Question 1

INTRODUCTION : The Fisher cost function is given by $J(w) = \frac{w^T S_b w}{w^T S_w w}$. The value of w which optimizes this cost function is given by $w_0 = S_w^{-1}(m_1 - m_2)$.
In this experiment, we modify the Fisher cost function by replacing the within class scatter matrix S_w by the identity matrix, in the Fisher cost function. This implies that the value of w which optimizes the modified Fisher cost function is given by $w_m = (m_1 - m_2)$.

AIM : We aim to classify different data sets by these two classifiers – One using the Fisher cost function, and the other, using the modified Fisher cost function; and compare their performance.

METHOD : w_0 - solution obtained by optimizing the correct Fisher cost function.
w_m - solution obtained by optimizing the modified Fisher cost function.

We used synthetic data for all our experiments, since it gave great flexibility in evaluating the performance of our classifiers for different data sets. Basically, since we wanted to see, whether the classification obtained by w_m, can ever be better than that obtained by w_0, in some cases, we had to force the data to be such, that it gives a better performance using the modified Fisher cost function.
.
We wrote two programs in Matlab for each of the classifiers, which

1) Seperate the two data sets by finding the optimum w
2) Draw the projections of the data on the projection plane
3) Draw the separation hyperplane.
4) Evaluate the performance of classification.

Performance was measured by projection the data onto the plane containing w, and examining whether the separation hyperplane separates the data in the two classes satisfactorily.

The algorithm for drawing the plots are as follows:

1) Find w vector from the formulae.
2) Find the offset of the separation hyper surface from origin w0 = **-m**.**w**
   where **m** is the global mean of the data.
3) Find a perpendicular to **w**. For 2 D if w =[w(1) w(2)] we can choose a perpendicular like [-w(2) w(1)] and use this to draw a line through w0.
4) For the 3D case if w=[w(1) w(2) w(3)] find a perpendicular direction like wn= [-w(2) w(1) 0].

5) Project the data vectors on wn and subtract **data-projection on wn** to find the the projection on plane containing w.

6) To draw the hypersurface in 3-D compute

$$Z=(-w(1).x- w(2).y + w0)/w(3)$$

7) Use MATLAB mesh(z) command to draw the plane.

EXPERIMENTS:

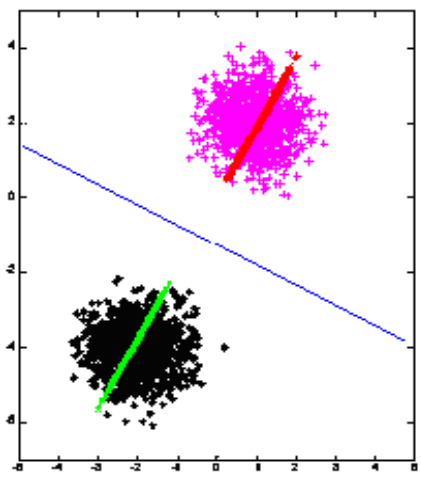
We performed experiments on data by varying the amount of separation, and by varying the dimension. In this section, we have listed the various experiments conducted, the plots of the data along with the separation hyperplane and the projection plane, for both the methods, and their respective performance measures.

2-D Data

1) Very well separated data

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

	Fisher	Modified Fisher
Error %	0	0



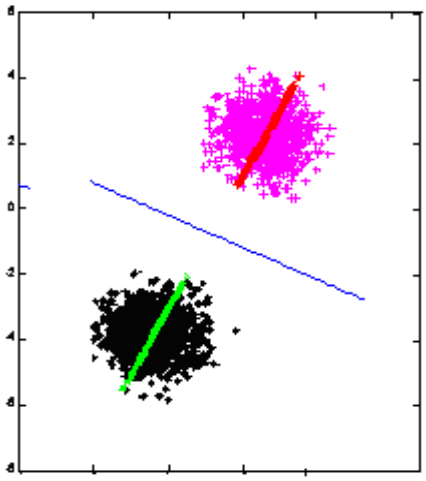


Figure (i) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

Since the data is very well separated, both the methods classify the data very well, and have a zero error.

2) Separated data which leads to error.

In this experiment, we purposely generated data, for which the Fisher method will perform better than the modified Fisher.

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

	Fisher	Modified Fisher
Error %	0	3.85

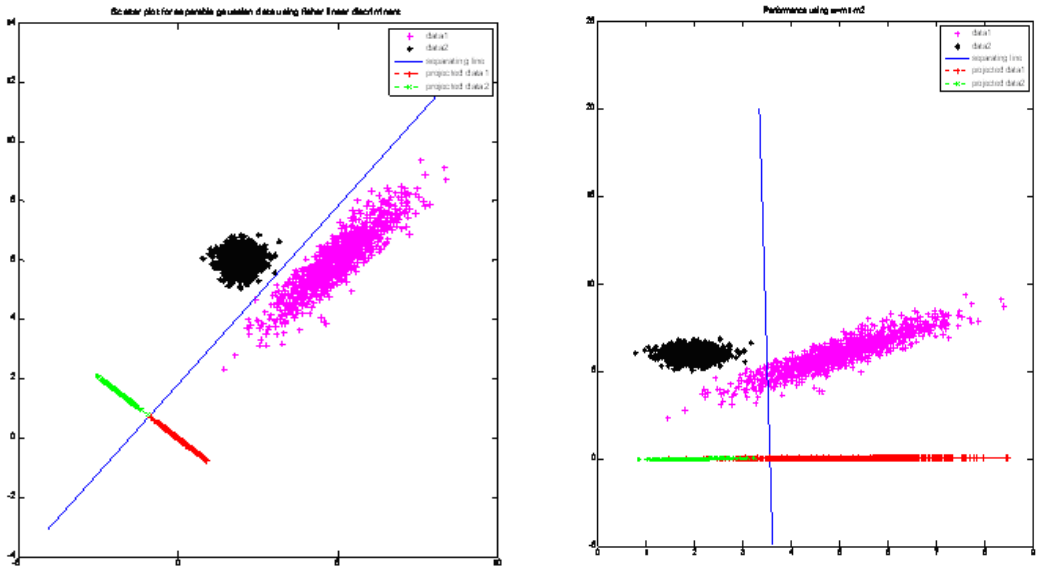


Figure (ii) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

Modified Fisher method performs worse than Fisher in this case, since the separation plane it draws is perpendicular to the difference of the means, i.e. it only tries to maximize the separation of the means of the projected data. On the other hand, the Fisher method not only tries to maximize the separation of the means of the projected data, but also minimizes the variance, and hence, in this case, it gives a better result.

3) **Overlapping data**

	Data 1	Data 2
Mean	([2 8]	$\begin{pmatrix} 0.08 & 0 & 0 \\ 0 & 0.08 & 0 \\ 0 & 0 & 0.08 \end{pmatrix}$
Variance	$\begin{pmatrix} 8.48 & -0.05 \\ -0.05 & 7.6 \end{pmatrix}$	$\begin{pmatrix} 8.64 & 0.15 \\ 0.15 & 8.53 \end{pmatrix}$

	Fisher	Modified Fisher
Error %	16.75%	16.85 %

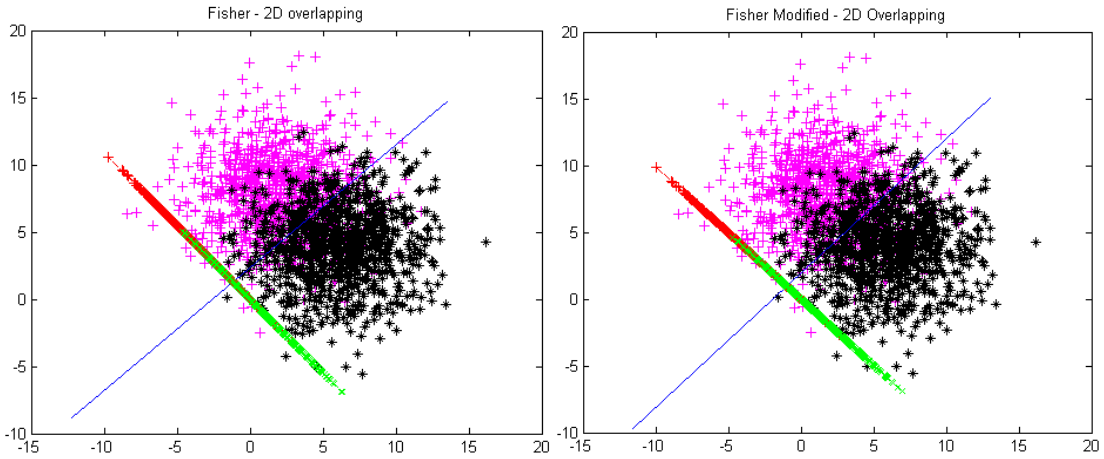


Figure (iii) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

We observe, that the Fisher method again gives a lower error. But the modified Fisher method also gives a comparable result in this case because, for this choice of data, the w vector obtained by the Fisher method is very close to w1-w2.

4) **Non overlapping data (which gives better performance with modified Fisher)**

The results of all the experiments so far lead us into believing that Fisher method is better than Modified Fisher. The motivation of this experiment was to investigate, whether this is always true. Keeping in mind that the Fisher method tries to maximize the seperation of the means and minimize the variance of the projected data, while the modified Fisher method only maximized the separation of the means, we generated a data set, in which, trying to minimize the variance of the projected data, will lead to a wrong separation surface.

We generated data, which has a huge variance in the x direction, and very less variance in the y direction. Also, we placed the means on the x-axis. Following is a diagram, showing the shape of the data set.

Fig(iv) Showing the data and its projection (enlarged view for the MODIFIED fisher's method)

We can intuitively see, that modified Fisher will draw a separation line which is perpendicular to the x-axis. On the other hand, in order to reduce the variance, the Fisher method will draw a separation which is slightly tilted, and hence lead to misclassification.

