances of the two methods for non-overlapping data sets, from which we clearly see that the Fisher method surpasses the modified fisher method in performance.

To conclude, one can say that from the experiments performed, it is possible to say that the Fisher method is the superior to the modified Fisher method. Although we have shown a case which runs contrary to the previous conclusion, such cases are extremely rare to occur, and even in this case, it was tailor-made for the sake of experimentation. Hence we can finally conclude that the Fisher linear discriminant is superior to its modified version.

The following segment shows the MATLAB routine for this experiment.

MATLAB CODE:

```
%Fisher Discriminant function for 2 and 3D data. X1 , X2 are data vectors
%with each row being a feature vector . d is the dimesnsion size and k is
%the offset for plotting in 3-D

function fisher(X1,X2,d,k,rt)

%variables to hold size of input data
[m n]=size(X1);
m1=zeros(1,d);
m2=zeros(1,d);

%Checking for data's dimensionality
if(d==3)
plot3(X1(:,1),X1(:,2),X1(:,3),'r+:');
hold on
plot3(X2(:,1),X2(:,2),X2(:,3),'g*:');
end

if(d==2)
plot(X1(:,1),X1(:,2),'m+');
hold on
plot(X2(:,1),X2(:,2),'k*');
end


m1=mean(X1)
m2=mean(X2)
X=[X1;X2];

%glm stores the global mean
glm=mean(X)
S1=cov(X1)
S2=cov(X2)
SW=S1+S2;

%Compute Weight function
w=(SW^-1)*(m1-m2)'; %change this to (m1-m2)' for modified fisher

if(d==3)
plot3(w(1)/(sqrt(w(1)^2 + w(2)^2 + w(3)^2)),w(2)/(sqrt(w(1)^2 + w(2)^2 +
w(3)^2)),w(3)/(sqrt(w(1)^2 + w(2)^2 + w(3)^2)),'X');
end
if(d==2)
    plot(w(1)/(sqrt(w(1)^2 + w(2)^2)),w(2)/(sqrt(w(1)^2 + w(2)^2)),'X');
end
magw=sqrt(w'*w);
vecw=w/magw;


if(d==3)
vecw2=[-vecw(2) vecw(1) 0];
mag2=sqrt(vecw2*vecw2');
vecw2=vecw2/mag2;
```

```
end

if(d==3)
for i=1:m
    vec1(i,:)=X1(i,:)-(X1(i,:)*vecw2')*vecw2;
    vec2(i,:)=X2(i,:)-(X2(i,:)*vecw2')*vecw2;
end
end

%variables used for plotting
xmin=min(X(:,1));
xmax=max(X(:,1));
ymin=min(X(:,2));
ymax=max(X(:,2));

%plots the plane in between classes
if(d==3)
    x=[xmin:0.1:xmax];
    y=[ymin:0.1:ymax];
    for i=1:length(x)
        for j=1:length(y)
    z(i,j)=(-w(1)*x(i)-w(2)*y(j) +glm*w)/w(3);
        end
    end
[p q]=size(z);
mesh(y,x,z);
end

%Computes number of misclassified points
if(d==2)
for i=1:m
    vec1(i,:)=(X1(i,:)*vecw)*vecw';
    vec2(i,:)=(X2(i,:)*vecw)*vecw';
end
end
misc1=0;
misc2=0;
if(d==3)
    for i=1:m
        t1(i,:)=vec1(i,:)+k*vecw2;
        t2(i,:)=vec2(i,:)+k*vecw2;
        if(X1(i,:)*w - glm*w < 0)
            misc1=misc1+1;
        end
        if(X2(i,:)*w - glm*w >0)
            misc2=misc2+1;
        end
    end
end

if(d==3)
plot3(t1(:,1),t1(:,2),t1(:,3),'+','Color','b');
plot3(t2(:,1),t2(:,2),t2(:,3),'x','Color','g');
end

%Compute accuracy
```

```
misc1=0;
misc2=0;
w0=glm*vecw
r=w0*vecw';
if(d==2)
    plot(r(1),r(2),'X');
    line([(r(1)-k*vecw(2)),(r(1)+rt*vecw(2))],[(r(2)+k*vecw(1)),(r(2)-
rt*vecw(1))]);
    plot(vec1(:,1),vec1(:,2),'b+:');
    plot(vec2(:,1),vec2(:,2),'gx:');
    for i=1:m
        if(X1(i,:)*w - glm*w < 0)
            misc1=misc1+1;
        end
        if(X2(i,:)*w - glm*w >0)
            misc2=misc2+1;
        end
    end
    accuracy=100*(1-(misc1+misc2)/(2*m))
end
```

Question II

Aim:

To obtain a set of training data and divide the training data into training data and test data, and experiment with designing a classifier using the neural network approach and the support vector machine approach, and to compare the results so obtained.

Introduction:

Neural Networks are powerful, biologically inspired tools used for pattern classification. When we talk of neural networks in the context of pattern classification, we often mean artificial neural networks, or ANN, that are made of interconnected artificial neurons. These neurons mimic the performance of the biological neurons.

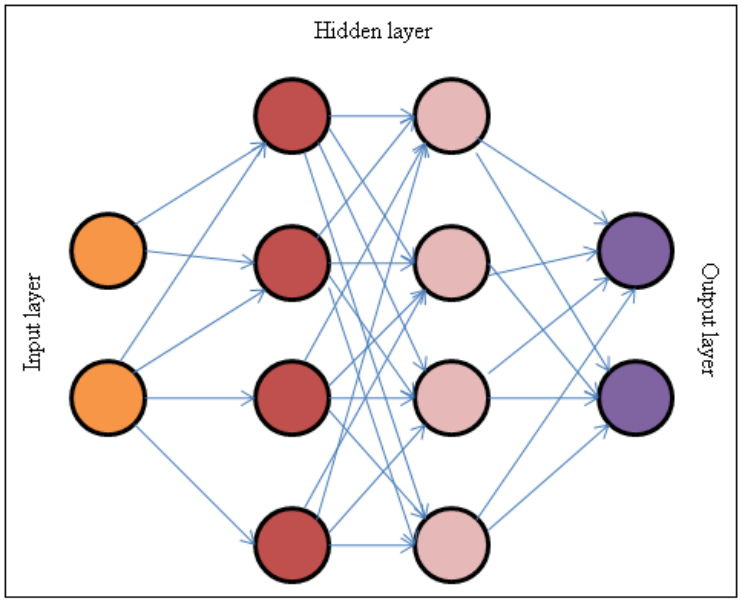

Fig9: Artificial neural networks

Another very important tool used for pattern classification and supervised learning methods used for classification and regression, that belongs to the family of generalized linear classifiers is the Support vector machine (SVM). They can also be considered a special case of Tikhonov regularization. The philosophy behind the working of SVM is to minimize the empirical classification error and maximize the geometric margin, by the virtue of which it is alternatively termed as maximum margin classifier.

Procedure:

Before we start explaining the experiments, let us have a first-look at the neural networks and support vector machine tools that are available in Matlab. These tools were used in the process of experimentation.

SVM tool in Matlab®.

Described below are the commands used in the experiments:

Command	Input	Functions	Comments
svmtrain	-test vectors, -the group they belong to -the kernel function -the order of the polynomial (for polynomial kernel) -display plot option	-show-plot option plots the data, labels the classes, and draws the separation hyper-surface	Of the several kernel functions that are available, we have experimented with Radial Basis function and the polynomial kernel of varying orders.
svmclassify	-structure returned by the above function	-classifies the data as belonging to either class 1 or 2, and returns a vector containing classes of test data	-The vectored that is returned is used to measure the accuracy of the SVM.

Neural Network tool in Matlab®

Matlab® provides a Graphical User Interface (GUI) Tool, for working with neural networks. It works on the basis of accepting inputs from the user, in the form of data and parameters. This provides an extremely flexible tool to the user, wherein several parameters of the network can be changed, so that the desired performance can be achieved. Some of the parameters that can be modified by the user are the activation function, the initial weights, the training and test data, the number of epochs and the target data for the given training set.

The user inputs the training data, which is a *(d* x *n)* matrix, where the dimension of the data is given by d, and the number of training vectors is given by n. The target data is actually the data

that contains the values that ought to be the output upon correct classification. The error is calculated as the difference between the actual output and the target output. Once this data is given we can "create" the network by specifying the following:

- Type of network
- Training function
- Adaption Learning Function
- Number of layers
- Number of neurons per layer
- Transfer Function for neurons.

Let us look at some of the snapshots of the nntool in Matlab.

