QUESTION I

Fisher's Discriminant

The objective of this experiment is to see what happens when the solution of

$$J(w) = \frac{w^t(S_b)w}{w^t(S_w)w}$$

under the constraints that ||w||=1 which is **w**= (S_w^{-1}) (m1-m2) is instead set to **w**=m1-m2.That is we are forcibly setting the within class scatter matrix to unity in the solution to w. This makes the line which is used to project the data dependent only on the means of the 2 classes.

Experiments

The motivation of the experiments were to see how the modified **w** performs and whether there are cases where this can yield a better performance. We measure performance by finding the w vector and projecting the data onto it and seeing if the separation surface does the task of classification of the projected data points satisfactorily.

For all experiments we compare the performance of the established method of finding **w** and compare it against the **w** which is only dependent on the means. We have written the code which takes the data from the 2 classes finds the w vector and also the separation hyper plane and plots the data, the projections and the separation surface.

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.

The structure for the next sections will be the data used followed by error rates and the plots obtained. We have performed experiments for 2-D and 3-D overlapping and non overlapping data. The 2 classes of data have **1000 points** each.

2-D Data

1) **Well separated** data

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

Classification error based on criterion $w^t X+w0>0$ class 1 and $w^t X+w0<0$ Class 2

Result for **w=** S_w^{-1} ***(m1-m2)**

Error = 0%

Result using wo=m1-m2

Error =0%

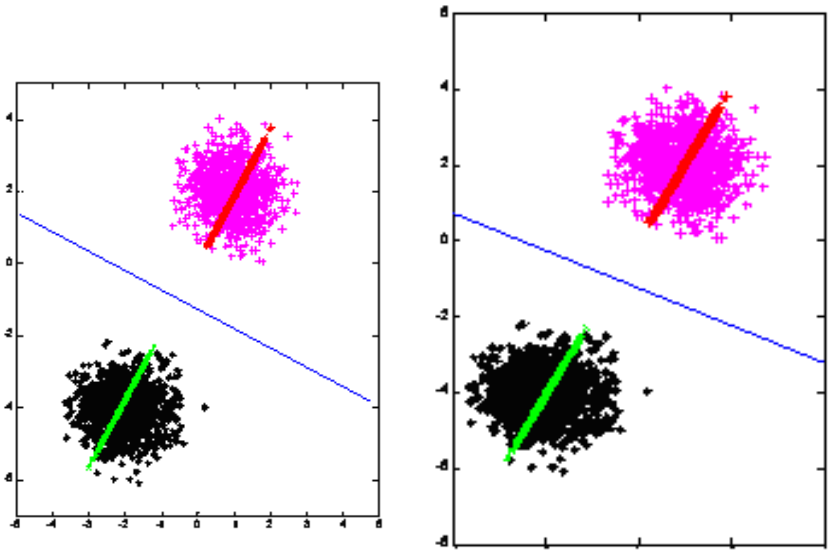


Figure (i) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

In this case there is no difference in the error computed. This is because the data is very well separated out and hence it will not be very constructive to interpret results about which method is better.

2) **2-D separated data which leads to error using m1-m2.**

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

Result for wo=(Sw^-1)*(m1-m2)