	Data 1	Data 2
Mean	[25 0]	[-25 0]
Variance	$\begin{pmatrix} 1026.9 & 6.3 \\ 6.3 & 16.9 \end{pmatrix}$	$\begin{pmatrix} 1027.1 & 4.4 \\ 4.4 & 17.1 \end{pmatrix}$

	Fisher	Modified Fisher
Error %	16.025	0

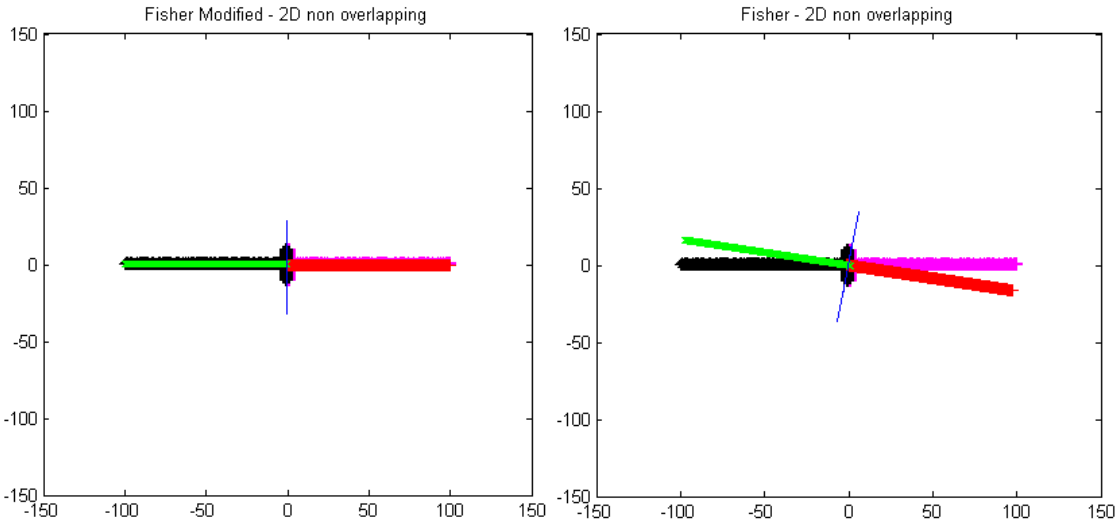


Figure (v) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

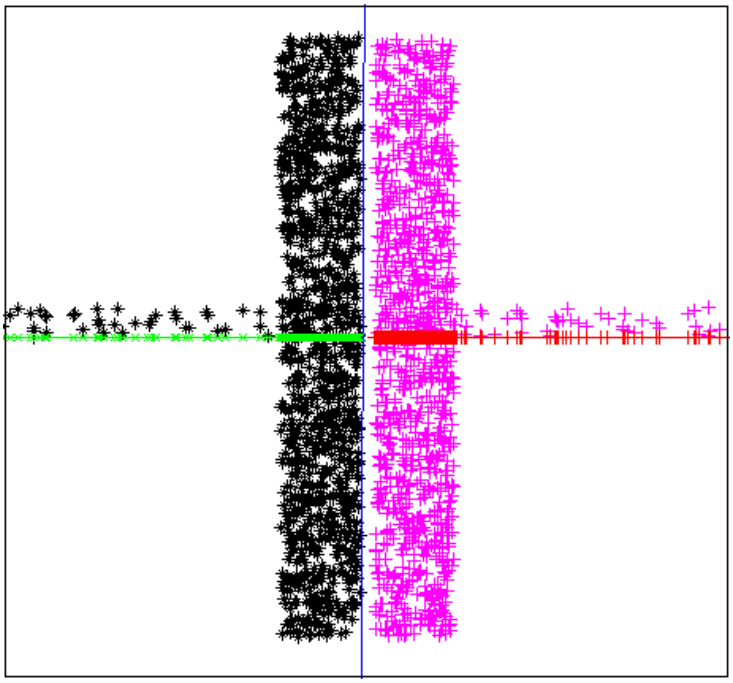


Figure (vi) showing the projections and separation hyper plane for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

We observe that, Fisher method does not always perform better than the modified Fisher method. But since the probability of coming across such data in practice is pretty small, and considering the success of the Fisher method in the rest of the cases, we can conclude that Fisher method of classification does much better than the Modified Fisher method.

3-D data

5) Well separated data in which both yield similar results

	Data 1	Data 2
Mean	[1.0285 1.9437 3.0294]	[-0.9586 -2.0054 -2.9299]
Variance	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

	Fisher	Modified Fisher
Error %	0	0

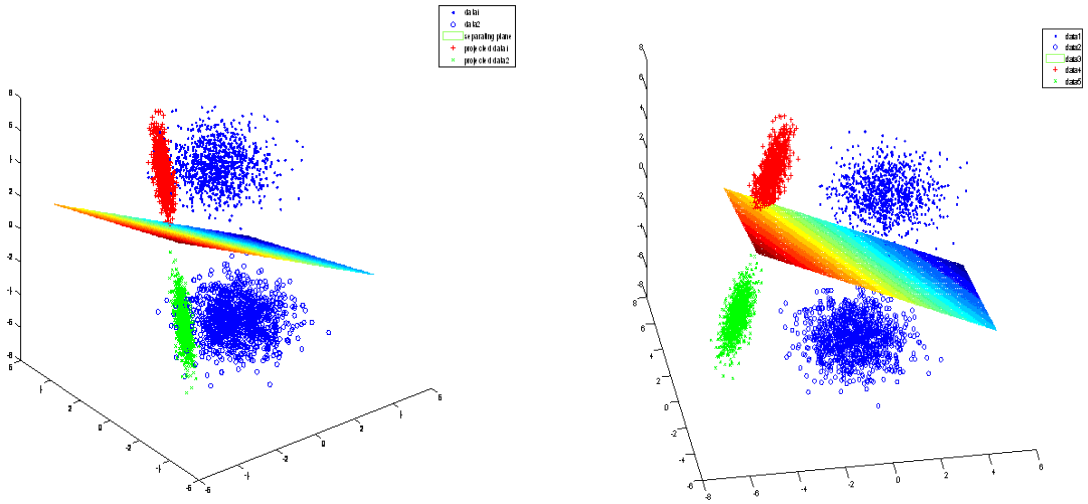


Figure (vii) showing 2 classes , the offset projection and the separating plane for both the methods.

Since the data is very well separated, both the methods classify the data very well, and have a zero error. This result is same as that obtained in 2-D, as expected.

6) Overlapping data

	Data 1	Data 2
Mean	[0.9912 2.0148 3.0280]	[-0.9889 0.9819 1.9961]
Variance	$\begin{pmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.4 \\ 0.1 & 0.4 & 0.9 \end{pmatrix}$	$\begin{pmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.4 \\ 0.1 & 0.4 & 0.9 \end{pmatrix}$

	Fisher	Modified Fisher
Error %	10.65	11.85

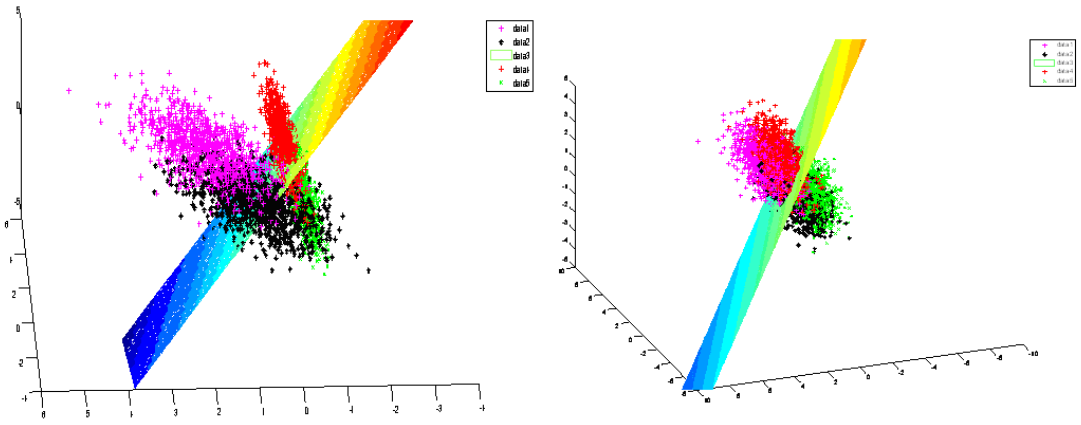


Figure (viii) showing 2 classes in pink and black , the offset projections in red and green and the separating plane

We observe, that the Fisher method performs only marginally better than the modified Fisher method. This result is also the same as that obtained in 2-D.

7) 3D Nonoverlapping in which Fisher does better

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

	Fisher	Modified Fisher
Error %	0	11

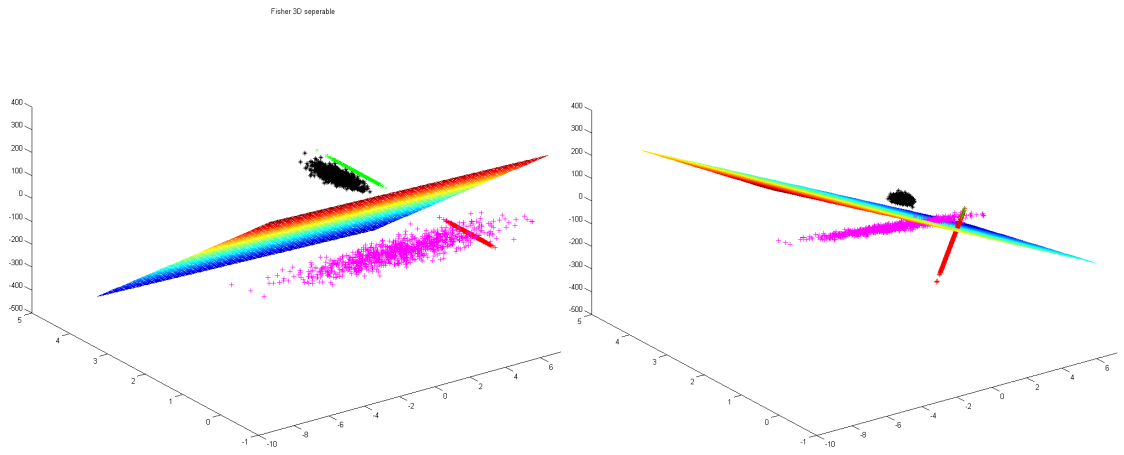


Figure (ix) showing 2 classes in pink and black , the offset projections in red and green and the separating plane using the normal fisher method

Modified Fisher method performs worse than Fisher in this case, since the separation plane it draws is perpendicular to the difference of the means, i. e. it only tries to maximize the separation of the means of the projected data. On the other hand, the Fisher method not only tries to maximize the separation of the means of the projected data, but also minimized the variance, and hence, in this case, it gives a correct separation plane.

CONCLUSION

We saw that in most of the cases for which we experimented, the Fisher method performed better than the modified Fisher method.

But we also looked at a case (Experiment 4), where the modified Fisher's method outperforms Fisher's method, and hence, we can't say that Fisher is always better than modified Fisher. But considering the fact that it is very unlikely to come across such data in practice very frequently, we can safely say that the Fisher method can be the better choice among the two.