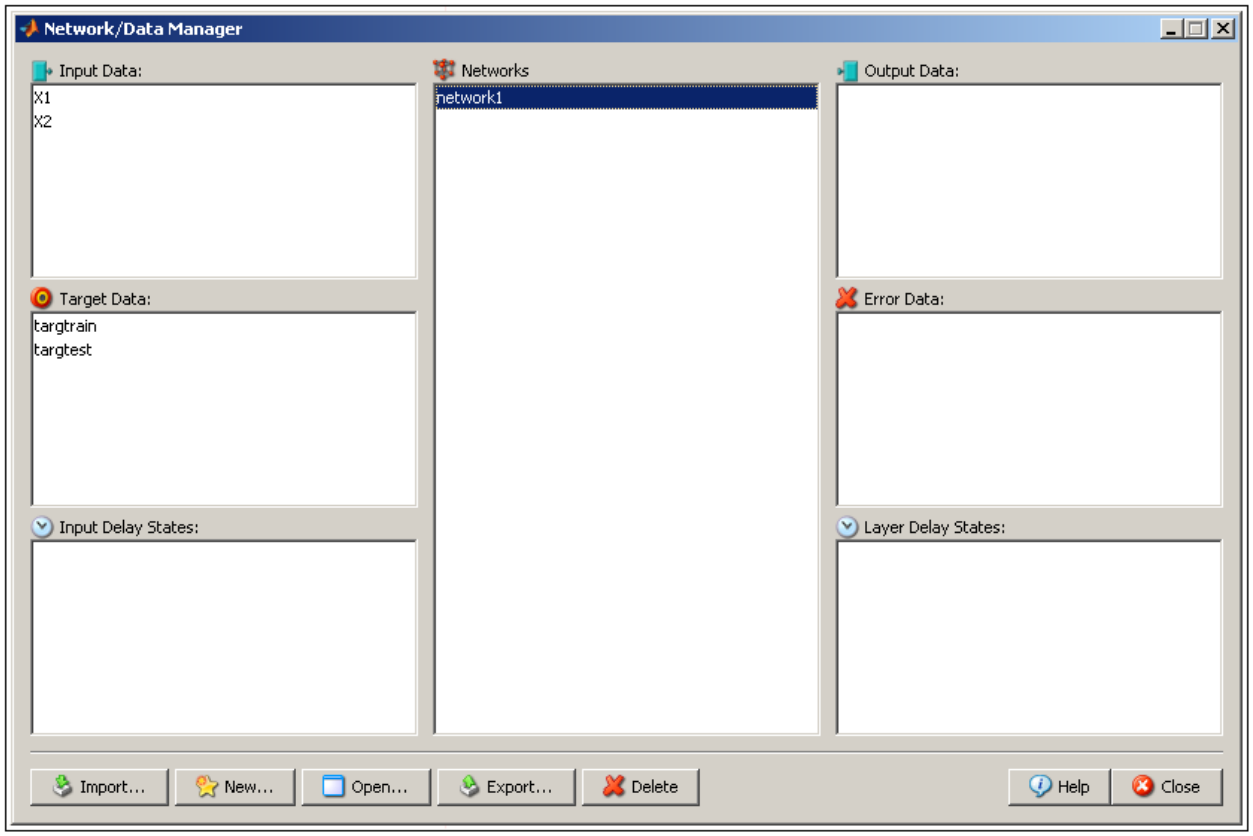

Fig10: NNtool in Matlab

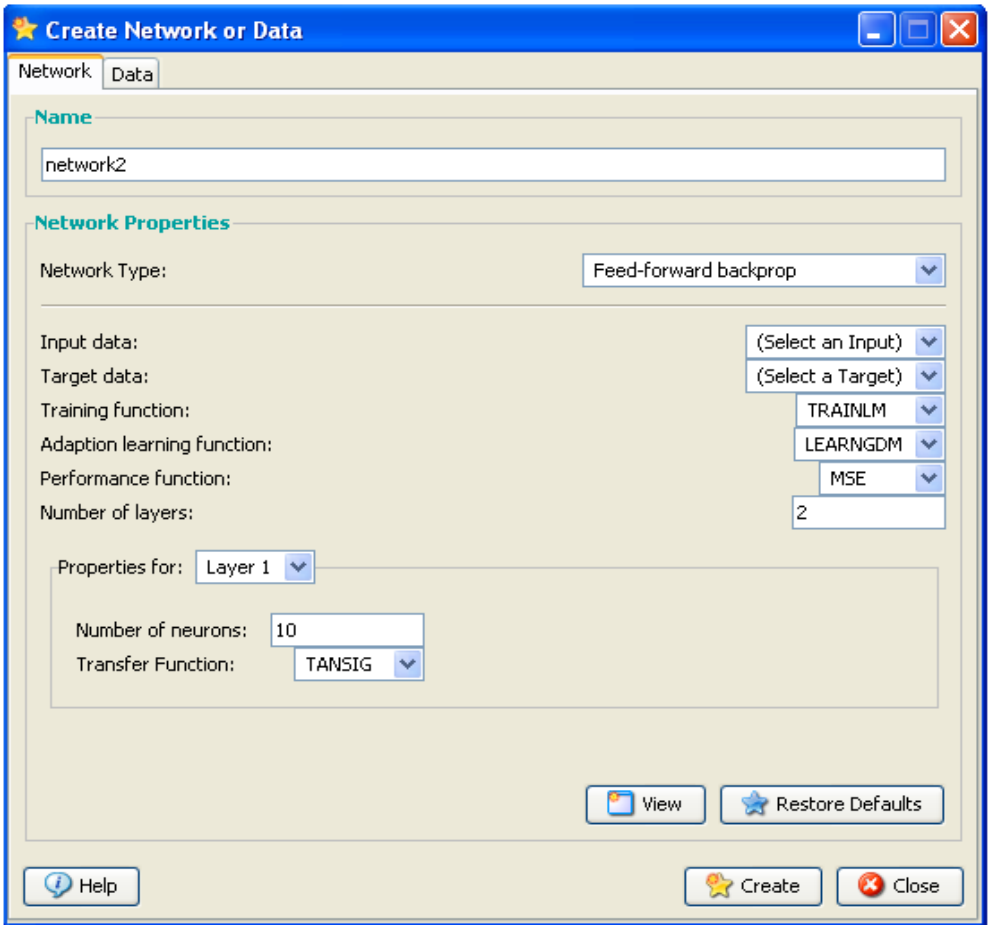

Fig11: NNtool in Matlab

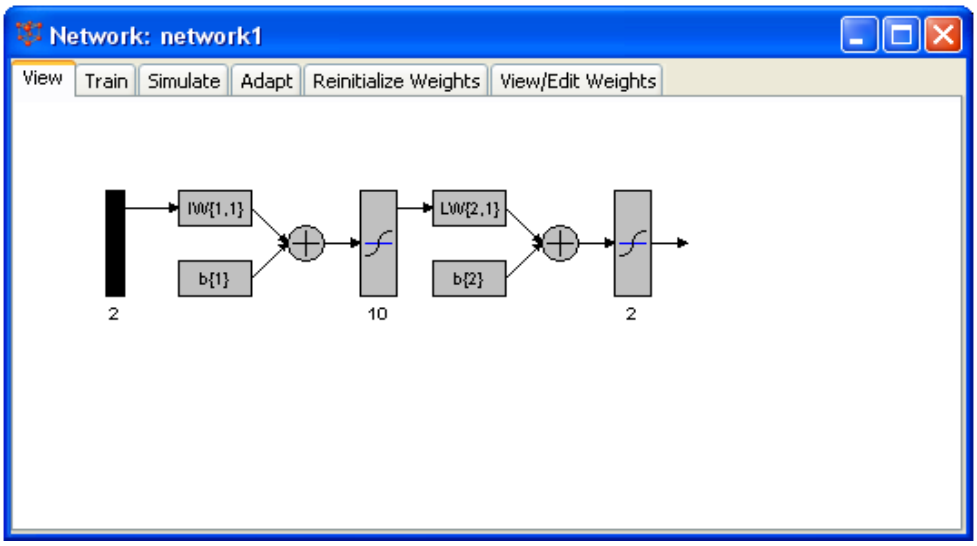

Fig12: NNtool in Matlab

The following table gives us the details of the parameters used during the experimentation process. Parameters such as the number of hidden layer neurons and the number of time-epochs used for convergence were varied throughout the experiment.

Network type	feed forward back propagation
Training function	LM function
Adaption Learning function	LEARNGDM function
Error performance	Mean Square Error
Activation function	Tan-sigmoid
# hidden layer neurons	Varying throughout the experiment
# time-epochs	Varying throughout the experiment

Results and Discussion:

Experiments were conducted to evaluate the performance of both classifiers. Initially, 1000 points of each class were generated. Next, these data points were split as training data and testing data, and the percentage split was varied as 10-90, 35-65, 65-35 and 90-10 respectively. For SVM, the performance evaluation was done for different kernels. As for the neural networks, the number of neurons present in the hidden layer was varied. Shown below are tables comprising of data for SVM, followed by figures. Subsequently, we also take a comparative look at the performances of neural networks and SVM.

Case 1

We consider here 2-D data that are overlapping. The table below shows the means and variances of the two classes, and this is followed by the plots for SVM classification.

	Class I	Class II
Mean	[1 1]	[4 3]
Variance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$

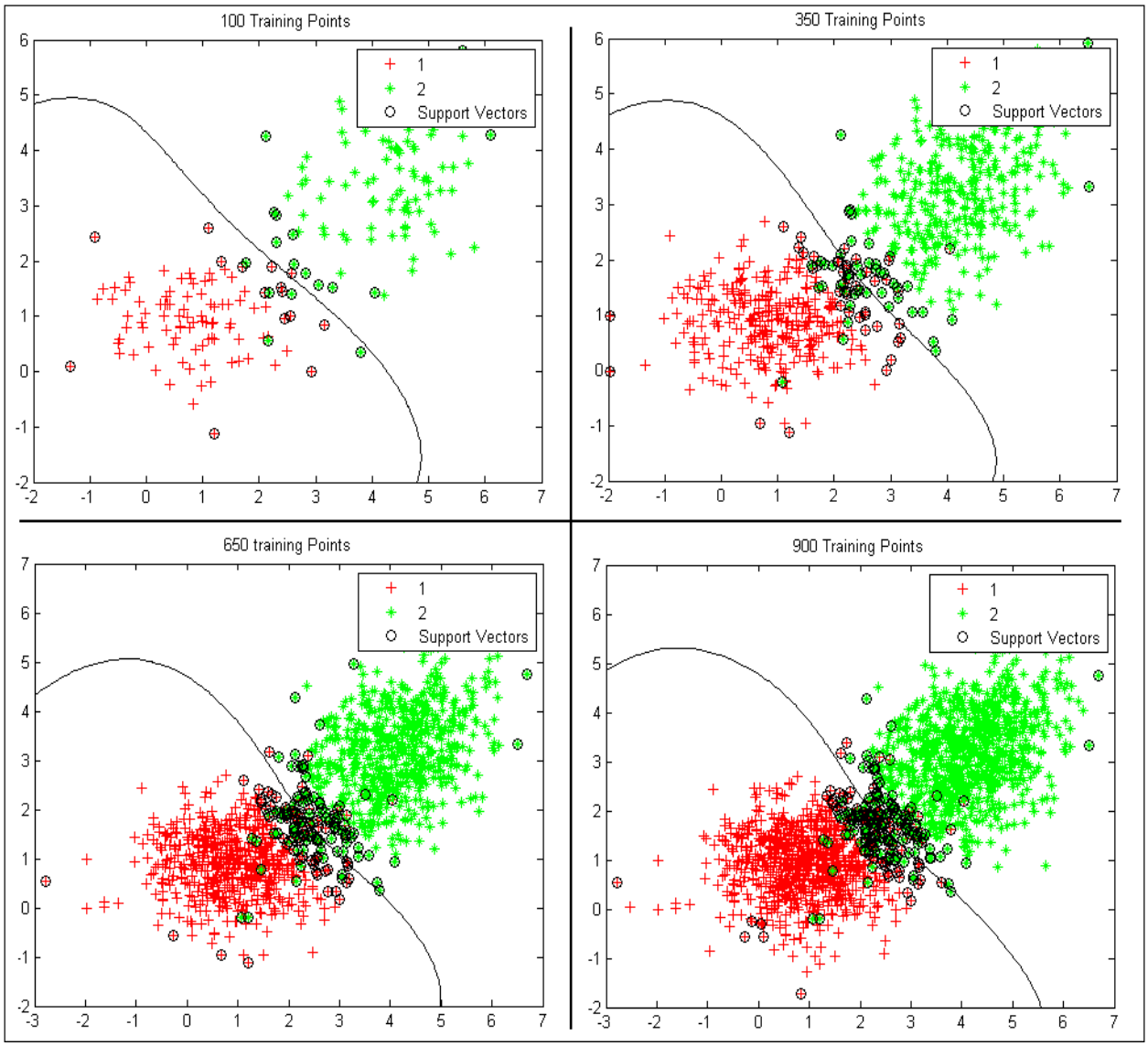

Fig13: Training data and support vectors for different proportions of training data using **Radial Basis Functions**

# of Training Samples	100	350	650	900
Accuracy Neural Network (5 neurons in hidden layer)	93.72	95.42	96.85	94.83
Accuracy- RBF Kernel	94.9444	94.8462	95.4286	96
Accuracy 2nd degree poly. Kernel	94.6667	94.7692	94.1429	94.5000
Accuracy 3rd degree poly. Kernel	94.5556	94.6154	94.2857	95

A look at the data chosen above would reveal that the data have separated means, but the variances chosen result in an overlap between the classes. Classification using Support Vector machines results in an accuracy percentage in the range of 94-96%, which is fairly high. Also, there is no appreciable difference in the performances of SVM using RBF and the polynomial kernel (of order 2 and 3). Thus, we can say that for data sets that don't have much of an overlap, SVM performs well, giving good accuracy rates.