Fisher Error = 0%

Using m1-m2 alone

Error= 3.8500 %

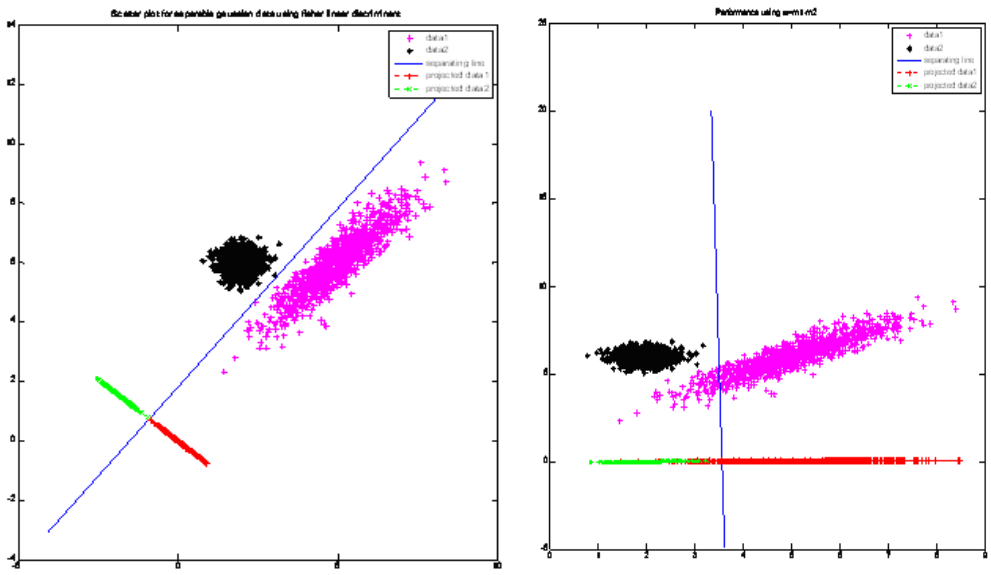


Figure (ii) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

This case clearly motivates why we should not use m1-m2 for finding the optimum weight vector. Clearly the method does not take into account the variance of the data resulting in the **w** vector being along the direction of means. The error using just m1-m2 is close to 4% while fisher draws the hyper surface such that there is no error.

3) **2-D OVERLAPPING data**

In order to experiment further we decided to see what happens when the 2 classes have data that is over lapping.

	Data 1	Data 2
Mean	[2 8]	[6 5]
Variance	$\begin{pmatrix} 8.48 & -0.05 \\ -0.05 & 7.6 \end{pmatrix}$	$\begin{pmatrix} 8.64 & 0.15 \\ 0.15 & 8.53 \end{pmatrix}$

Fisher
Error = 16.75%

Modified Fisher
Error = 16.85 %

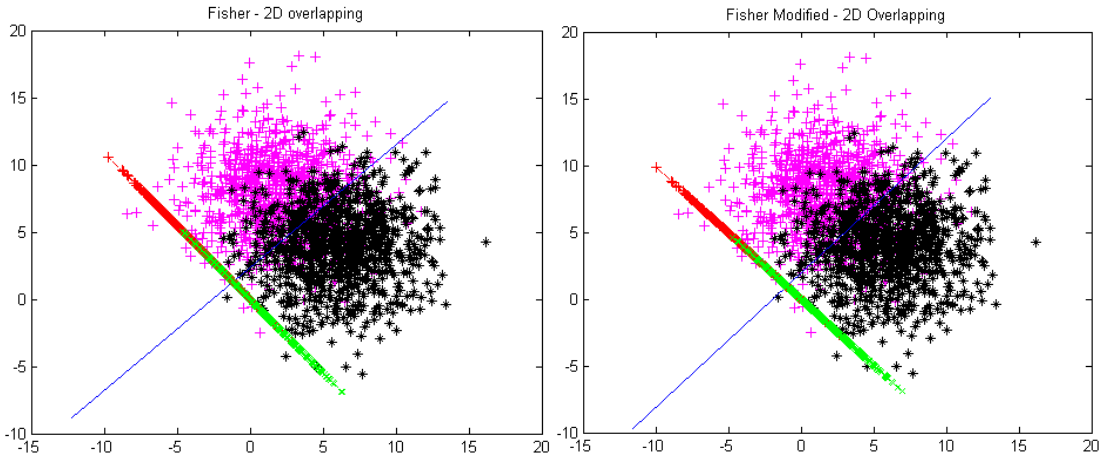


Figure (iii) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

In this case fisher's method once again gives us lower error rate. The reason why the m1-m2 method is also giving a comparable error rate is that the actual **w** vector is very nearly equal to **m1-m2** in this case. But fisher's method is precise and finds the optimum solution to maximize the between class means and minimize the within class scatter.

4) **2-D uniform data**

The motivation for the following data set is to find some data for which the fisher's method actually gives a worse performance. We have constructed the synthetic data **for each class** by combining 2 sets of distributed data - one horizontal and one vertical. The variance of such data is very large along X direction as can be seen from the plots below.

	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 's Method gives an error of 16.025%
Modified Fisher gives a error of 0%