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)';

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 the plane in 3-D
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 and plot for 2-D
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 2

AIM: In this experiment, our aim is to perform a comparative analysis of Neural Networks and Support Vector Machines in terms of classification performance.

METHOD : We have made use of in built tool boxes in Matlab® to perform our experiments. A brief description of the toolboxes can be found at the end of this section.

EXPERIMENTS:

In all our experiments, we use 1000 points for each of the classes and vary the amount of training and test data in the proportion 10-90,35-65,65-35 and 90-10 respectively.

For SVM, we have used two different kernels – the radial basis kernel, and polynomial kernel, and looked at their relative performances.

We have used the MLP neural network trained using Levenberg-Marwuardt algorithm. We have also experimented by changing the number of hidden neurons in some cases.

2-D data

2-D overlapping data

	Data 1	Data 2
Mean	[1 1]	[4 3]
Variance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$

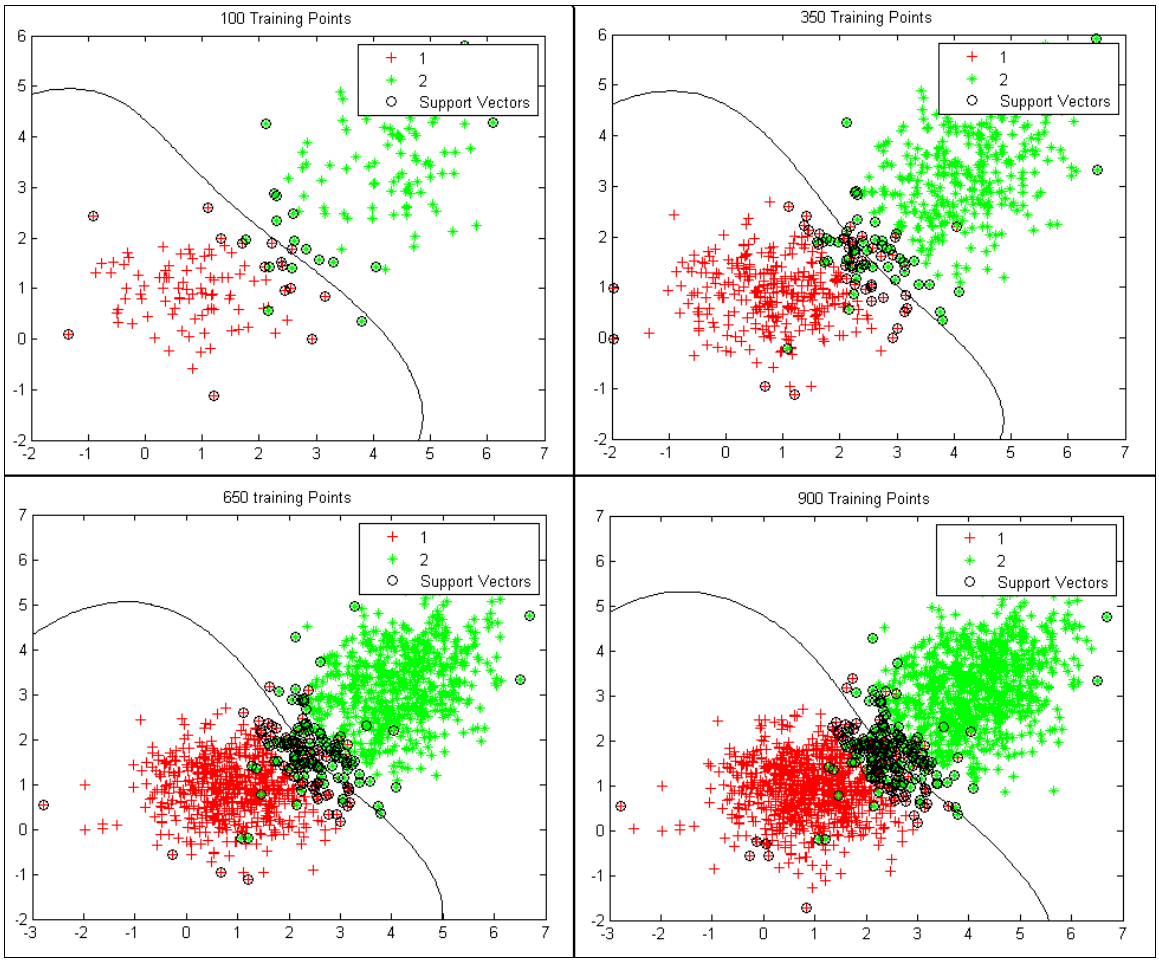


Figure (i) Training data and support vectors for different proportions of training data using ***radial basis function***

# of Training Samples	100	350	650	900
Accuracy Neural Network (10 hidden neurons)	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

In this case, we have chosen data which has a small amount of overlap. We observe that SVM using RBF kernel and polynomial kernel have a similar performance in this case.

With an increase in the amount of training data, we observe that the performance of SVM stays almost the same.

As opposed to this, with neural networks, we observe that with increase in the amount of training data from 10% to 65%, the accuracy of classification improves; but it drops suddenly in the end, when we use 90% of the data as training data. This trend is found in all our experiments, and this proves that the performance of neural networks drops due to over fitting.

If we look at the relative performance of SVM and neural networks, we can see that there is an insignificant difference between their performances in this case. But the time taken in case of neural networks was more than that of SVM.
Also, if we increase the number of hidden neurons in the neural network, their performance improves, but it comes at an expense of increase in the operating time.

2D more overlapping data with closer means

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

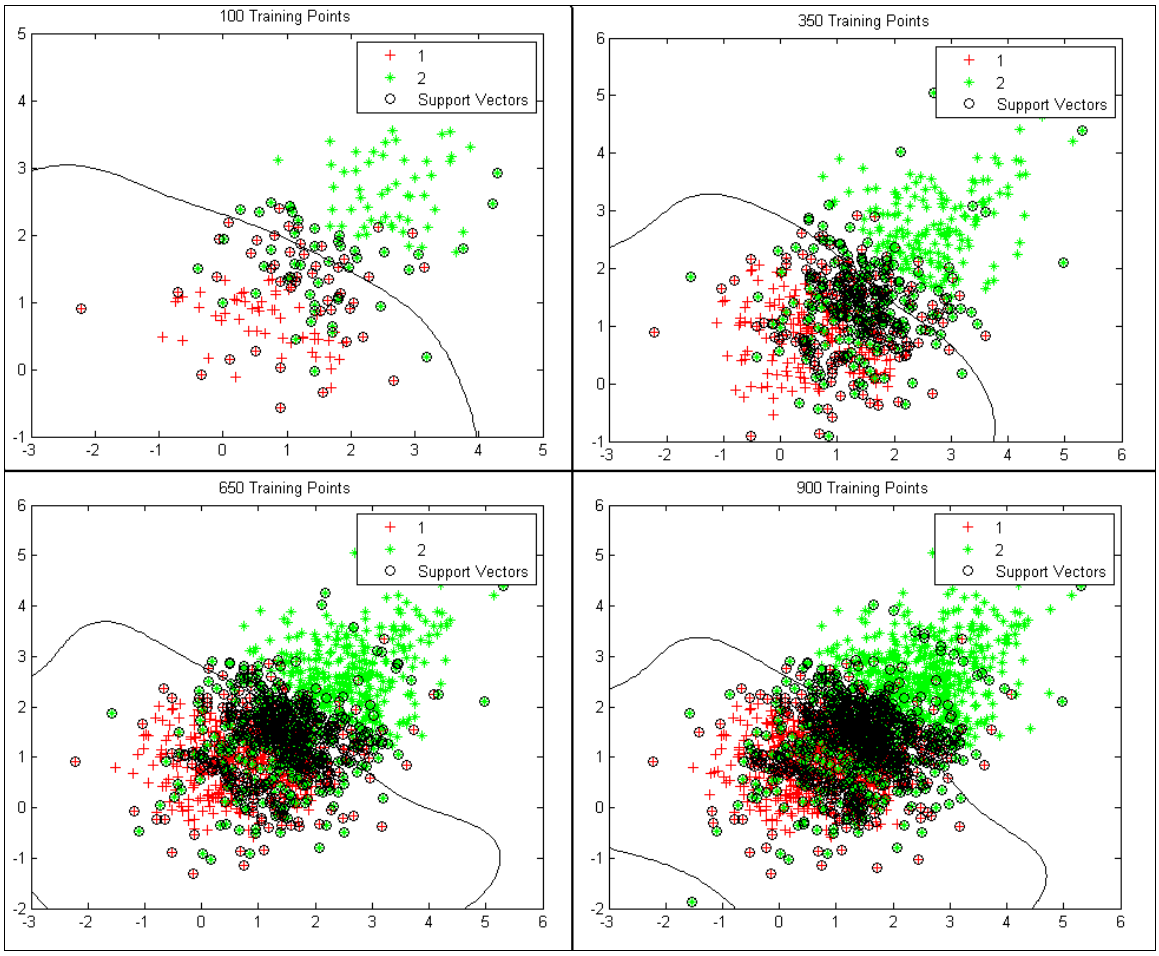


Figure (ii) Training data and support vectors for different proportions of training data using ***radial basis function***

# 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 2nd degree poly. kernel	74.8333	74.6923	73.7143	77
Accuracy 2nd degree poly. kernel	74.0556	73.5385	NC	NC

In this experiment we wanted to see what happens when the amount of overlap in the data is increased. We observed that the performance of both the networks drops significantly in this case.
With lesser amount of training, we observe that neural networks perform better than SVM. But as we increase the amount of training, the performance of SVM improves and is pretty close to that of the neural network. Since SVM takes much lower time than NN, we can say that their relative performances in this case are almost equivalent.

3D data

Small amount of Overlap

	Data 1	Data 2
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)	88.3889	89.3077	89.2857	92.5000

SVM does marginally better than NN in this case.
Overtraining of NN is also demonstrated here.

3D more overlapping by just increasing the variance.

	Data 1	Data 2
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)	87.7778	88.8462	88.2857	87.5000

In this case, the data have the same means as that of the previous experiment, but the variance of one of the data sets is increased. And we can clearly see that there is a decrease in the accuracy of the SVM in this case. Thus, we conclude, that by increasing the amount of overlap, the performance of the SVM drops.

3-D more overlapping with shifted mean

	Data 1	Data 2
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)	72.2222	70.8462	72.1429	74.5000

In this case, we increased the amount of overlap by bringing the means closer together.