For the case of ANN with 5 neurons, the accuracy rate starts at 93% for a training set of 100 points, increases gradually with an increase in the amount of training data to around 97%, and then dips when the amount of training data increases drastically to 900 points. This is understandable, as it means that the performance improves with the increase in training data, as the system becomes "more acclimatized" to the data set, and starts performing better. However, with a huge training data set, the network faces what is known as "overtraining hazard", because the network starts "following" the data curve too closely. Further, although not quantified, we also observed that the neural network took appreciably more time to converge than its SVM counterpart. This can be considered a potential glitch for problem involving large data sets, or when the margin of error for convergence is very small.

Case 2

We now consider the case in which we have 2D data sets that are more overlapping, as a consequence of having closely located means.

	Class I	Class II
Mean	[1 1]	[2 2]
Variance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 0.7 & 0.21 \\ 0.21 & 0.56 \end{pmatrix}$

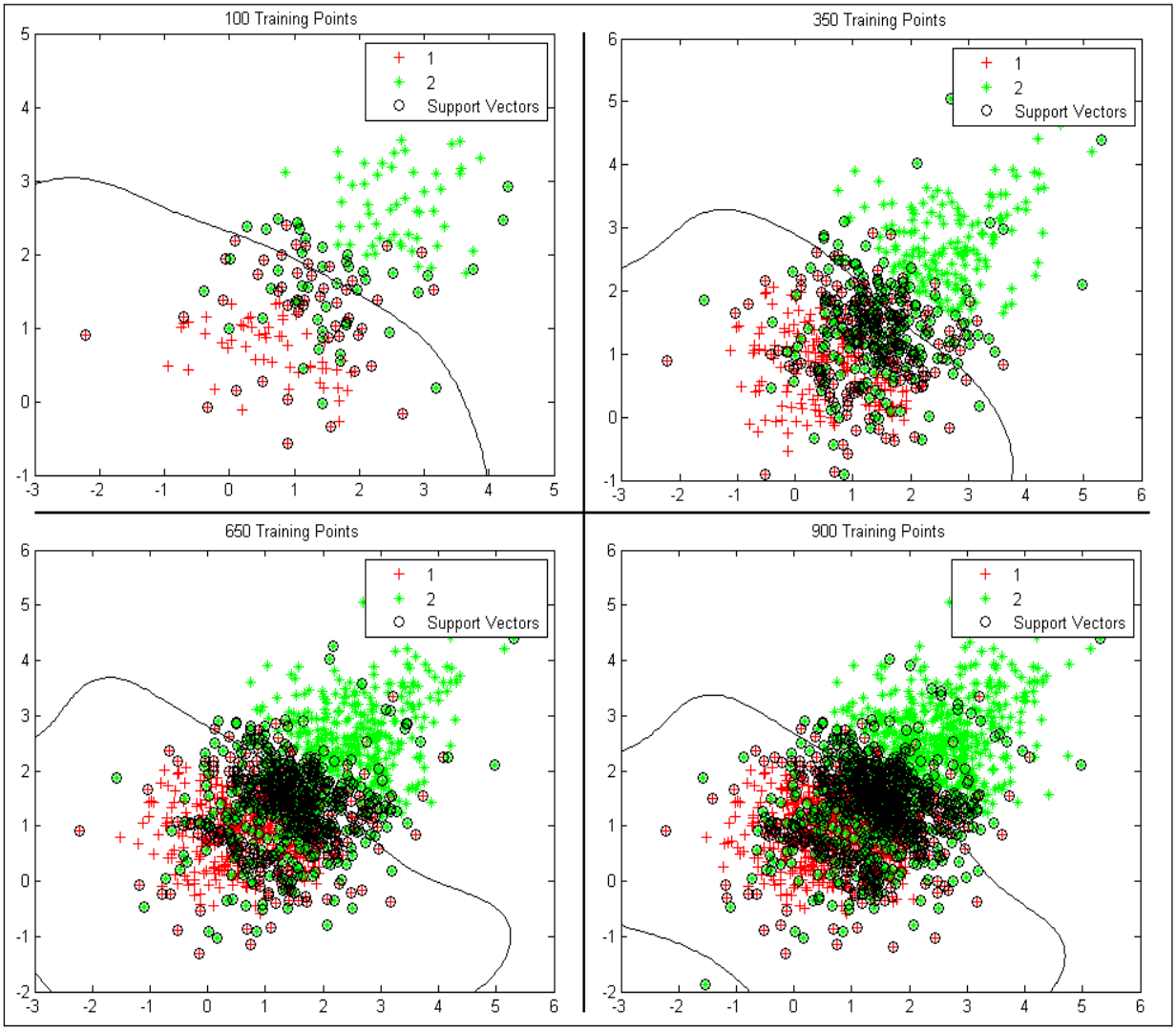

Fig14: Training data and support vectors for different proportions of training data using **Radial Basis Functions**

# of Training Samples	100	350	650	900
Accuracy Neural Network	76.67	77.61	80.14	50.0
Accuracy - RBF Kernel	74.7222	75.0769	75.7143	78
Accuracy 2^{nd} degree poly. Kernel	74.8333	74.6923	73.7143	77
Accuracy 3^{rd} degree poly. kernel	74.0556	73.5385	NC	NC

The motivation behind conducting this experiment was to see the performance of the tools when we had overlapping data. In this case support vector machines with radial basis functions have a accuracy between 74 and 78%. Similar performance was obtained for polynomial kernels of 2^{nd} and 3^{rd} order. However, we were unable to achieve a convergent value for 3^{rd} order polynomial kernel operating on 650 and 900 training data points, owing to the inability of the system to handle large amounts of data. As for the case of neural networks with 5 neurons, a steady increase in performance is seen with an increase in training data, and as expected, a dip in the performance for extremely large values of training data.

As is seen above, neural networks perform poorly as compared to SVM for an overly trained scenario of 90%. However, it performs better that SVM for 65% training data usage. But this bettered performance comes at the price of increased time requirements in that, neural networks take a considerable larger amount of time than SVM, which can be an important set-back in certain applications.

Case 3

We shall now see some more additional cases, by changing the distribution of data, and we shall also consider the 3-D case.

	Class I	Class II
Mean	[1 1 1]	[3 3 3]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

# of Training Samples	100	350	650	900
Accuracy Neural Network	85.55	87.38	88.0	86.32
Accuracy SVM(RBF Kernel)	88.38	89.30	89.29	92.50

Here we consider 3D data with more overlapping, by just increasing the variance.

	Class I	Class II
Mean	[1 1 1]	[3 3 3]
Variance	$\begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 2.5 & 0 \\ 0 & 0 & 5.5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

# of Training Samples	100	350	650	900
Accuracy (RBF Kernel)	87.78	88.85	88.29	87.50

Finally, we consider 3-D data with more overlapping, by shifting the means.

	Class I	Class II
Mean	[2 2 2]	[1 1 1]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

# of Training Samples	100	350	650	900
Accuracy Neural Network	68.16	69.31	72.0	71.33
Accuracy (RBF Kernel)	72.22	70.85	72.14	74.50

Case 4

We shall now consider a very interesting case, in which we deal with sets of data that are peculiarly oriented. Motivations for performing such an experiment arose from [2], where Stanislaw Osowski et.al have compared the performance of SVM and neural networks for data sets that are oriented spirally. Experiments using such data sets were performed by varying the angle of orientation, and also by varying the parameters of the neural network. Shown below are the specifications of the data, followed by the concerned plots for SVM and neural networks.

Initially, experiments were performed by varying the training data without sampling. By this we mean that if 100 training points are supplied, these 100 points are the first 100 points of the data, and NOT ones obtained by sampling the data at fixed intervals.

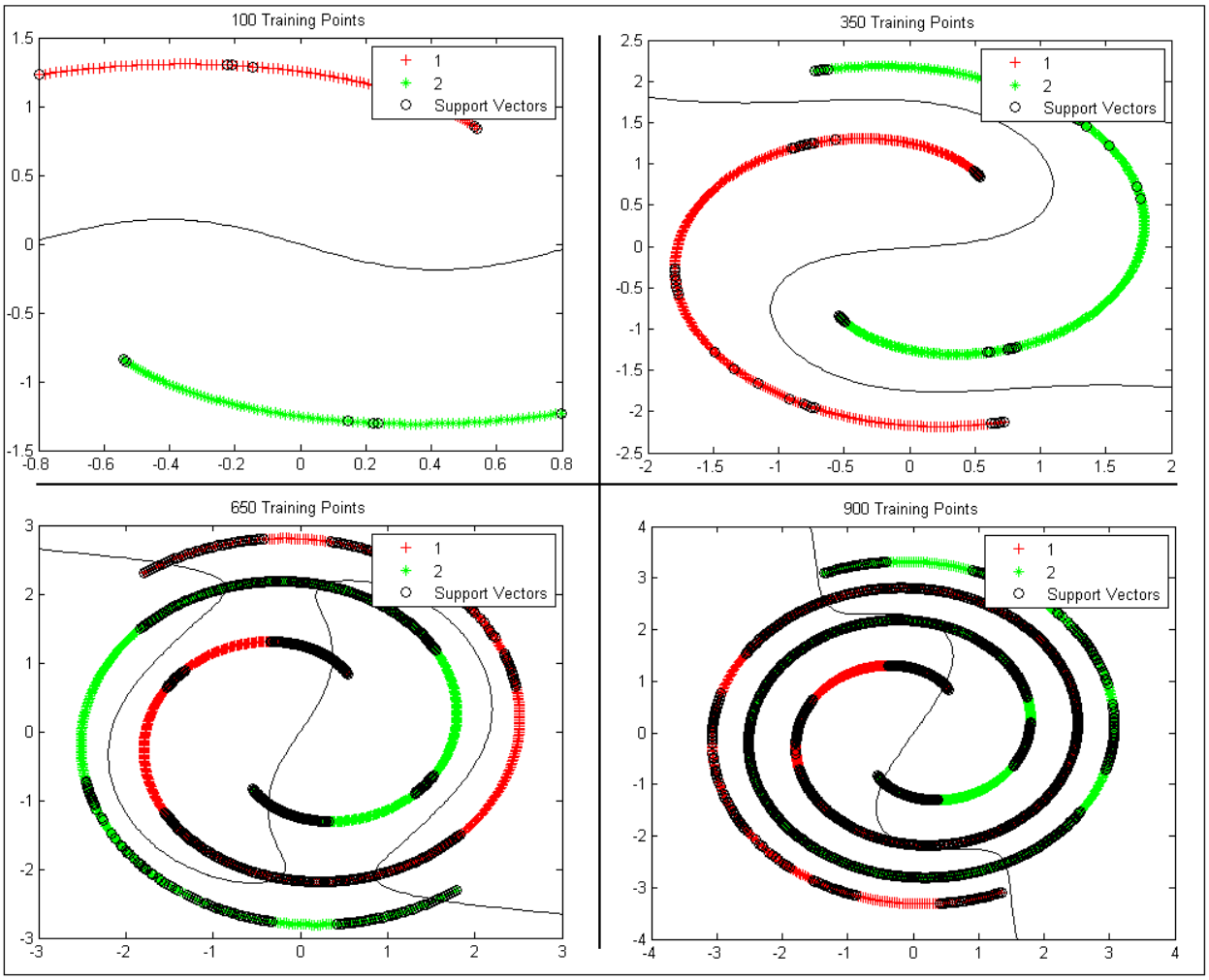

Fig15: Training data and support vectors for different proportions of training data using **Radial Basis Functions**

# of Training Samples	100	350	650	900
Accuracy Neural Network	41.61	45.07	30.71	33.0
Accuracy RBF Kernel	47.89	48.15	20.86	2.50

As seen in the table above, the accuracy of both the neural network and SVM is very poor. In particular, we note the performance of the SVM at 900 training data points. This gives almost 100% error because, from the figure shown above, the test data points of class one almost fill up the left of the discriminating hyper-surface, while the test points of class 2 fill up the right side, which means that the classification becomes entirely wrong for this case.