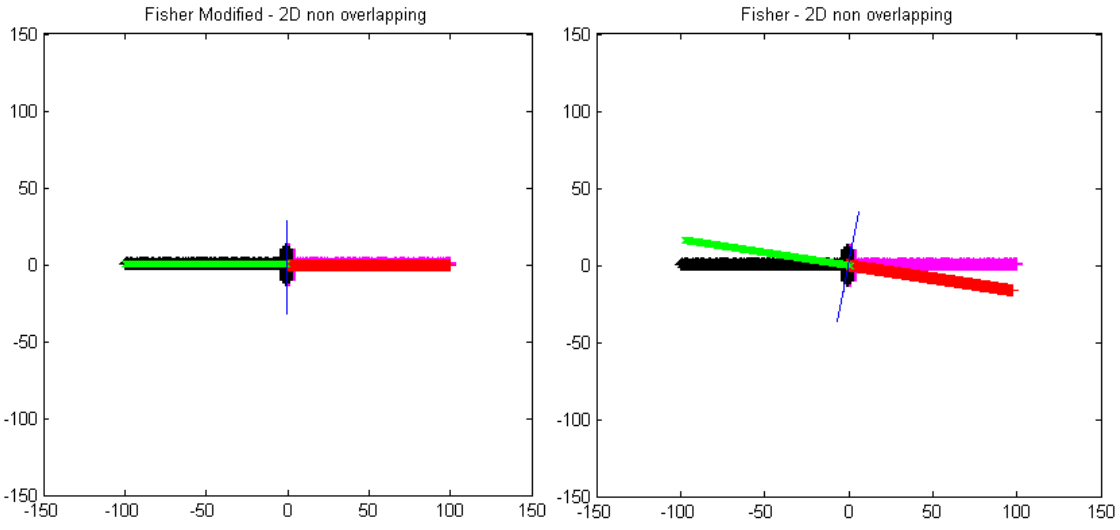


Figure (iv) showing the projections and separation hyper plane for the fisher discriminant and for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

The data and its projection (enlarged view for the MODIFIED fisher's method)

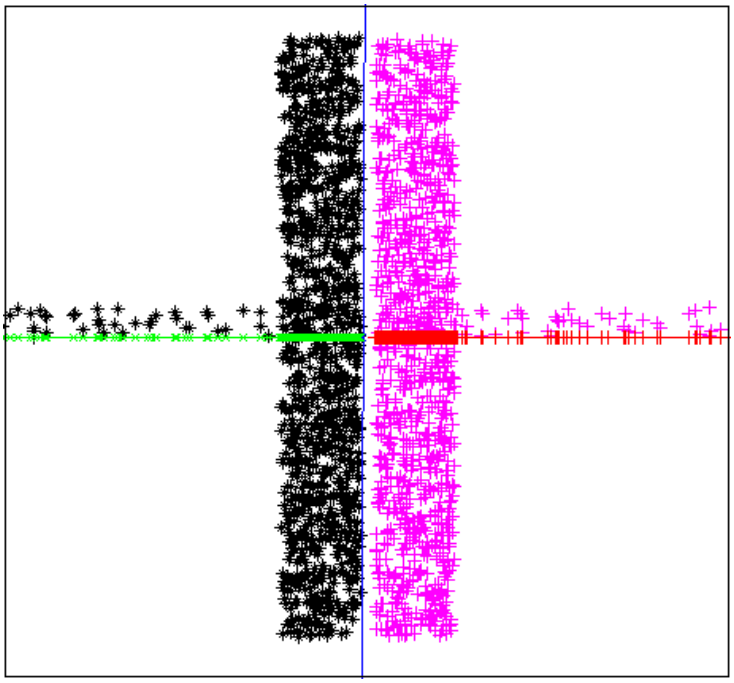


Figure (v) showing the projections and separation hyper plane for the modified fisher method respectively. The pink and black are class I and class 2 data and the red and green are the projections of the data. The blue line is the separation line

In this case **m1-m2 seems to be a better w,** measuring the classification error by the method specified earlier. This is expected as we can see from the figure that the projection plane is clearly along the line joining the means of the 2 classes. However this data has been generated in this manner and we are not sure how about likely we are to come across such a distribution in practice.

Operations on 3-D data

Next we wish to investigate what happens when we operate in a higher dimensionality. We choose 3 dimensional features so that we can visualize the results easily.

1) **Well separated data**

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

Classification Error on the projected data using Fisher's method=0%
Error using the modified expression for **w** =0%