We observe that as the amount of overlap increases, the performance of both the classifiers reduces significantly. But in all the cases, the performance of SVM is only marginally better than the NN. But this is not enough to say that SVM is “better” than NNs. We can increase the number of hidden layers or the number of epochs, and in that case, the performance of the NN may improve, but at the expense of the increased time.

Thus, we can say that if trained appropriately, NNs can achieve a good accuracy, but SVMs can do as well, if not better, and in a much lesser amount of time.

SPIRAL DATA

The two spiral problem is one of the most demanding classification problems [1] and [2], and we wanted to see the relative performance of SVMs and ANNs for this problem. The data points of the two classes form two interlocking spirals, going about the origin. The equations of the two spirals are $r = \pm\sqrt{\theta}$.
The code used to generate this kind of data, and for experimentation is attached in the end of this section.

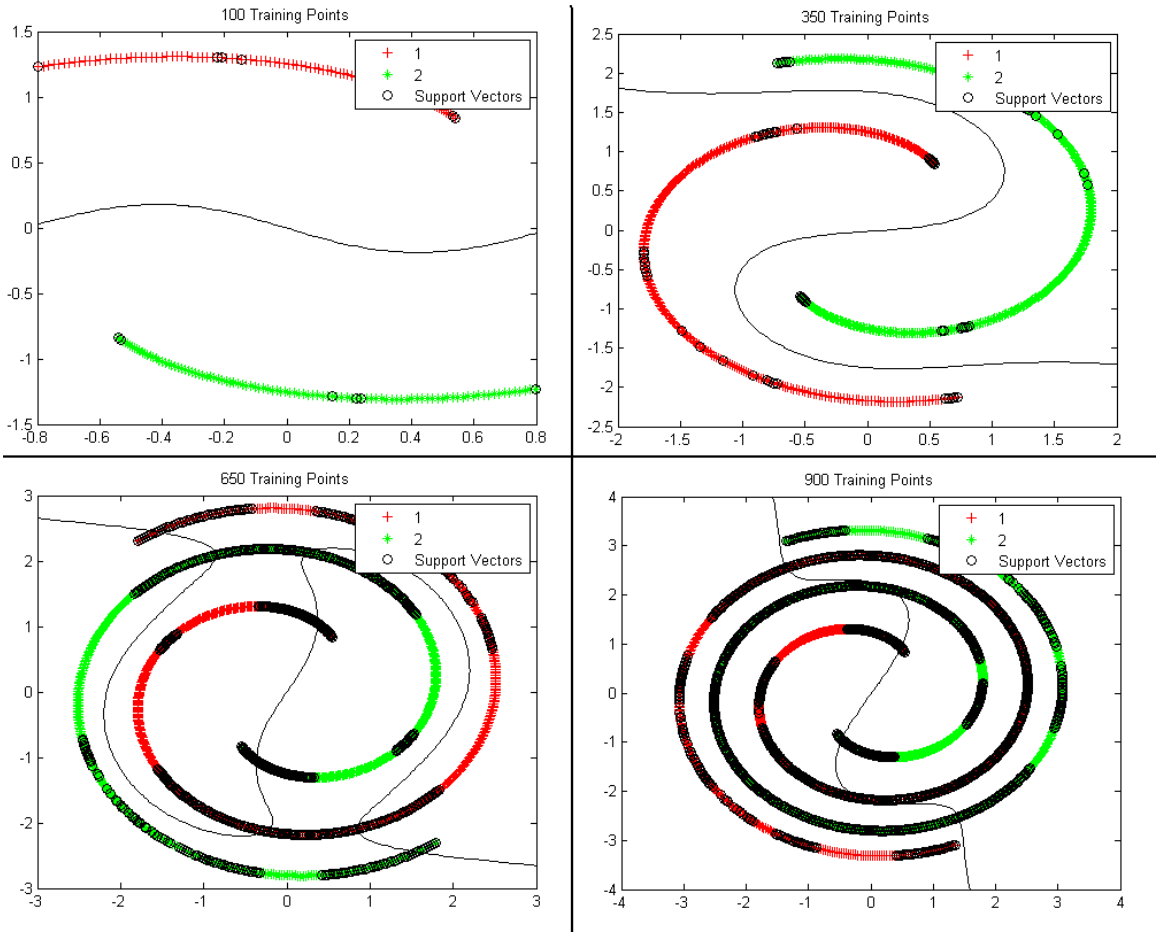


Figure (iii) ***Training data*** *and support vectors for different proportions of training data using* ***radial basis function***

# of Training Samples	100	350	650	900
Accuracy Neural Network	41.61	45.07	30.71	33.0
Accuracy RBF Kernel	47.8889	48.1538	20.8571	2.5000

In this case, we trained the SVM and NN, using the first 100, 350, 650 and 900 points, respectively. We observe that the results obtained are very poor for both NN and SVM. This can be attributed to the fact that, since we used the initial points for training, the separation surfaces were good enough only to separate the training data. And since the test data is not at all correlated with the training data, both the networks performed very poorly.

We can see this by observing the 4^{th} figure, in which the accuracy is only 2.5%. This is because, almost all the testing data which is to appear for the first class will fall in the region of the second class, and vice versa.

Hence, it is not a good idea to draw any conclusions about the classification performance of the networks, by using this experiment, since, essentially, we are asking it to predict the data, rather than classify.

And hence, in our next experiments, we sampled the data uniformly, and gave it for testing, in which case, the results will be much more meaningful, and will give the true performance in terms of classification.

Sampled Spiral

For this experiment, we chose the spiral to have a maximum angle of 4*pi. And we also varied the kernel function of the SVM, and the number of hidden neurons in the Neural Network.

Fraction of samples	1/10	1/20
Accuracy NN 25 hidden neurons	99.94	99.94
Accuracy Neural Network 10 hidden neurons	82.77	75.47
Accuracy with 8^{th} degree poly Kernel	99	94.2105
Accuracy with Rbf Kernel	55.77	56.63

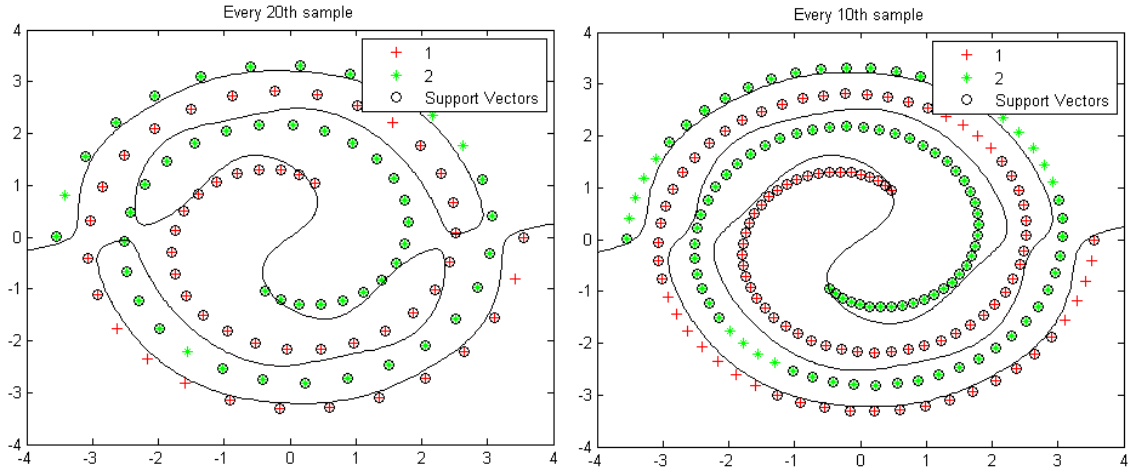


Figure (iv) Training data and support vectors for different proportions of training data using 8^{th} order polynomial function using every 20^{th} and 10^{th} sample.

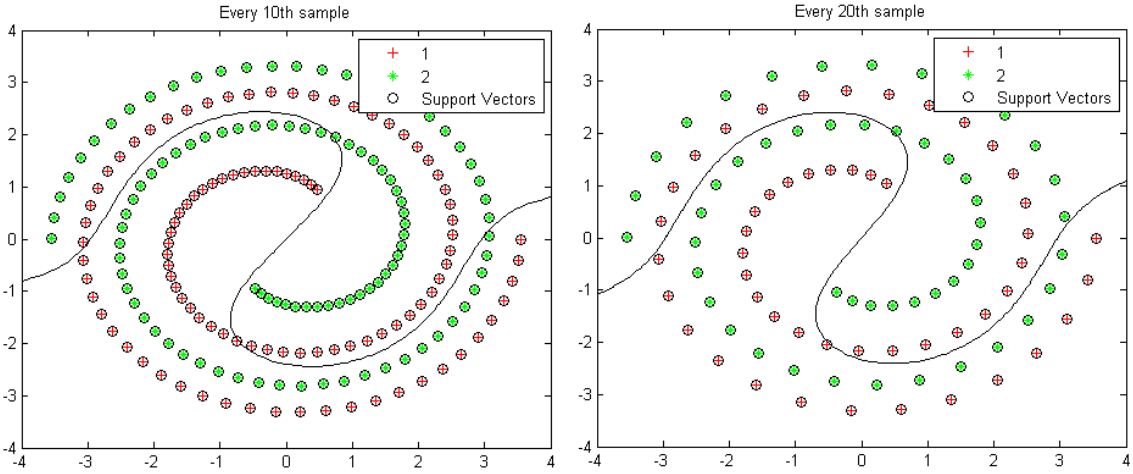


Figure (v) Training data and support vectors for different proportions of training data using radial basis function using every 10^{th} and 20^{th} sample.