These shortcomings in classification is largely because of the fact that we have used test and training data without sampling the given data, which means we leave a lot for prediction and chance, and the training data does not serve its purpose to "train" the network.

Case 5

We here deal with the case wherein the input data is sampled first, and then split as training and test data. For example, if we need 100 training points, we sample the input data at one for every ten data points, thereby obtaining 100 data points for a data set of size 1000 points. Let us now examine the efficiency of this method.

First we take every 10th sample from the data set, and next, we take every 20th sample from the data set, and the experiments are performed with polynomial kernel and RBF.

For angle 4π:

Fraction of samples	1/10	1/20
Accuracy Neural Network 25 hidden neurons	99.94	99.95
Accuracy Neural Network 10 hidden neurons	82.77	75.47
Accuracy with 8^{th} degree poly Kernel	99	94.21
Accuracy with RBF Kernel	55.78	56.63

Using a 7th degree polynomial kernel, the accuracy was 98.78%. However, below this, poor accuracy rates were obtained. This goes on to show that an increase in the spiral angle demands higher degree polynomials for robust error performances.

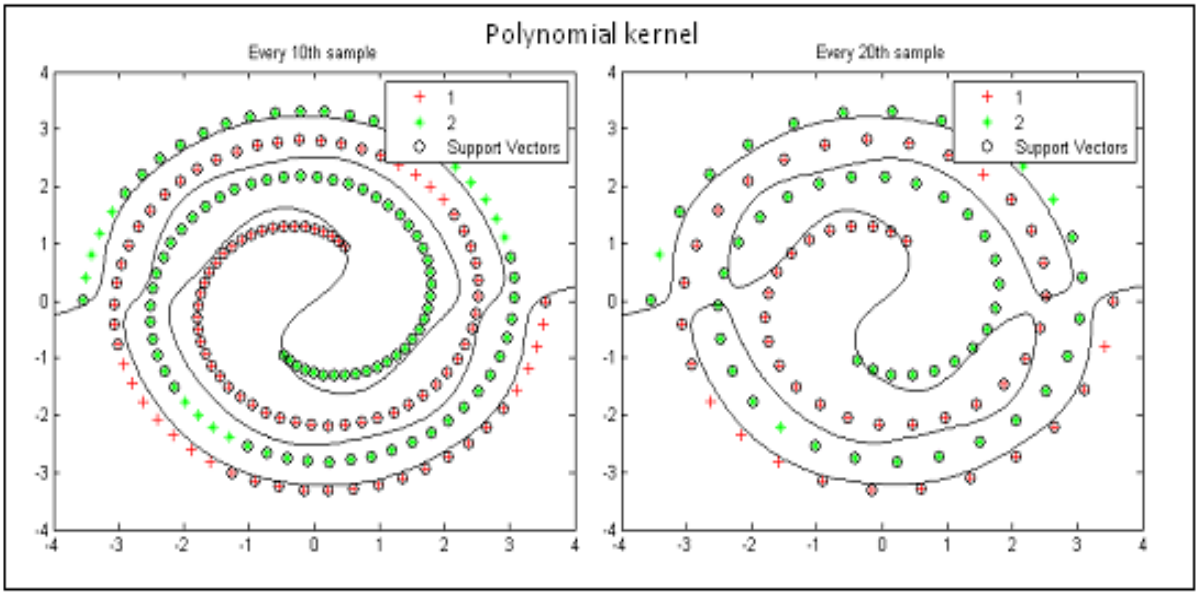

Fig16: Training data and support vectors for different proportions of training data using **polynomial Functions**

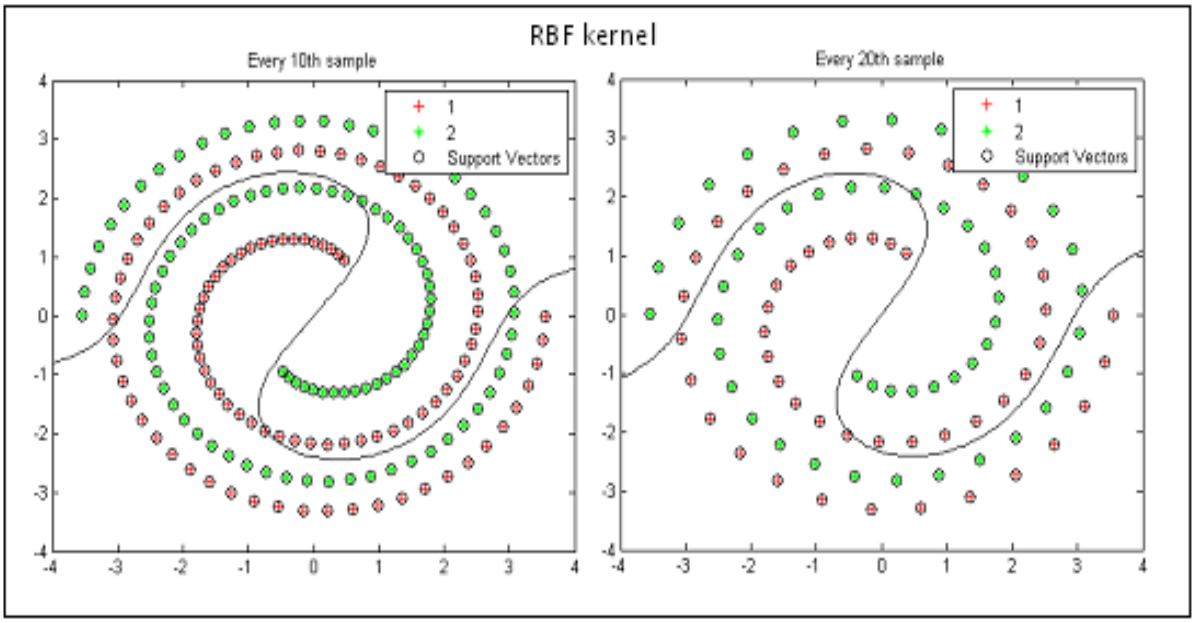

Fig17: Training data and support vectors for different proportions of training data using **Radial Basis Functions**

For angle 2.5π:

Fraction of Samples	1/10	1/5	1/2
ANN acc. with 25 hidden neurons	99.94	99.94	99.94
Accuracy- Neural Network	66.89	76.23	81.33
Accuracy- RBF Kernel	86.50	95.25	96.50

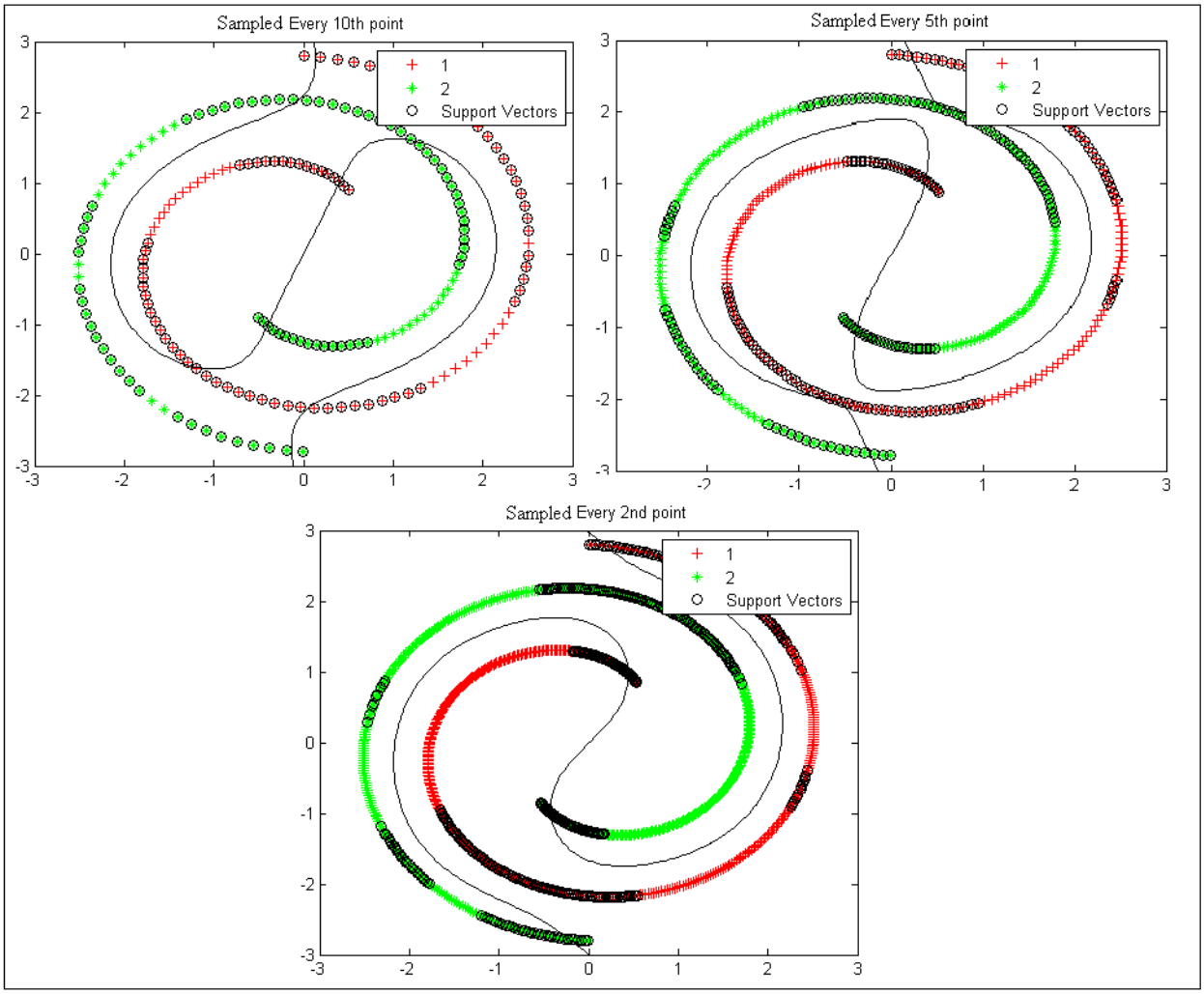

Fig18: Training data and support vectors for different proportions of training data using **Radial Basis Functions**

Finally, let us have a look at the snapshots of the GUI tools in Matlab that appear as the output, by making a sample run of NNtool in Matlab.