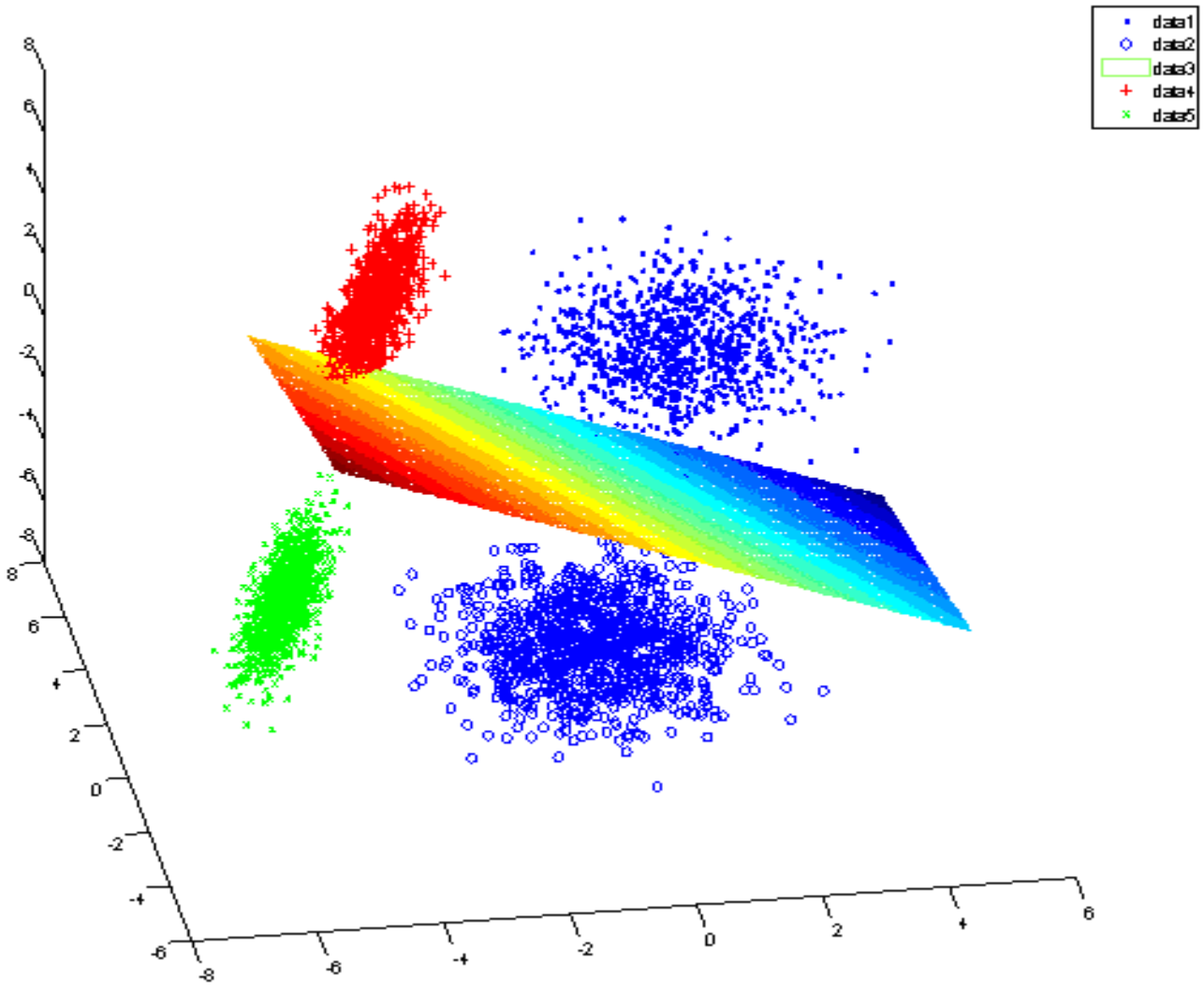


Figure (vi) showing 2 classes , the offset projection and the separating plane

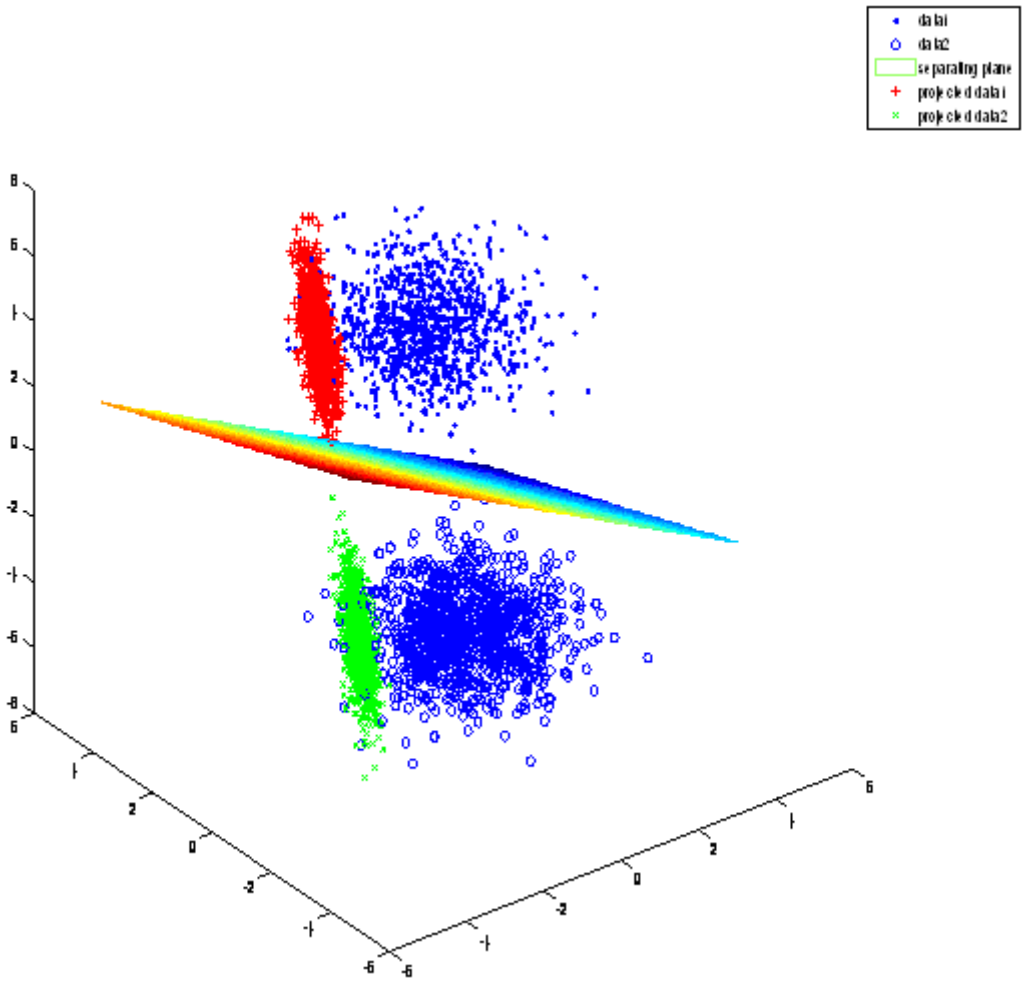


Figure (vii) showing 2 classes , the offset projection and the separating plane for w= m1-m2

As expected the result is same as the 2-D case of well separated data.