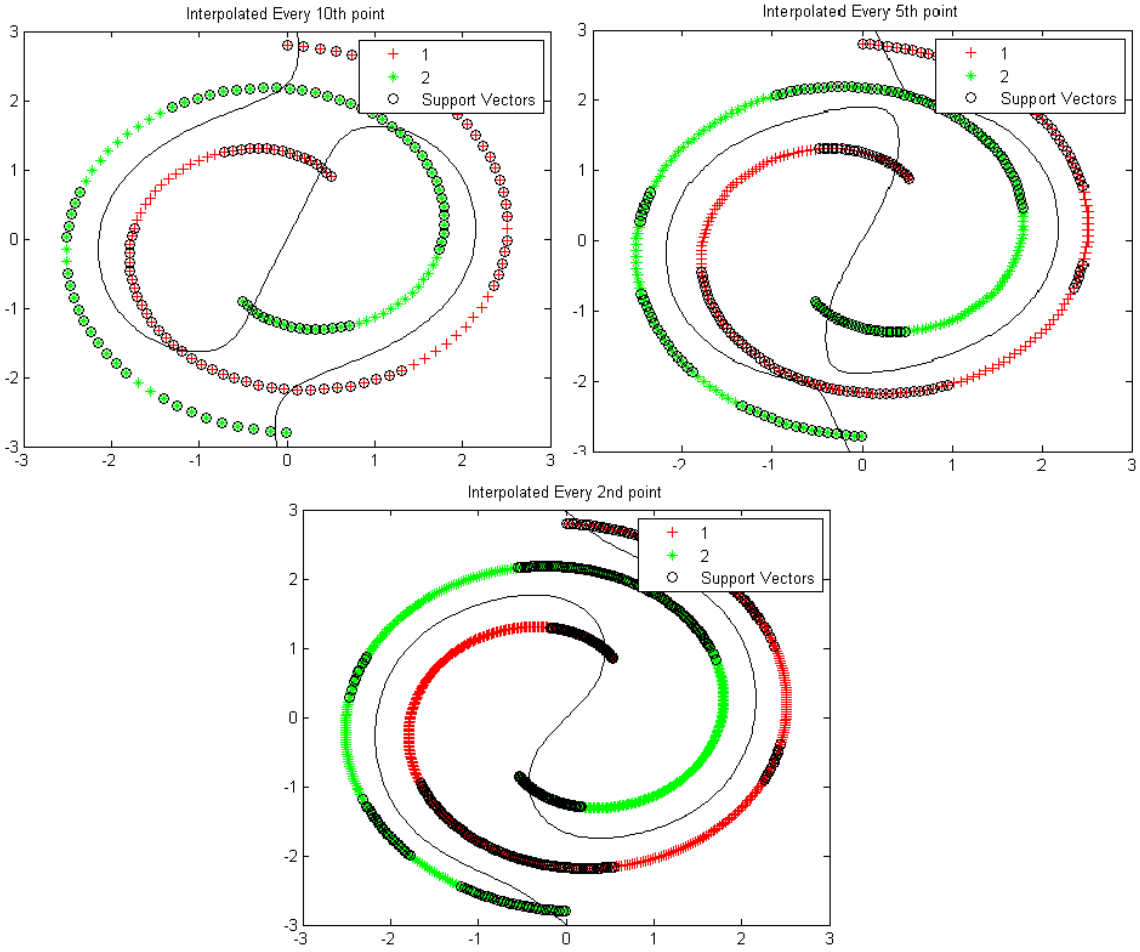
Comparison between kernels of SVM : We can clearly see that using a polynomial kernel of 8^{th} degree gives a much better performance than that obtained by using a RBF kernel. We also tried using polynomial kernels of lesser degree, and found that below 7^{th} degree, the kernels perform very poorly. For 7^{th} degree polynomial kernel, the accuracy was found to be 98.77%. Since the 7^{th} and 8^{th} degree kernels have a very high degree of freedom, the results are found to be really impressive, even in the case of spiral data. Also, we found that as we increase the # of rotations of the spiral, we will need a polynomial kernel with higher and higher degree.

Comparison between NN and SVM : We can see that SVM with an 8^{th} degree polynomial used as kernel gives an accuracy of 99% when trained with every 10^{th} data point. And this is very good, if we compare it with the performance of the NN with 10

hidden neurons. But if we increase the number of hidden neurons in the neural network, we can obtain an accuracy of almost 100%, even if we train it with half the number of points, which is impressive. But this comes at the expense of increased operation time.

Thus, we can not say that one method is better than the other, but the choice of method depends on the type of application. If the application demands low time complexity, then SVM is certainly a better option. But if it demands very high accuracy, and there is no time constraint, then NNs are better. In the rest of the cases, the decision has to be made, based on what we are willing to sacrifice, accuracy or time.

We also performed an experiment by reducing the angle of the spiral to 2.5*pi. The motivation of this experiment was to see whether the RBF kernel always performs badly. In order to test this, we reduced the complexity of the spiral classification problem, by reducing the angle of the spiral to 2.5*pi.



Figure(vi) ***Training data*** *and support vectors for different proportions of training data using* ***radial basis function..***

Fraction of Samples	1/10	1/5	1/2
ANN with 25 hidden neurons	99.94	99.94	99.94
Accuracy of ANN with 10 hidden neurons	66.89	76.23	81.33
Accuracy RBF Kernel	86.5000	95.2500	96.5000

We can see that the accuracy of the RBF network has improved significantly in this case, as compared to the previous case, and is actually better than the accuracy of the NN with 10 hidden neurons. But as we observed in the previous case, we can always optimize the number of neurons in the hidden layer, and achieve a very good performance with Neural Networks.

CONCLUSION

We performed classification by using both SVM and neural networks, for different types of data, with varying amounts of overlaps, and also with spiral data, which is considered to the most demanding classification problem. Based on our results, it is difficult to say that one is better than the other. But one thing, which we can surely say, is that, although Neural Networks give a highly accurate performance, they take a lot of time. Also, they face the problem of overfitting, and so, the amount of training has to be optimized to avoid this problem. On the other hand, SVMs take very less time for training, and the performance does not seem to vary a lot with the amount of training. Also, unlike ANNs, the computational complexity of SVMs does not depend on the dimensionality of the input space.

In essence, the choice of method depends on the type of application. If the application demands low time complexity, then SVM is certainly a better option. But if it demands very high accuracy, and there is no time constraint, then NNs are better. In the rest of the cases, the decision has to be made, based on what we are willing to sacrifice, accuracy or time.

SVM tool in Matlab®.

Commands used in the experiments:

1) svmtrain

svmtrain takes as input the test vectors, the group they belong to, the kernel function , the order of the polynomial in the case of a polynomial kernel and a option to show plot. The show-plot option plots the training data labeling the classes and indicates the support vectors chosen. It also draws the separation hypersurface. The function returns all these data as a structure variable. This can be used in classification.
There are several options for Kernel Function and we have experimented with the Radial Basis function and the polynomial kernel of varying orders.

2) svmclassify

This command takes the above returned structure variable and the matrix of test vectors and classifies them as either class 1 or class 2. It returns the classes of all the test vectors in the order in which they appear. We can use this returned information to check the accuracy of the classifier.

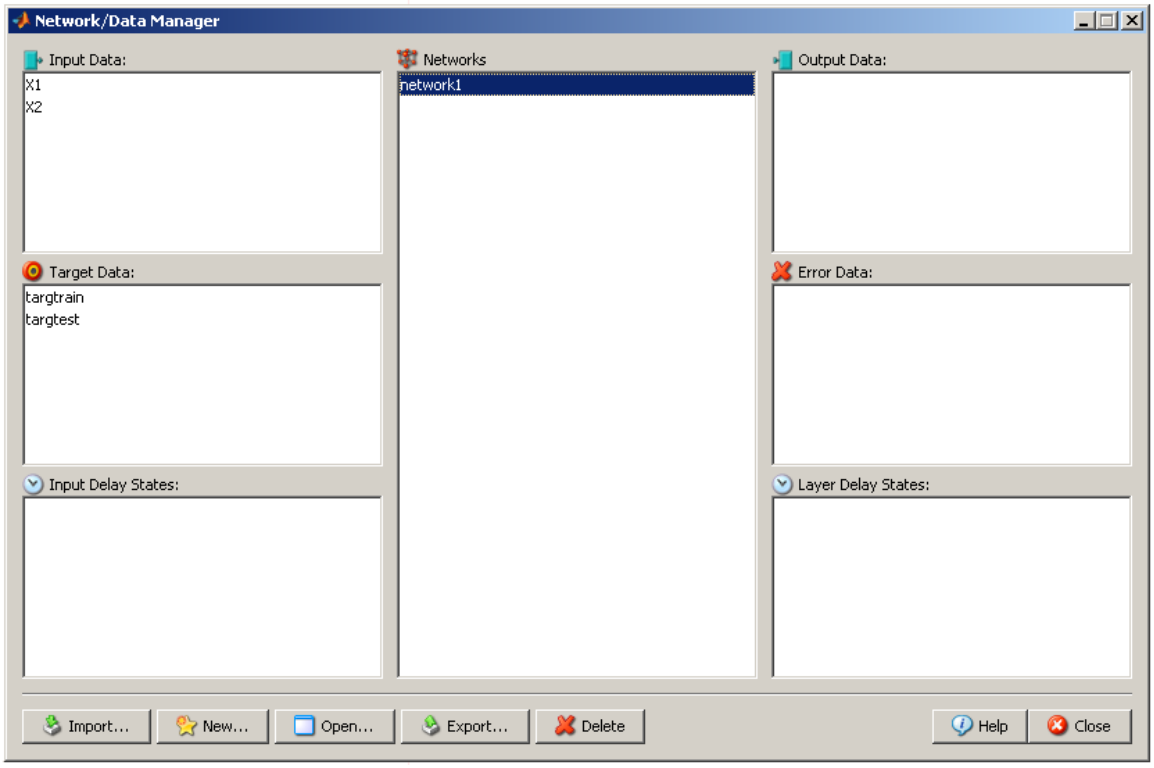
NNtool in Matlab®

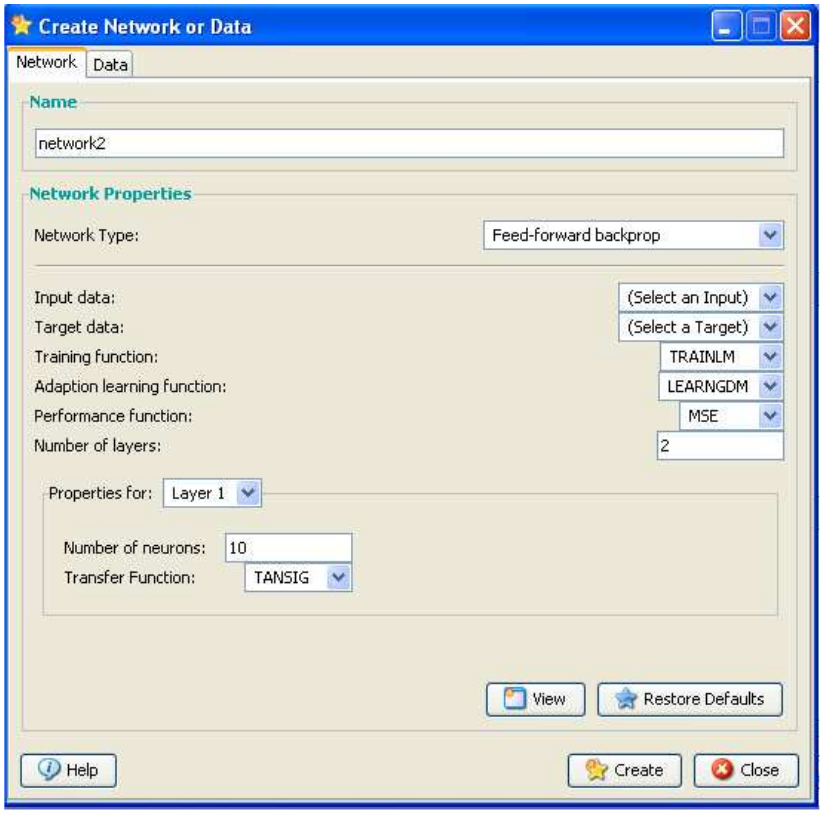
This tool is a Graphical User Interface Tool. The user can specify several parameters of the neural network such as the activation function, the initial weights, the training and test data, the number of epochs and the target data for the giving training set.