Error Convergence Curve:

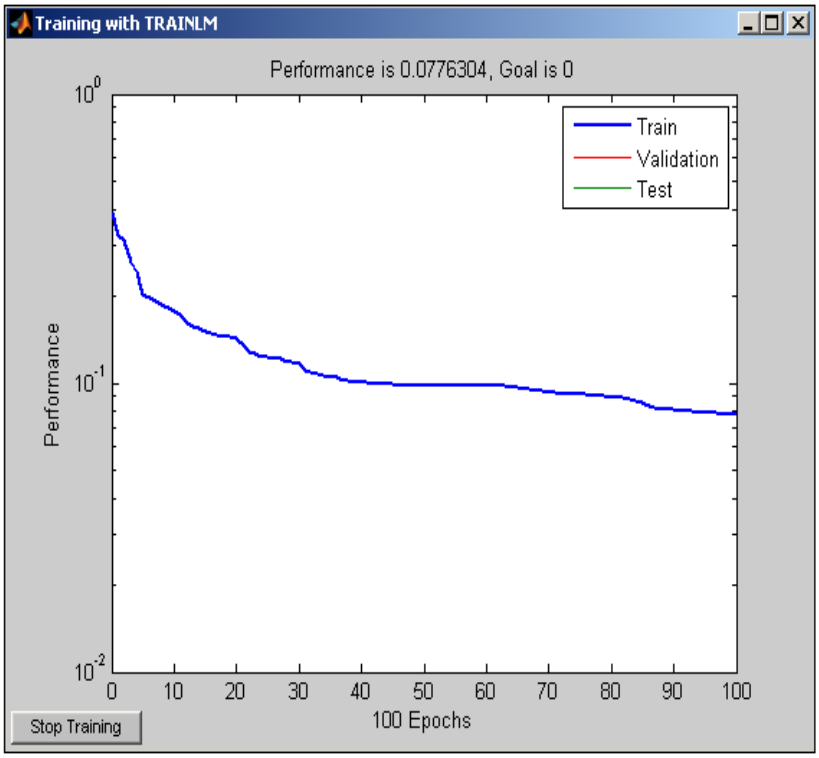

Fig19: NNtool in Matlab

The Neural Network that gets generated based on input parameters:

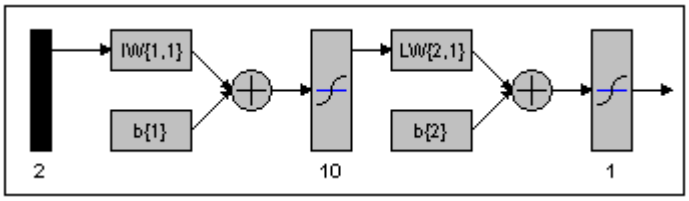

Fig20: NNtool in Matlab

To conclude, one can say that SVM performs better as compared to neural networks operating upon with lesser neurons. Increase in number of neurons can contribute to a significant increase in the performance of the neural network, but this comes at the expense of increased operating time. Also, we find that SVMs do not depend on the dimensionality of the data under consideration, as compared to ANNs. Further, in SVM, the polynomial kernel function performs

better than the radial basis function, because the former has greater degrees of freedom and has a greater chance of developing complex curves as compared to the radial basis function.

The following segment shows the MATLAB routine for this experiment.

MATLAB CODE:

```
%Script for running SVM
clear all
close all
clc

X2=mvnrnd([2 2 2],[1 0 0; 0 2 0; 0 0 5],1000);
X1=mvnrnd([1 1 1],[1 0 0; 0 2 0; 0 0 5],1000);

trn1=350;
trn2=350;
Xtrain=[X1(1:trn1,:);X2(1:trn2,:)];
class=[ones(trn1,1);1+ones(trn2,1)];

s=svmtrain(Xtrain,class,'Method','SMO','Kernel_function','rbf','showplot',1);

Xtest=[X1(trn1+1:1000,:);X2(1+trn2:1000,:)];
p=svmclassify(s,Xtest)

q=[ones(1000-trn1,1);1+ones(1000-trn2,1)];
miscl=sum(abs(p-q));
accuracy=100-100*miscl/(1000-trn1+1000-trn2)
```

```
%Neural network script file for spiral data
clear all
close all
clc

%Creating a variable running from 0-4pi
t=linspace(1,4*pi,1000)';
r1=sqrt(t);
r2=-sqrt(t);

%Variables to create spiral path
for i=1:length(t)
    x1(i)=r1(i)*cos(t(i));
    y1(i)=r1(i)*sin(t(i));
    x2(i)=r2(i)*cos(t(i));
    y2(i)=r2(i)*sin(t(i));
end

X1=[x1' y1']';
X2=[x2' y2']';

%Preparing a sampled set of train-data from original data
```

```
j=1;
k=1;
for i=1:length(X1)
    if(mod(i,10)==0)
        X1_new(:,j)=X1(:,i);
        X2_new(:,j)=X2(:,i);
        j=j+1;
    else
        X11_new(:,k)=X1(:,i);
        X21_new(:,k)=X2(:,i);
        k=k+1;
    end
end
plot(X1_new(1,:),X1_new(2,:),'x')
hold on
plot(X2_new(1,:),X2_new(2,:),'ro')

%Prepare a set of testing data and target data
ntrain = length(X1_new);
dim = 2;
traindata = [X1_new X2_new];
testdata = [X11_new X21_new];
ntest = length(X1)-ntrain;
targtrain = [ones(1,ntrain) 1+ones(1,ntrain)];
targtest = [ones(1,ntest) 1+ones(1,ntest)];

%Compute number of misclassified points
sample_outputs=round(network1_outputs);
count=0;
for i=1:ntest
    if(sample_outputs(i)==2)
        count=count+1;
    end
end
for i=ntest:length(targtest)
    if(sample_outputs(i)==1)
        count=count+1;
    end
end

%Accuracy
accuracy = 100*(length(testdata)-count)/length(testdata)
```

Question III

Aim:

To design classifiers using Parzen window, K-nearest neighbor and nearest neighbor techniques using the same data as that used in the previous task, and compare the performances of the three approaches.

Introduction:

The Parzen window technique is one of the very robust techniques available for pattern classification. It is one of those techniques that compute the probability density function of the data for data classification. In this technique, we define a window function φ, which has a value only within a particular range, and is zero elsewhere. Using this, we calculate the estimate the densities using the following equation as

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{v_n}\varphi\left(\frac{x - x_i}{h_n}\right) \tag{2}$$

Here h_n controls the size of the window, and has an important bearing on the performance if a rectangular window is used.

The k-nearest neighbor (KNN) technique is a very simple yet effective technique for pattern classification. Objects are classified based on a majority vote of its neighbors, such that the object is assigned to that particular class which is most commonly present amongst its neighbors. K is a positive integer, and is usually kept small. In the event of k=1, this algorithms gets transformed into the nearest neighbor algorithm, which is a special case. Usually, it is advisable to keep K as an odd number, so that conflicts will not arise due to an equal number of votes for either class.

Procedure:

While performing experiments with Parzen windows, both rectangular and Gaussian windows were taken into consideration. It is instructive here, to explore the case where no points fall inside the window for a given h_n, or if there are an equal number of points from either class

within the window. In such cases, conflicts are usually resolved using priori probabilities of each class. However, since we assume here that both classes under consideration have equal priori probabilities, we resolve conflicts using a flip of a coin. This is practically done using a random number generator, that generates randomly one of two values (1,2). We decide on the class of the object based on the outcome of this RV generator.

Results and Discussion:

Let us now discuss the results so obtained for the experiments conducted. The remainder of this section is organized as follows. We first develop the results for Parzen windows technique, followed by the k-nearest neighbor technique. The nearest neighbor technique is discussed as a special case.

Case 1

First consider using the Parzen Window Technique for 2-D well separated data. Experiments for different values of "window" size were conducted for both Gaussian and Rectangular windows. Following are the results:

	Class I	Class II
Mean	[1 1]	[4 3]
Variance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$

%training / H	10		35		65		90	
	G	R	G	R	G	R	G	R
0.15	95.83	94.05	95.77	95.0	96.14	95.14	95.0	94.5
0.30	95.89	90.06	96.07	94.38	96.29	94.00	95.0	93.5
0.50	95.94	78.17	96.0	88.39	95.71	91.42	95.5	93.0
0.75	95.78	62.28	95.85	76.31	95.14	82.29	94.0	84.0

The following are the plots generated for the h=0.75. We have plotted the classification of data for varying proportions of training data, for both rectangular and Gaussian windows.

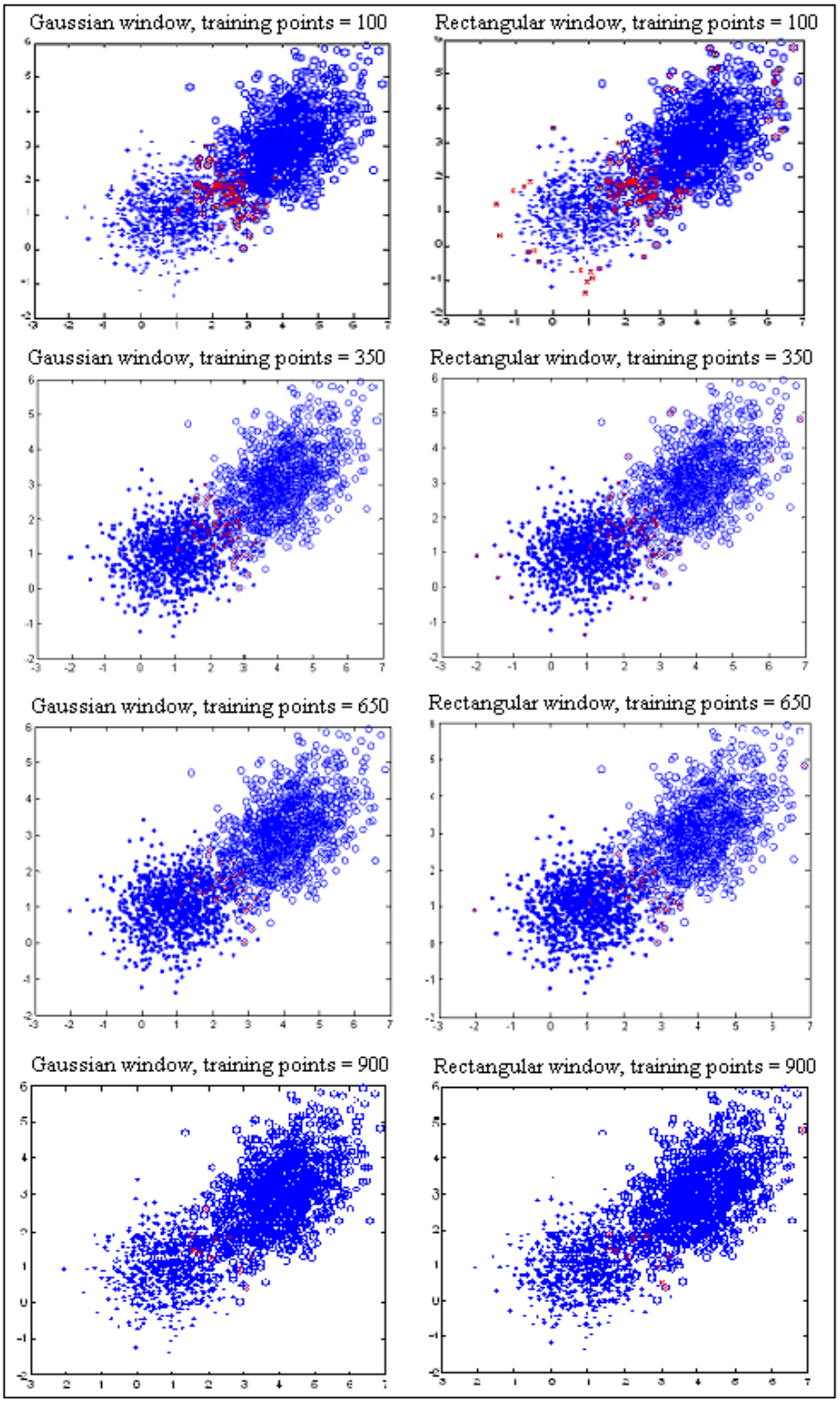

Fig20: Training data and misclassified points for different proportions of training data using Parzen windows

Case 2

We now consider the case of 2-D data that overlap.