2) Overlapping data

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

Error using fisher = 10.65 %
Error using the modified fisher= 11.85%

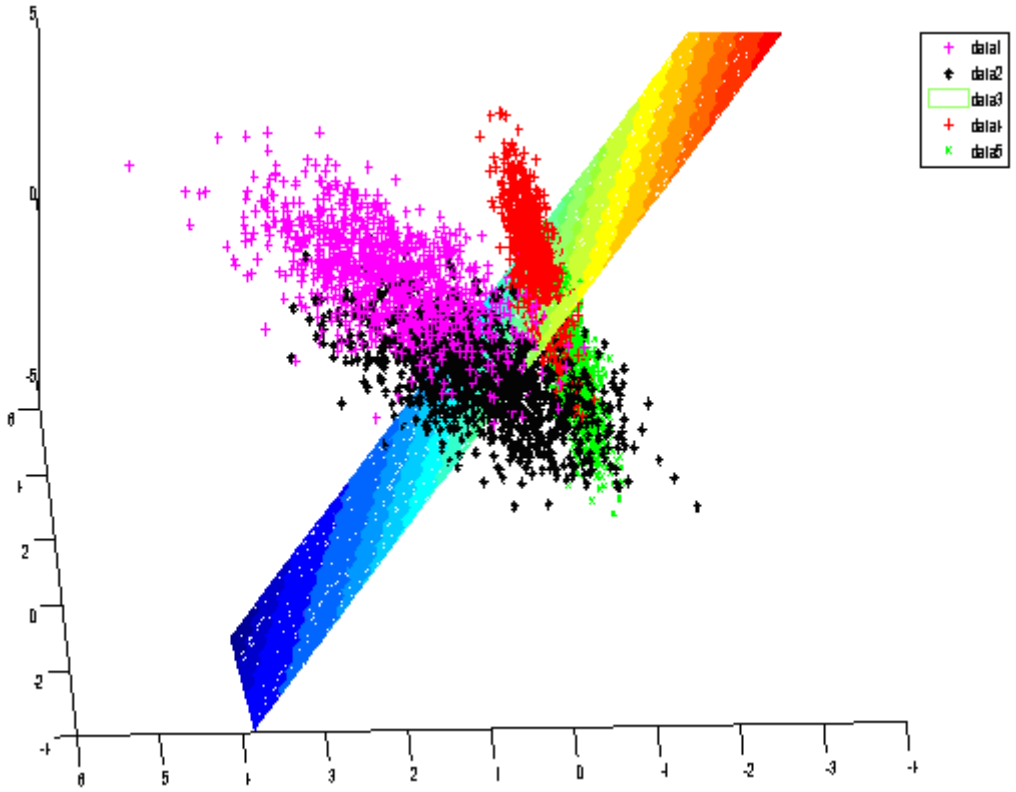


Figure (viii) showing 2 classes in pink and black , the offset projections in red and green and the separating plane