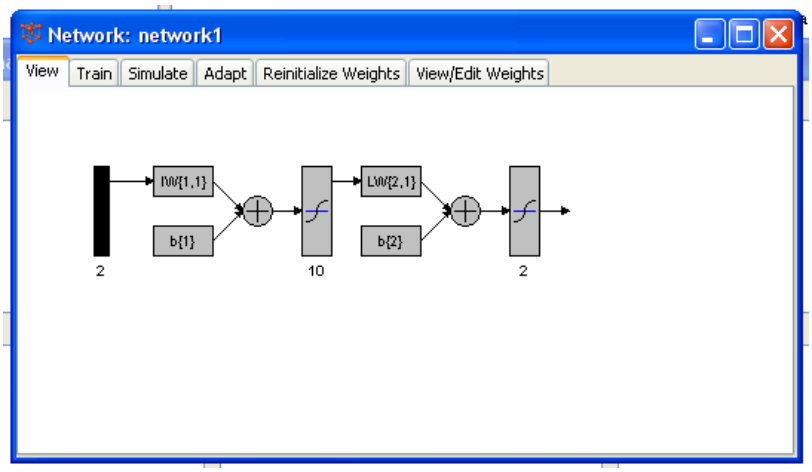
The tool expects the user to give the training data in the form of a d X n matrix where d is dimensionality of the data and n is the number of training vectors. It expects “target” data in the form of a 1 X n vector each entry specifying the class of the corresponding training sample. Once this data is given we can “create” the network by specifying the following:

a) Type of network
b) Training function
c) Adaption Learning Function
d) Number of layers
e) Number of neurons per layer
f) Transfer Function for neurons.

Following are some snapshots of the tool


Network/Data Manager
Input Data:
X1
X2
Networks
network1
Output Data:
Target Data:
targtrain
targtest
Error Data:
Input Delay States:
Layer Delay States:
Import...
New...
Open...
Export...
Delete
Help
Close


Create Network or Data
Network
Data
Name
network2
Network Properties
Network Type:
Feed-forward backprop
Input data:
(Select an Input)
Target data:
(Select a Target)
Training function:
TRAINLM
Adaption learning function:
LEARNGDM
Performance function:
MSE
Number of layers:
2
Properties for:
Layer 1
Number of neurons:
10
Transfer Function:
TANSIG
View
Restore Defaults
Help
Create
Close



MATLAB Code for spiral data generation and sampling

Equation of the spiral:

$$\mathbf{r} = \pm\sqrt{\theta}$$

```
clear all
clc

t=linspace(1,4*pi,100)';
r1=sqrt(t);
r2=-sqrt(t);

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']';


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

Xtrain=[X1_new X2_new]';
Xtest=[X11_new X21_new]';

ntrain=length(X1_new);
ntest=100-ntrain;


class = [ones(1,ntrain) 1+ones(1,ntrain)]';
q = [ones(1,ntest) 1+ones(1,ntest)]';


s=svmtrain(Xtrain,class,'Kernel_function','polynomial','polyorder',8,'s
howplot',1);


p=svmclassify(s,Xtest)


miscl=sum(abs(p-q));
accuracy=100-100*miscl/(2*ntest)
```

Question 3

Parzen Window, K Nearest Neighbors and Nearest Neighbors technique

AIM : In this experiment, we use three non-parametric techniques, namely, Parzen Window, K Nearest Neighbours, and Nearest Neighbour for classification of data, and compare their performances.

We used the same data as was used for Question 2.

1) Parzen Window Technique

We developed Matlab code for Parzen Window technique for two types of window functions – Gaussian and Rectangular window functions. To resolve ties, we made use of the priors of the data. In the data that we used, we had the same number of points in both the classes, and hence the priors were 50%. In the code, we used a random number generator to generate 1 or 2, and assigned the test point to class 1 or class 2 accordingly.

EXPERIMENTS

We performed experiments for data by varying the amount of separation, and by varying the window size, for two types of window functions. Also, we varied the amount of data available for estimating the density function from 10% to 90%. In this section, we have listed the various experiments conducted, the plots of the data along with the misclassified test points, in red crosses, for both the methods, and their respective performance measures.

1) 2-D slightly overlapping data

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

%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

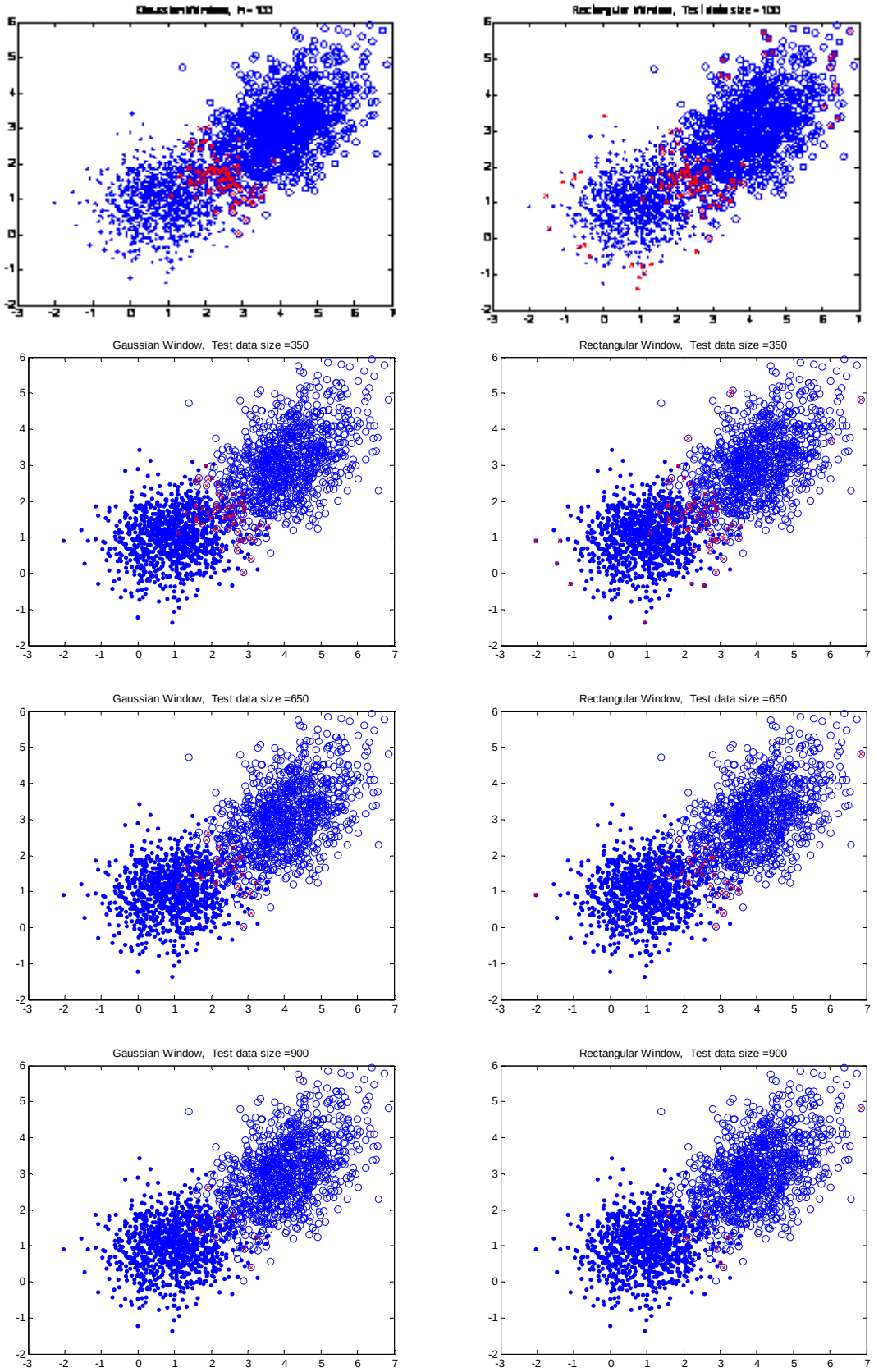


Figure (i) showing the plots of Parzen Window technique using h=0.75.The left half contains results for classification using Gaussian window and the right half contains rectangular window

From the tables and figures we can observe that the performanceof Parzen Window technique using Gaussian window function is a smoother estimate of the probability at a given point since it takes into account all the data points in the set by giving smaller weights to the points which are far away, and higher weights to closer points. On the other hand, the rectangular window just counts the number of points which are within a particular sized window around that point, and assigns the test point to a class which has a bigger count.

Moreover in the case of points which are far away from their actual class means (see the figure) we can observe that rectangular window method misclassifies the samples while Gaussian window performs much better. This can be especially seen in the first set of figures(100 training points). This happens because there are no points in a window of 0.75 around those points and the points are classified by their priors, which obviously gives worse results, unless there is a huge difference between the priors of the two clases.

For smaller window sizes the rectangular window seems to do better but not better than the Gaussian window function. There does not appear to be any appreciable effect of amount of training data on the performance of the classifier.