	Class I	Class II
Mean	[1 1]	[2 2]
Variance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 0.7 & 0.21 \\ 0.21 & 0.56 \end{pmatrix}$

%training / H	10		35		65		90	
	G	R	G	R	G	R	G	R
2	78.56	78.83	79.0	79.07	79.29	78.42	75.0	74.0
1.25	80.17	78.94	78.85	78.31	78.29	78.14	81.5	80.0
0.75	79.72	77.95	78.54	77.54	77.86	77.29	81.0	81.0
0.50	79.76	74.44	78.62	76.77	78.0	76.58	81.0	79.0

We find that the accuracy rates are not very high for all the window sizes, with the highest being just over 80% for Gaussian window. This is owing to the overlapping of the data sets. Also we observe that some of the extremal points have been misclassified. This is because, even in such cases, they may have been those points which would not have had neighbors for that particular window size.

Further, there are also points in the border that were not misclassified, even though they are farther away from their classes. The reason for this is these points are training points, and NOT test points, and hence *don't appear to be misclassified*.

Following is the results for accuracy for rectangular and Gaussian windows of varying sizes for a value of h=1.25.

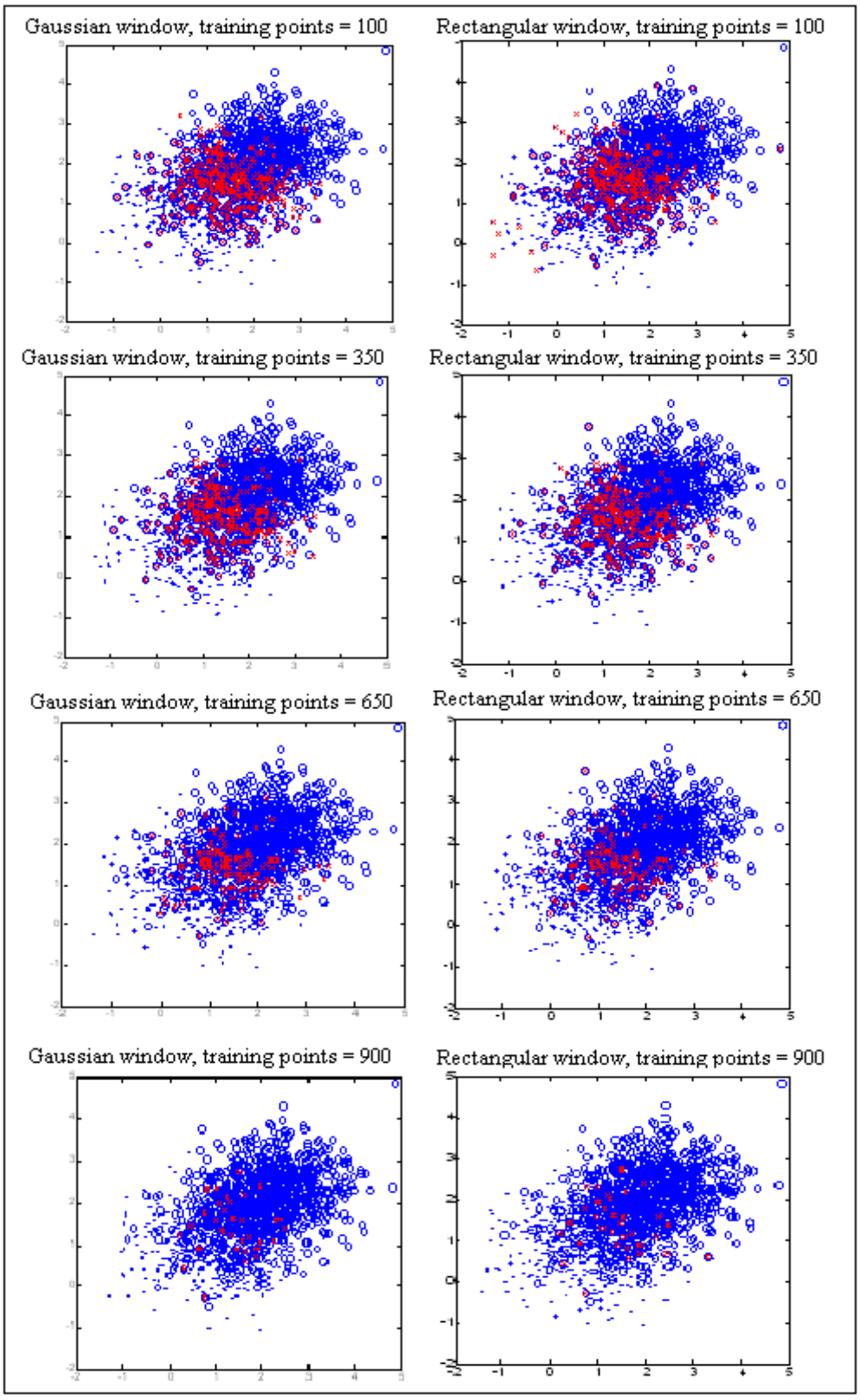

Fig21: Training data and misclassified points for different proportions of training data using Parzen windows

Case 3

Let us consider now 3-D data sets that are overlapping. The motivation behind going in for overlapping data sets is the observation that all the pattern classifiers perform exceptionally well under non-overlapping data, and this gives us very little scope for comparison of performance.

	Class I	Class II
Mean	[3 3 3]	[1 1 1]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

%Training H	10		35		65		90	
	R	G	R	G	R	G	R	G
0.5	58.61	91.72	74	92.6	80.71	93.42	82	92.5
1	82	91.33	89.76	92.30	91.14	92.42	92.5	91
1.5	88.94	90.11	92.93	91.46	93.42	92	92.5	89.5
2	91.5	89.33	92.92	90.38	93.57	89.85	92.5	88.5

Lets us also consider a few other cases, for academic interests, to see the performance of Parzen windows for other 3-d data sets.

	Class I	Class II
Mean	[3 3 3]	[1 1 1]
Variance	$\begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 2.5 & 0 \\ 0 & 0 & 5.5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

%Training H	10		35		65		90	
	R	G	R	G	R	G	R	G
0.15	50	82.83	50.4	84.53	50.57	85.14	50	87.00
0.5	53.6	86.61	60.15	88.38	63	88.4	63.5	90
1	67.38	88.11	80.23	88.07	83	88.14	86	88
1.5	78.33	87.77	85.15	87.53	85.71	87.57	88	87.5
2	83.72	87.44	87.23	87.07	87.28	87.14	88	86.5

We observe a decrease in the accuracy percentages with an increase in the amount of overlapping, caused by an increase in the variance of data. This is perfectly in order with the performance of the classifiers seen in the previous sections.

We shall plot the data sets and misclassified points, for a value of h=0.5. We here note that a large number of correct samples are misclassified as there are no neighbors in the window leading to the flip of a coin method as we have assumed equal priors.

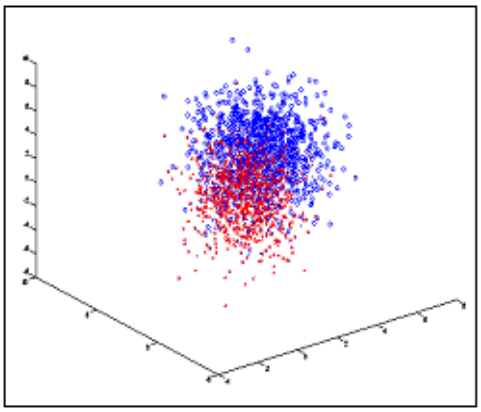

Fig21: Training data and misclassified points

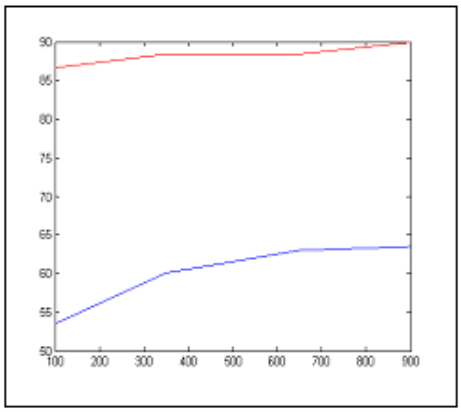

Fig22: Error plot for Parzen window, h=0.50

Now consider other 3-D data sets with more overlapping by shifting the mean

	Class I	Class II
Mean	[2 2 2]	[1 1 1]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

%Training	10		35		65		90	
H	R	G	R	G	R	G	R	G
0.25	50.56	60.67	50.92	67.58	50.28	68.12	53	73
0.5	50.88	67.55	53.76	72.23	55.71	72.14	56.5	73
1	57.94	71.67	63.23	73.53	66.85	73.71	70	74.5
2	69.27	71.72	72.46	75.38	76.71	73.85	78.5	74.5

As expected, and as is seen from the previous cases, the same argument hold for this case too, in that the accuracy shows a sharp decrease due to the presence of large overlapping in the data sets.

We shall now plot the data sets and the misclassified points for a value of h=1, by varying the training data's proportion, and the error plot.