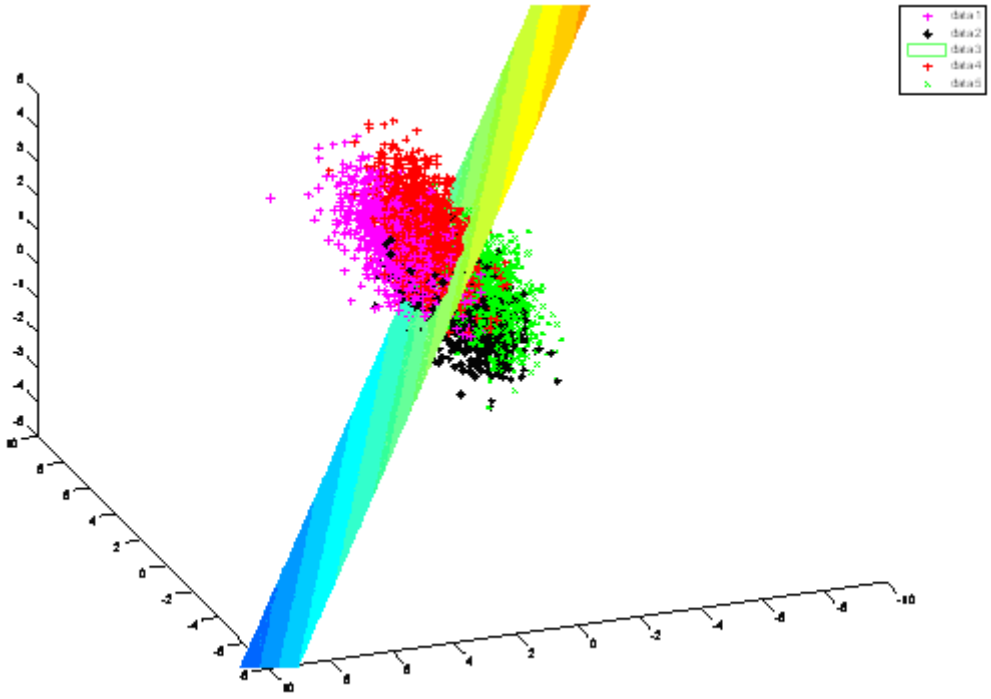


Figure (ix) showing 2 classes in pink and black , the offset projections in red and green and the separating plane using the w=m1-m2

In this case as is in 2-D case fisher's method performs better.