2) Overlapping 2D Data

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

%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

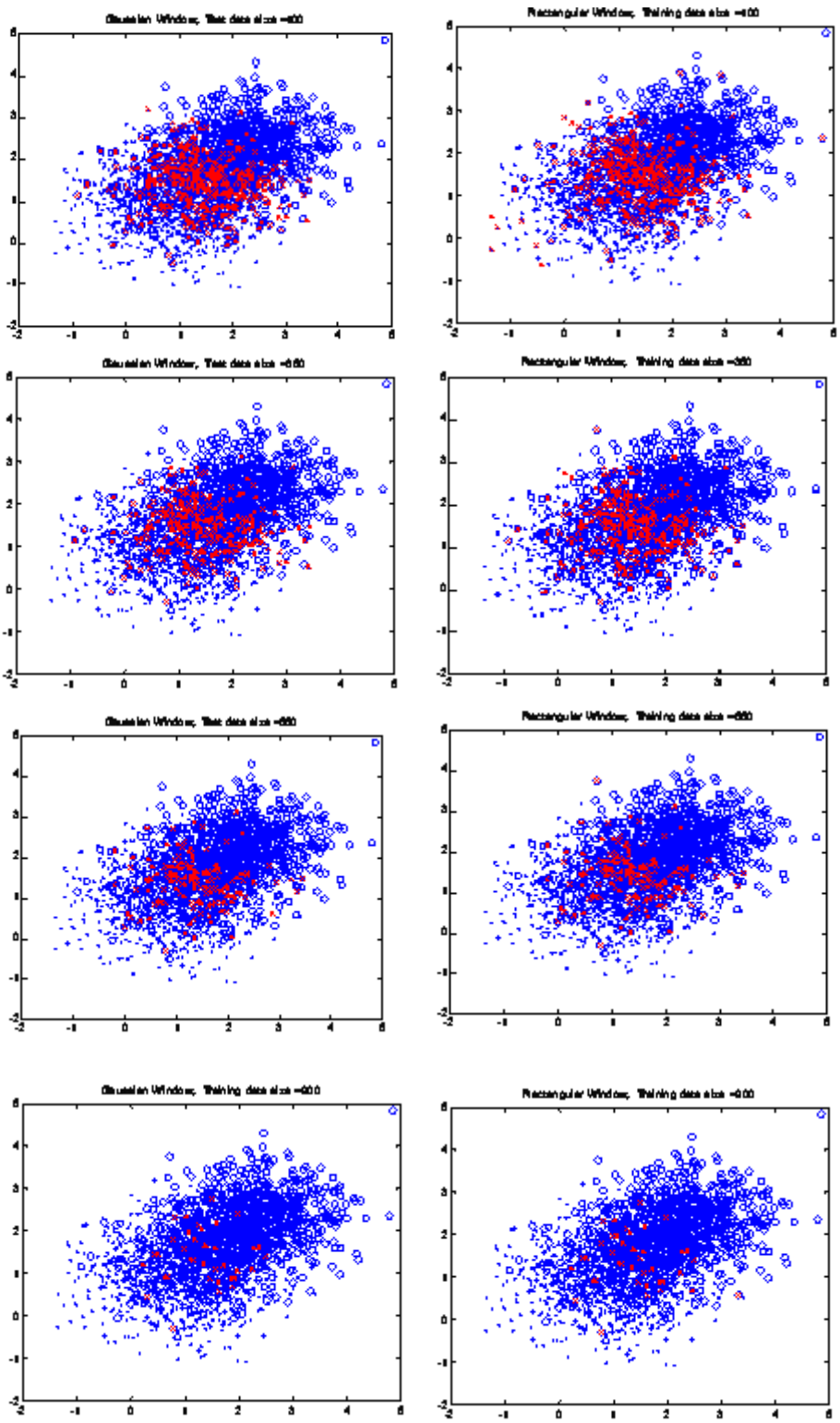


Figure (ii) showing the plots of Parzen Window technique using h=1.25. The left half contains results for classification using Gaussian window and the right half contains rectangular window

In the above case we have chose overlapping data samples. As expected we start getting incorrect estimates of the density at the test points leading to large error as show in the tabular columns. However even in this case we notice that the Gaussian window function still gives a better estimate as it takes into account the contribution of all training points. However we can say that in the case of overlapping data the classifier as such does not do very well in the overlapped regions.

3) 3-D overlapping data I

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

%Train / 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

4) 3-D overlapping data II

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

%Train / 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

Note large numbers of correct samples are misclassified as there are no neighbours in the window leading to the flip of a coin method as we have assumed equal priors.

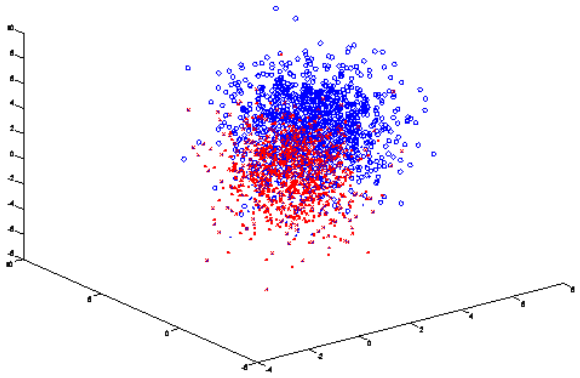
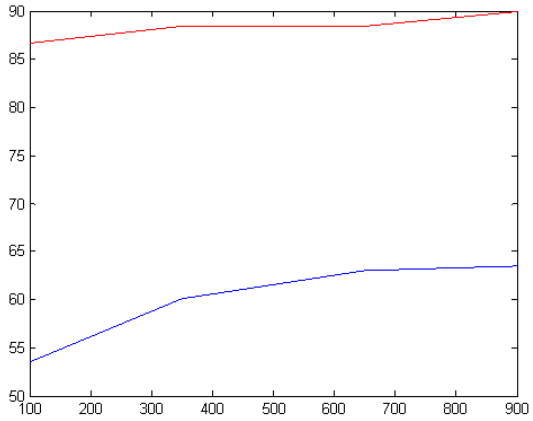


Figure (iii) Sample plot for h=0.50 and 10% training data



Fig(iv) The error plot for h=0.50

3-D more overlapping with shifted mean

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

%Train / H	10		35		65		90	
	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

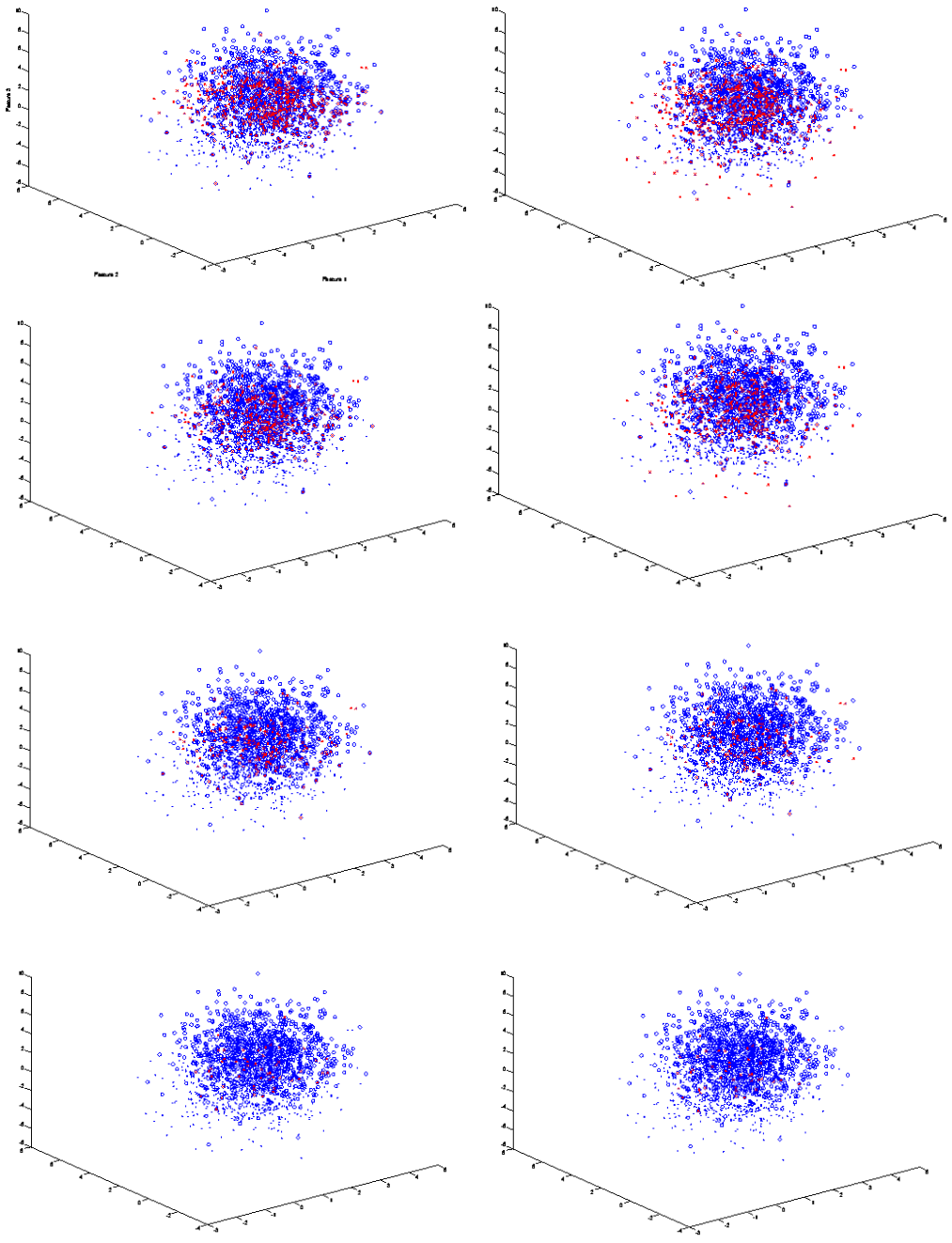


Figure (v) showing the plots for h=1 for Gaussian and rectangular window for 900,650,350 and 100 training samples respectively.

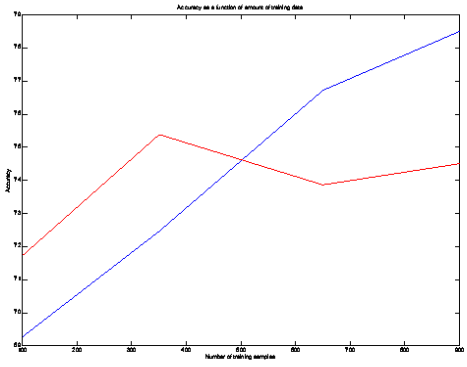


Figure (vi) showing the accuracy as a function of training samples

We can observe that the results are similar to that of the 2D case, and the Gaussian window does much better than the rectangular window. As the amount of overlap increases, the performance of the classifier gets significantly poor. Also, the amount of training data (i.e. data available for estimation) does not seem to have a significant effect on the performance of the classifier in both the windows.