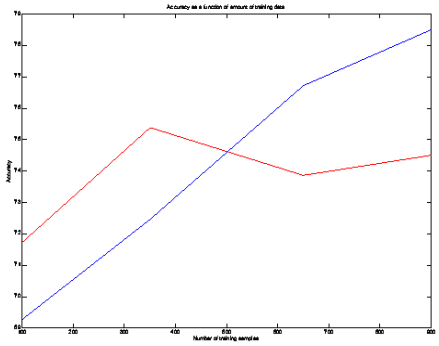

Fig23: Error plot for Parzen window, h=0.50

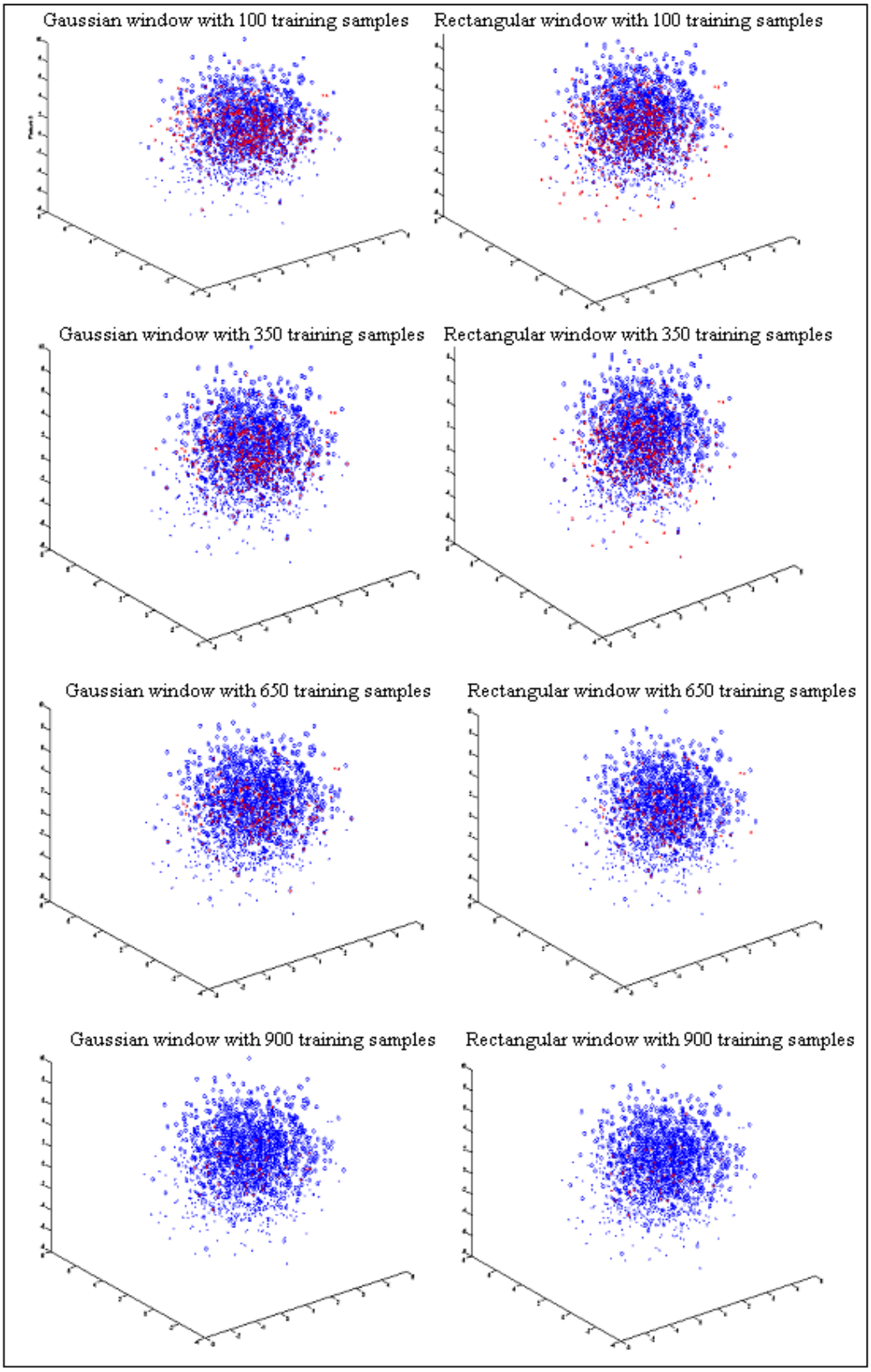

Fig24: Training data and misclassified points

We shall now look at the experimental results for the k-nearest neighbor technique. We have operated on exactly the same data as that used in Parzen Window technique

Case 4

As usual, we first consider 2-D well separated data, and the results are tabulated for the nearest neighbor (K=1) and other values of K.

	Class I	Class II
Mean	[1 1]	[4 3]
Variance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$

%training / K	10	35	65	90
1	91.00	91.07	88.57	87.0
3	90.89	90.00	88.28	89.0
5	91.11	91.07	88.57	89.0
7	91.22	91.38	89.71	88.0

Experiments were also conducted by changing the distance metric from ‘Euclidian’ to ‘city block’ distance. However, we do not observe any significant improvement in doing so.

%training / K	10	35	65	90
1	88.67	86.61	85.43	88.0
3	89.22	88.15	86.28	90.0
5	89.88	90.15	88.0	87.0
7	90.56	89.86	89.14	86.0

The following are the plots of KNN with Euclidian distance metric K=1

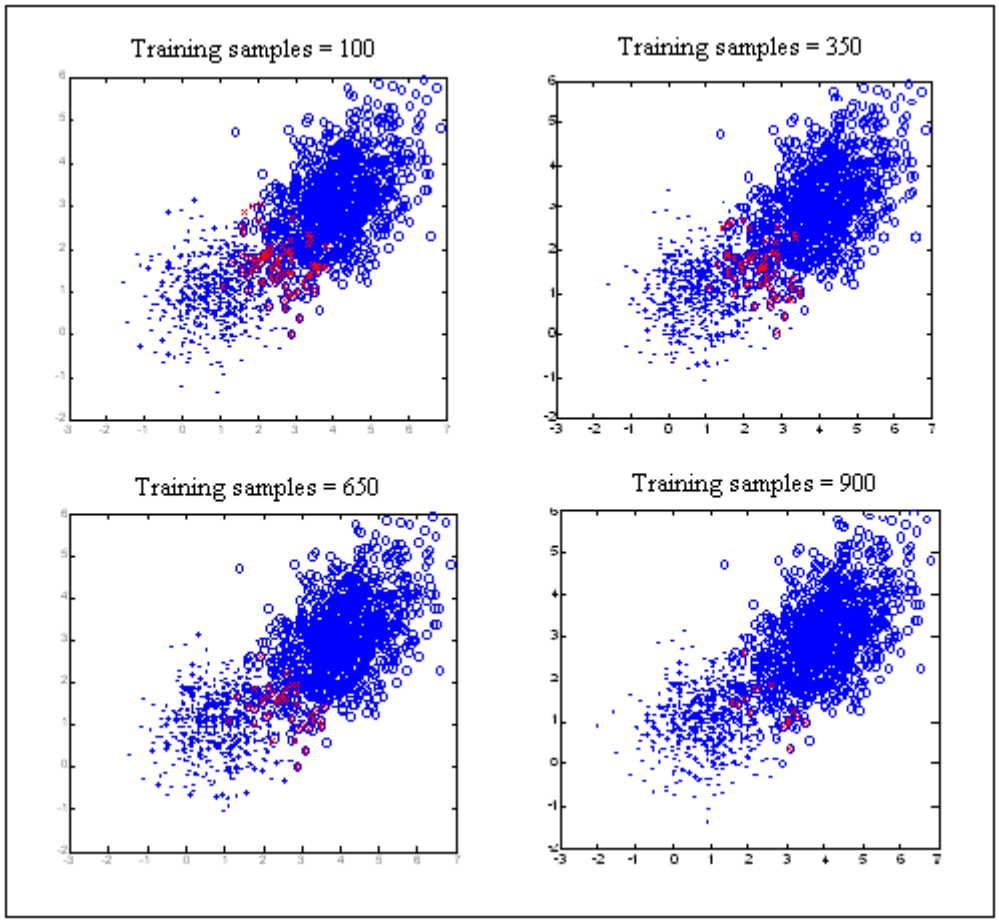

Fig25: Training data and misclassified points

Case 5

Consider next 2-D data sets that are overlapping, as was experimented with in the previous case. The data is given as:

	Class I	Class II
Mean	[1 1]	[2 2]
Variance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 0.7 & 0.21 \\ 0.21 & 0.56 \end{pmatrix}$

%training / K	10	35	65	90
1	43.89	38.15	42.57	47.0
3	53.11	45.23	50.28	60.0
5	54.33	50.46	52.57	59.0
7	59.67	56.0	56.28	61.0

As expected there is a decline in the accuracy rates, owing to an increase in the overlapping of data. We shall now plot the data sets and the misclassified points for a value of k=7. Figure 26 represents the plots for the above mentioned cases.

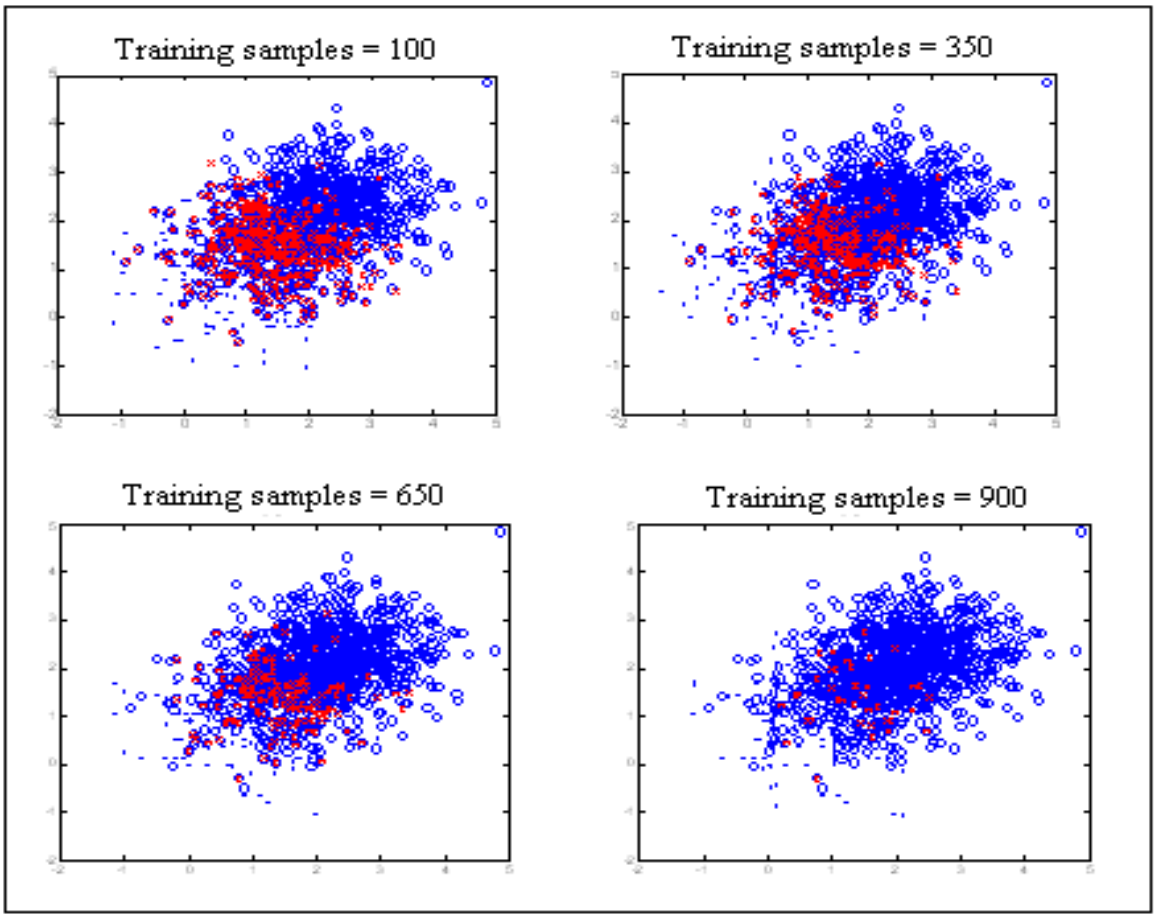

Fig26: Training data and misclassified points

Case 6

Consider next 3-D data sets that overlap to an extent. The data and the accuracy tables are given below. We find that the performance of this technique is pretty good, with accuracy percentages ranging from 82-87%. Plots are given for 2 cases of training data variation.