3) 3D Non-overlapping Data

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

Error using fisher = 0%
Error using modified fisher=11%

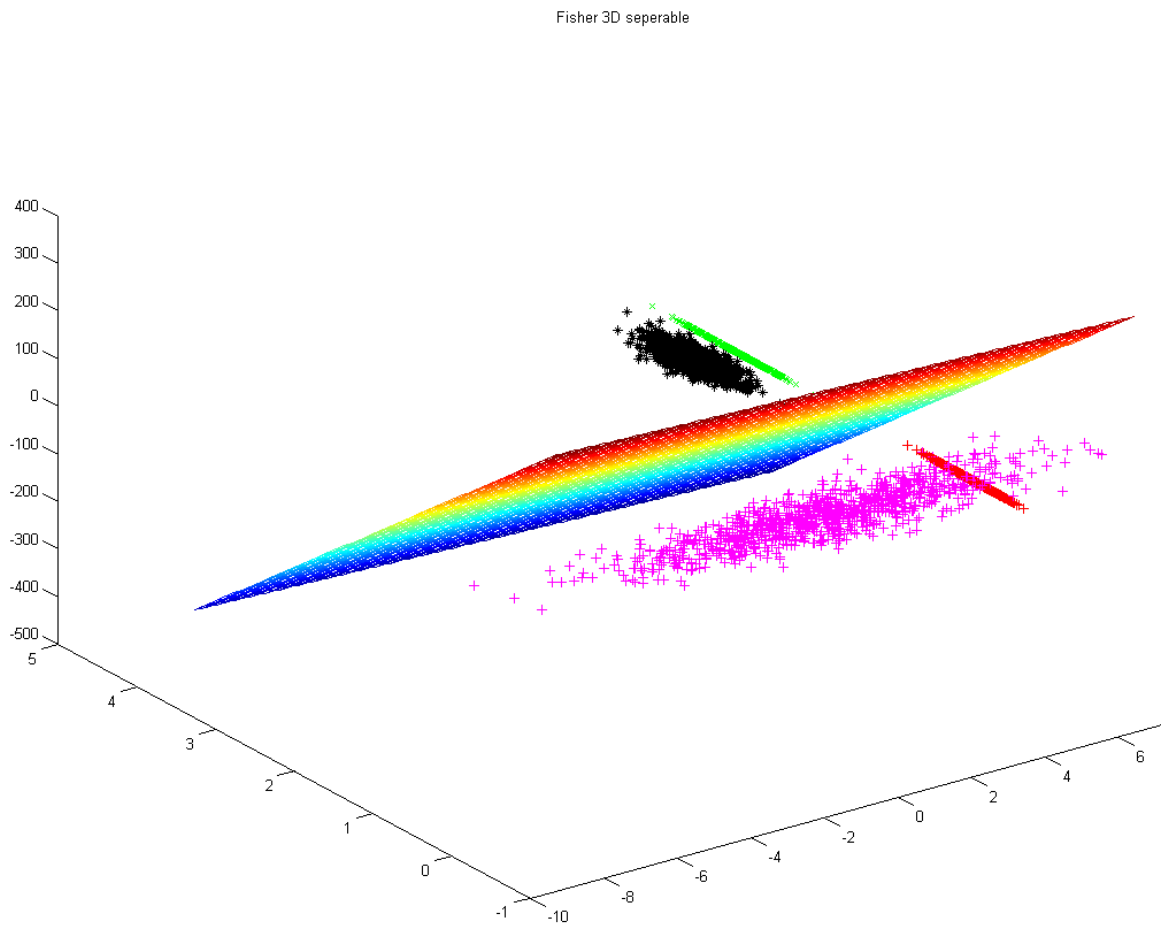


Figure (x) showing 2 classes in pink and black , the offset projections in red and green and the separating plane using the normal fisher method