We also observe that increasing the window size results in a significant improvement in the performance of the classifier using rectangular window function, wheras its effect is not much pronounced in the Gaussian case. This makes sense because, in this type of data, most of the error made in classification using rectangular window function, is because of the misclassification of the outlying points (i.e points which are too far away from the means of their respective classes), and by increasing the window size, we basically increase the amount of data points in the window, thus, reducing the chances of having a tie. Thus, we need to choose an optimum window size, which depends on the type of data, to maximize the classification accuracy.

K Nearest Neighbors and Nearest Neighbour Technique

We have operated on exactly the same data as in Parzen Window technique. The programs are implemented with the help of Matlab and the code is attached at the end of the section.

EXPERIMENTS

We performed experiments for data by varying the amount of separation, and by varying the value of K, for two types of window functions. Also, we vary the amount of data available in advance (training) from 10% to 90%. In this section, the various experiments which we conducted are listed, the plots of the data along with the misclassified test points (in red crosses), for both the methods, and their respective performance measures.

1) **2-D data with very less overlap.**

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

The results are tabulated for the nearest neighbour(K=1) and other values of K.

K \ %training	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

We also conducted experiments on the same data using Manhattan distance

%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

We do not observe a significant improvement of 1 distance metric over the other.

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

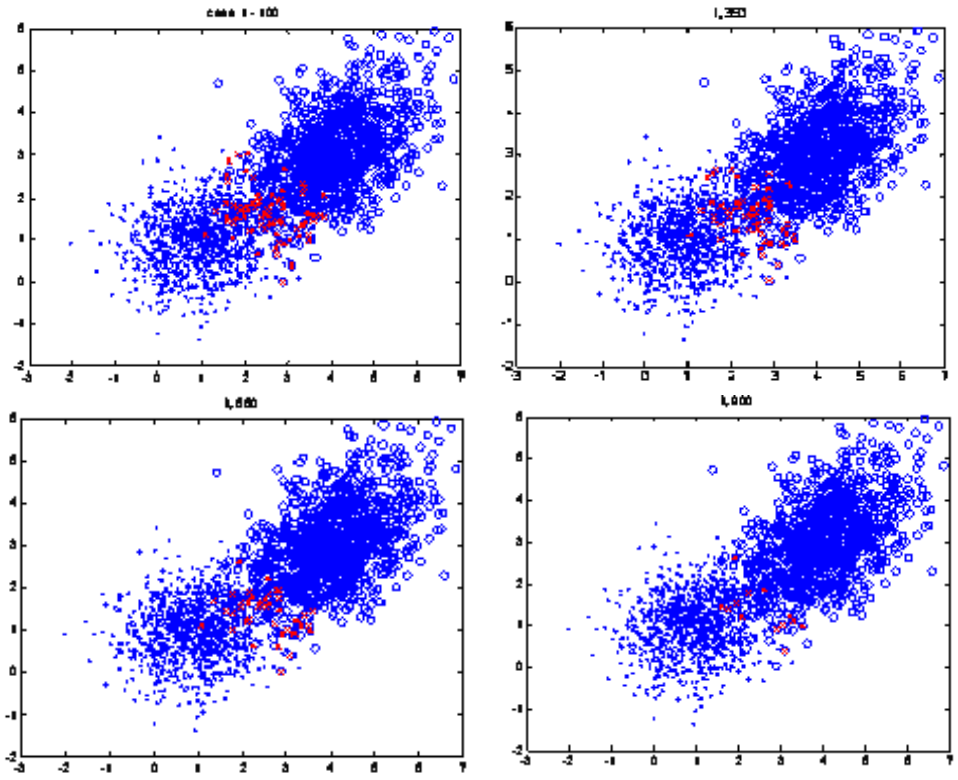


Figure (vii) the plots of Nearest neighbour with eucledian distance metric

In the case of K nearest neighbors we observe that classification accuracy varies as K is changed. We observe that K=1 does not perform as well as K=3,5 or 7. This indicates that K=1 is not as robust as taking higher values of K, which is quite intuitive.

To prevent ties, we chose odd values of K.

We observe some change in accuracy as the number of samples is increased but this is not appreciable to draw a conclusion about any correlation between increases in training samples against accuracy.

We experimented by using Manhattan distance as a metric and found that it performs worse than the Euclidian distance in this case.

---

2) 2-D data with more overlap.

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

K \ %training	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

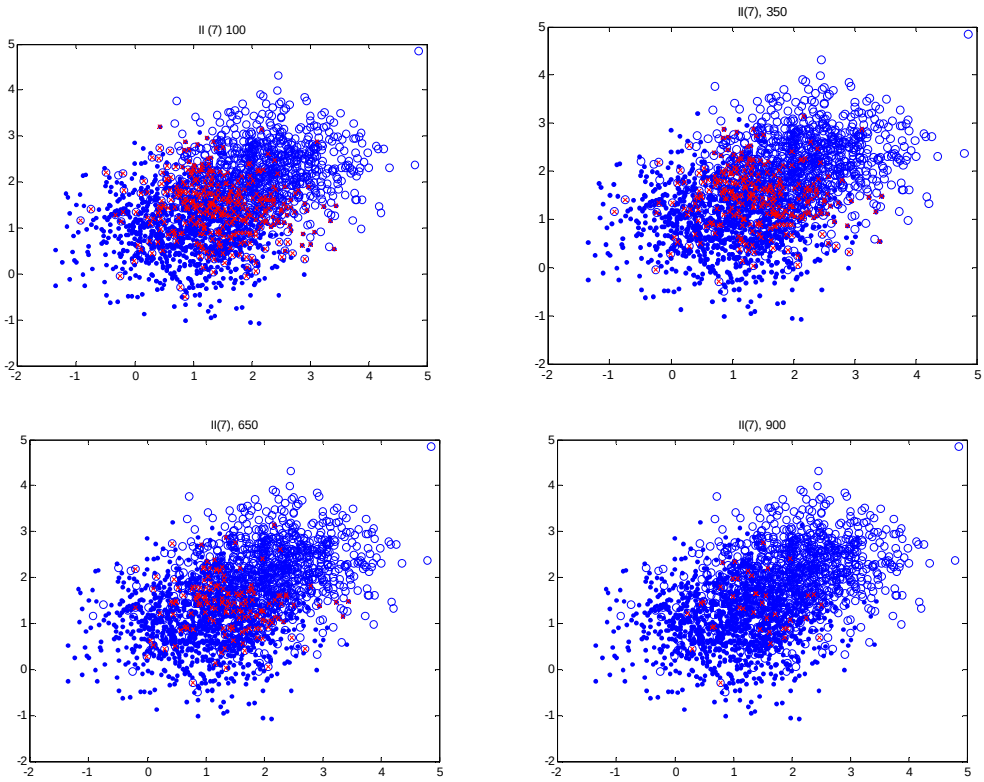


Figure (viii) showing K=7 classification for overlapping data

Due to more amount of overlap, the performance of all the classifiers has reduced, as expected. Again, we see that K=1 is not as good as higher values for K, and the performance increases with an increase in the value of K.

3) 3-D data with overlap

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

The results for various K are shown below:

%Tr / 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

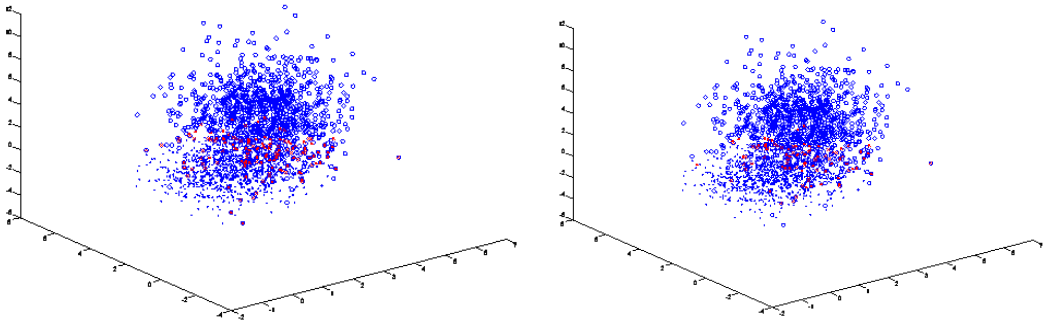


Figure (ix) showing K=7 classification for overlapping data with 100 and 350 training samples

We next consider 3-D dataset, with more overlap, by increasing the variance of one of the data sets.

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

Results for KNN

%Tr / 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

We also consider data with more overlap, by bringing the means closer together.

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

%Tr / 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

Here again, we can see that the nearest neighbor technique performs poorly as compared to the other values of K, thus proving that it is not a good classification technique, which is quite intuitive.

CONCLUSIONS

Based on our experiments on 2-D and 3-D for varying amounts of overlap using the Parzen Window and K nearest neighbor technique, we find that

1) Parzen Window using a Gaussian window function results in better performance than that using a rectangular window function.

2) K nearest neighbors out performs Nearest Neighbor technique and is more robust.

3) KNN is better than the Parzen window technique using rectangular kernel because it does not misclassify the points which lie too far away from the means of their respective classes.

4) Comparing Parzen window using Gaussian kernel, with KNN, in our experiments we observe that in several cases Parzen windows perform better than KNN. This may be contrary to what is observed in practice. We believe that this is because we experimented with mainly overlapping data, so as to determine which classifier performs better when the classification problem is very difficult. We have used 2-D and 3-D synthetic data mainly with the objective of ease of visualization.

5) The performance of SVMs and ANNs is very close to the performance of Parzen Window, KNN and Nearest Neighbor techniques, for the overlapping data that we considered.

In general, we conclude that there is no universal classifier, which can do well in all the classification tasks. The choice of the classifier is highly dependent on the nature of the data at hand.

MATLAB CODE for Parzen Window and KNN

The following codes are contain "script files" we wrote up so as to speed up testing of our data. The script files essentially take the data and break it up into training and test data performs classification and reports the accuracy of the classifier and also plots the results of classification and the error plots.

Script file for 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
```

Code for Parzen Window based classification using rectangular and Gaussian window functions.

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

%variables to hold size of input data ch gives the type of window
function to use
[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
```

```
Code for K nearest neighbors. It takes the data and the number of
nearest neighbors as inputs and does all the necessary tests.

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] S. Osowski, K Siwek, T.Markiewicz, MLP and SVM Networks –a Comparative Study, Proceedings of the 6th Nordic Signal Processing Symposium-NORSIG 2004
[2] S.E.Fahlman, C Lebiere, The cascade-correlation learning, in "Advances in NIPS2",D. Touretzky,Ed.,1990,pp.524-532