	Class I	Class II
Mean	[3 3 3]	[1 1 1]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

%Training / K	10	35	65	90
1	81	81.53	82.85	87
3	84.67	85.23	86	86
5	85.11	86.15	86	86
7	85.33	85.69	86.28	87
9	85.44	85.69	86.85	86
11	84.89	86.16	87.14	85
13	85	86.46	87.42	86

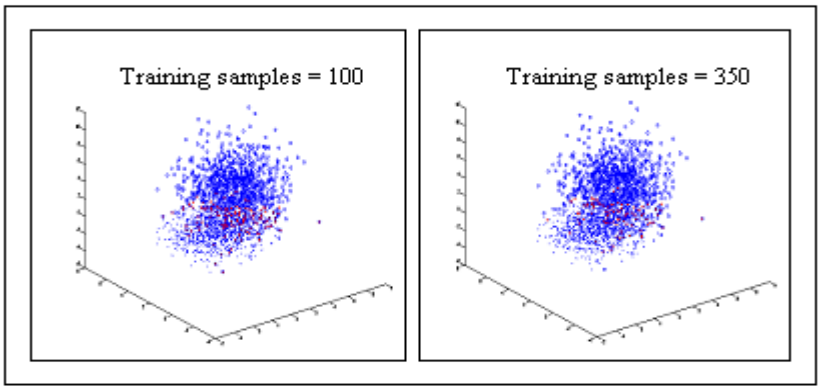

Fig27: Training data and misclassified points

Case 7

We shall see here the plots for 3-D data set II and set III. The results for set III are very poor, as we will see. We shall also have a corresponding look at the results of Parzen window, so that we can get to know the better of the two techniques.

	Class I	Class II
Mean	[3 3 3]	[1 1 1]
Variance	$\begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 2.5 & 0 \\ 0 & 0 & 5.5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

%Training / K	10	35	65	90
1	65.67	68.61	67.42	72
3	73.67	73.38	75.71	76
5	74.55	76	75.42	81
7	73.89	75.38	76.86	80
9	74.33	77.23	77.14	80
11	74.55	77.69	75.71	82
13	75.44	77.69	76.28	85

	Class I	Class II
Mean	[2 2 2]	[1 1 1]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}$

%T raining / K	10	35	65	90
1	29.44	31.23	30.28	33
3	39.77	37.23	38.85	38
5	40.77	39.23	35.71	39
7	42.77	38.15	43.42	47
9	43.11	42.76	41.42	47

The results, as can be seen are very poor, as there is a heavy amount of overlapping in the data. Let us have a look at Figures 23 and 24, and look at the corresponding table, which reveals that the Parzen window performs much better for the same data set. This is because for extremely overlapping datasets, there is a very high chance that the nearest neighbor could be a data point from the wrong class, and so the k-nearest neighbors technique succumbs to this grey point.

Hence, these are the conclusions that may be drawn from this experiment .

- Parzen window using Gaussian kernel is better than the one with rectangular kernel, which exhibits a heavy dependency on the size of the window h.
- K-NN performs better than nearest neighbor technique, and is more robust.
- K-NN is better than Parzen window operating with rectangular kernel as it doesn't misclassify extremal points. This is because, the extremal points, owing to their closeness to the correct class, can never get misclassified.
- Also, from our observation, we find that the Parzen window perform better than k-NN. This may be quite contrary to the existing theories, as the data that we have experimented with is one with lots of overlapping. This is because, the motivation was to experiment and study the performances of the classifiers for these types of data, as we were able to discern little about the performance of these classifiers on perfectly separable data.

The following segment shows the MATLAB routine for this experiment.

MATLAB CODE:

```
%Parzen window technique
function [e1 e2]=simulate_parzen(X1,X2,h)

%variables to hold size of input data
[m n]=size(X1);
[F d]=size(X2);
z=1;

%Gives different training percentages
H=[100 350 650 900];

%2-D case
if(d==2)
    for I=1:4
        plot(X1(:,1),X1(:,2),'.');hold on;plot(X2(:,1),X2(:,2),'o');

e1(z)=parzen(X1(1:H(I),:),X2(1:H(I),:),[X1(H(I)+1:m,:);X2(H(I)+1:m,:)],[ones(
m-H(I),1);2*ones(m-H(I),1)],h,1);
        figure
        plot(X1(:,1),X1(:,2),'.');hold on;plot(X2(:,1),X2(:,2),'o');

e2(z)=parzen(X1(1:H(I),:),X2(1:H(I),:),[X1(H(I)+1:m,:);X2(H(I)+1:m,:)],[ones(
m-H(I),1);2*ones(m-H(I),1)],h,2);
        z=z+1;
    end
end

%3-D case
if(d==3)
    for I=1:4
        plot3(X1(:,1),X1(:,2),X1(:,3),'.');hold
on;plot3(X2(:,1),X2(:,2),X2(:,3),'o');

e1(z)=parzen(X1(1:H(I),:),X2(1:H(I),:),[X1(H(I)+1:m,:);X2(H(I)+1:m,:)],[ones(
m-H(I),1);2*ones(m-H(I),1)],h,1);

        figure
        plot3(X1(:,1),X1(:,2),X1(:,3),'.');hold
on;plot3(X2(:,1),X2(:,2),X2(:,3),'o');

e2(z)=parzen(X1(1:H(I),:),X2(1:H(I),:),[X1(H(I)+1:m,:);X2(H(I)+1:m,:)],[ones(
m-H(I),1);2*ones(m-H(I),1)],h,2);
        z=z+1;
    end
end

plot(H,e1(1:z-1));hold on;plot(H,e2(1:z-1),'r')

%Compute accuracy
accuracy_rectangular=100-e1
accuracy_gaussian=100-e2

%function that performs Parzen window technique
```

```
function[error]=parzen(X1,X2,Xtest,group,h,ch)

%variables to hold size of input data
[m d]=size(X1);
[p d]=size(X2);
[Q R]=size(Xtest);

%counts the number of misclassified points
misc=0;

%Rectangular Window
if(ch==1)
    for k=1:Q
        px0w1=0;
        px0w2=0;
        for i=1:m
            count=0;
            for j=1:d
                %applying condition
                if(abs((X1(i,j)-Xtest(k,j))/h) <0.5)
                    count=count+1;
                end
            end
            if(count==d)
                px0w1=px0w1+1;
            end
        end
        for i=1:p
            count=0;
            for j=1:d
                %applying condition
                if(abs((X2(i,j)-Xtest(k,j))/h) <0.5)
                    count=count+1;
                end
            end
            if(count==d)
                px0w2=px0w2+1;
            end
        end

        %Making a decision; use a toss of a coin to resolve conflicts/ties
        if(px0w1>px0w2)
            class=1;
        elseif(px0w1<px0w2)
            class=2;
        else
            chance=randperm(2);
            if(chance(1) == 1)
                class =1;
            else
                class =2;
            end
        end

        %print misclassified points in red
        if(class~=0 && group(k)~=class)
```

```
        if(d==2),    plot(Xtest(k,1),Xtest(k,2),'rX');hold on
        end
        if(d==3),    plot3(Xtest(k,1),Xtest(k,2),Xtest(k,3),'rX');hold on
        end
            misc=misc+1;
        end


    end
end

%Gaussian Window
if(ch==2)
    for k=1:Q
      px0w1=0;
      px0w2=0;
        for i=1:m
            px0w1=px0w1+exp(-(0.5)*((X1(i,:)-Xtest(k,:))*(X1(i,:)-
Xtest(k,:))')/(h^2));
        end
        for i=1:p
            px0w2=px0w2+exp(-(0.5)*((X2(i,:)-Xtest(k,:))*(X2(i,:)-
Xtest(k,:))')/(h^2));
        end

        %Making a decision; use a toss of a coin to resolve conflicts/ties
        if(px0w1>px0w2)
            class=1;
        else
            class=2;
        end

        %print misclassified points in red
        if(group(k)~=class)
          if(d==2),  plot(Xtest(k,1),Xtest(k,2),'rX');hold on
          end
          if(d==3)
              plot3(Xtest(k,1),Xtest(k,2),Xtest(k,3),'rX');hold on;
          end
            misc=misc+1;
        end
    end
end

error=(misc/Q)*100;
hold off
```

```
%function that performs knn technique
function simulate_knn(X1,X2,k)

%variables to hold size of input data
[m n]=size(X1);
[F d]=size(X2);
z=1;

%Gives different training percentages
H=[100 350 650 900];

if(d==2)
    for I=1:4
        plot(X1(:,1),X1(:,2),'.');hold on;plot(X2(:,1),X2(:,2),'o');
        Xtrain=[X1(1:H(I),:);X2(1:H(I),:)];
        Xtest=[X1(H(I)+1:m,:);X2(H(I)+1:m,:)];
        group=[ones(H(I),1);2*ones(H(I),1)];
        expec=[ones(m-H(I),1);1+ones(m-H(I),1)];
        cl=knnclassify(Xtest,Xtrain,group,k);
        for t=1:length(cl)
            if(cl(t)-expec(t)~=0)
                plot(Xtest(t,1),Xtest(t,2),'rX');
            end
        end
    e(z)=(sum(abs(cl-expec))/(m-H(I)))*100;
    z=z+1;
    figure
    end
end
if(d==3)
    for I=1:4
        plot3(X1(:,1),X1(:,2),X1(:,3),'.');hold
on;plot(X2(:,1),X2(:,2),X2(:,3),'o');
        Xtrain=[X1(1:H(I),:);X2(1:H(I),:)];
        Xtest=[X1(H(I)+1:m,:);X2(H(I)+1:m,:)];
        group=[ones(H(I),1);2*ones(H(I),1)];
        expec=[ones(m-H(I),1);1+ones(m-H(I),1)];
        cl=knnclassify(Xtest,Xtrain,group,k,'cityblock');
        for t=1:length(cl)
            if(cl(t)-expec(t)~=0)
                plot3(Xtest(t,1),Xtest(t,2),Xtest(t,3),'rX');
            end
        end
        e(z)=(sum(abs(cl-expec))/(m-H(I)))*100;
        z=z+1;
        figure
    end
end

figure
plot(e(1:z-1))

%Accuracy
accuracy=100-e
```

References

[1] "*Pattern Classification*", Richard.O.Duda, Peter.E.Hart, David.G.Stork

[2] "MLP *and SVM Networks – a Comparative Study*", Stanislaw Osowski, Krzysztof Siwek and Tomasaz Mariewicz, NORSIG 2004