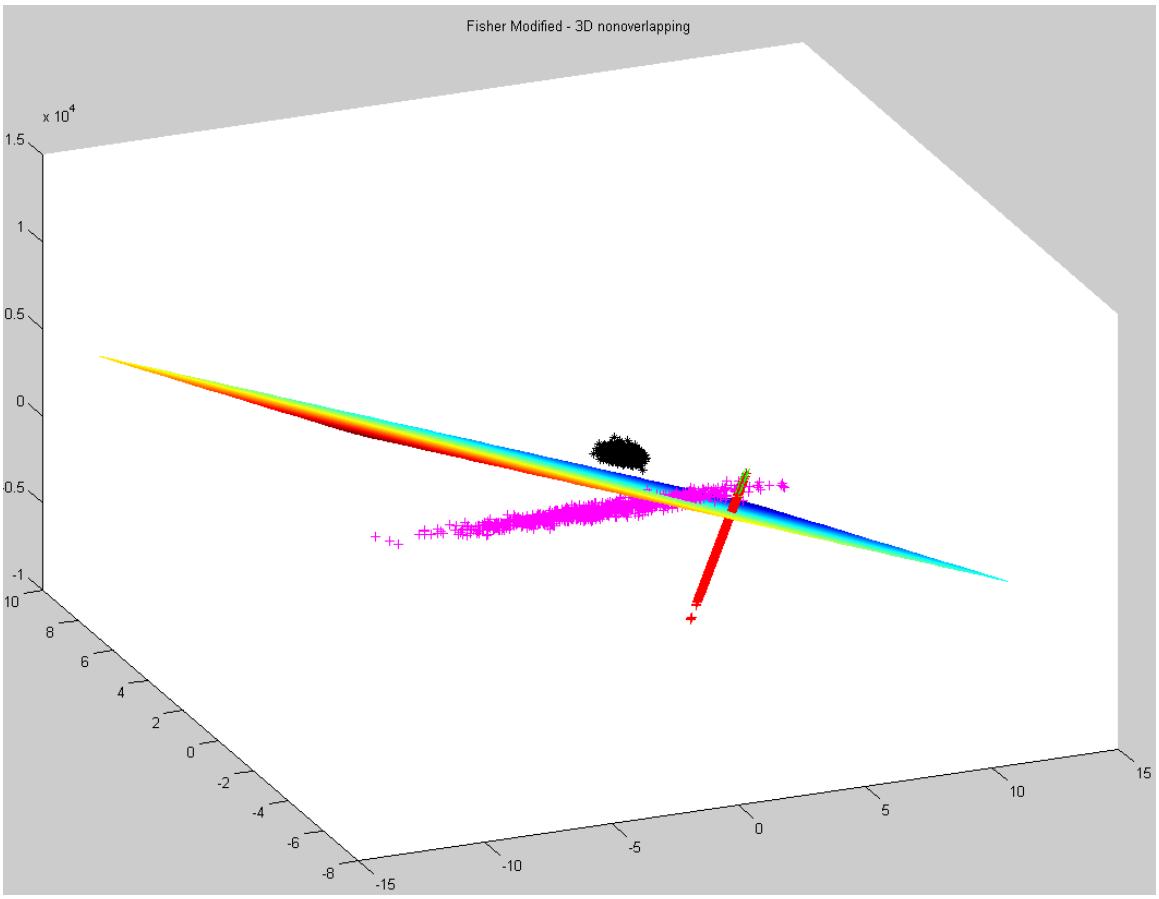


Figure (xi) showing 2 classes in pink and black , the offset projections in red and green and the separating plane using the w=m1-m2

Once again in this case (just as in 2-D) the m1-m2 method finds a poor solution by not taking into account the within class scatter matrix. In this case fisher performs much better and hence should be the choice of classifier.

Conclusion

As a conclusion it can be seen that in most cases the established fisher's method of finding the weight vector works better. It appears that choosing m1-m2 is trivializing the problem of classification.
However we have shown that there are anomalies and in such cases fisher's method yields a poor performance. This is clearly seen from 2-D case 4 of our experiments.

However it does not appear likely that such data types can occur frequently in practice and hence if we are to make a choice, we believe that fisher's well established method is a better option.

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

Support Vector Machines and Artificial Neural networks

Before beginning we discuss a brief description of the tools we have used. For Support Vector Machines and Neural Networks we have made use of in built tool boxes in Matlab®.

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.

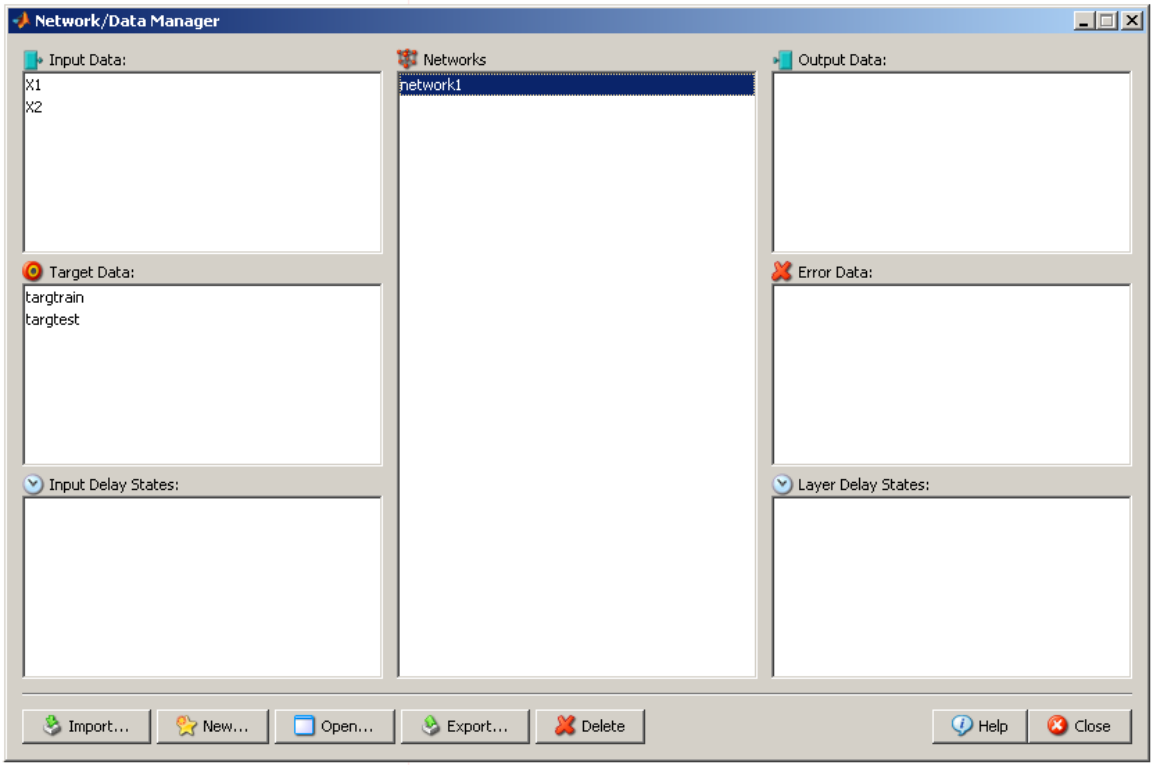
NNtool in Matlab®

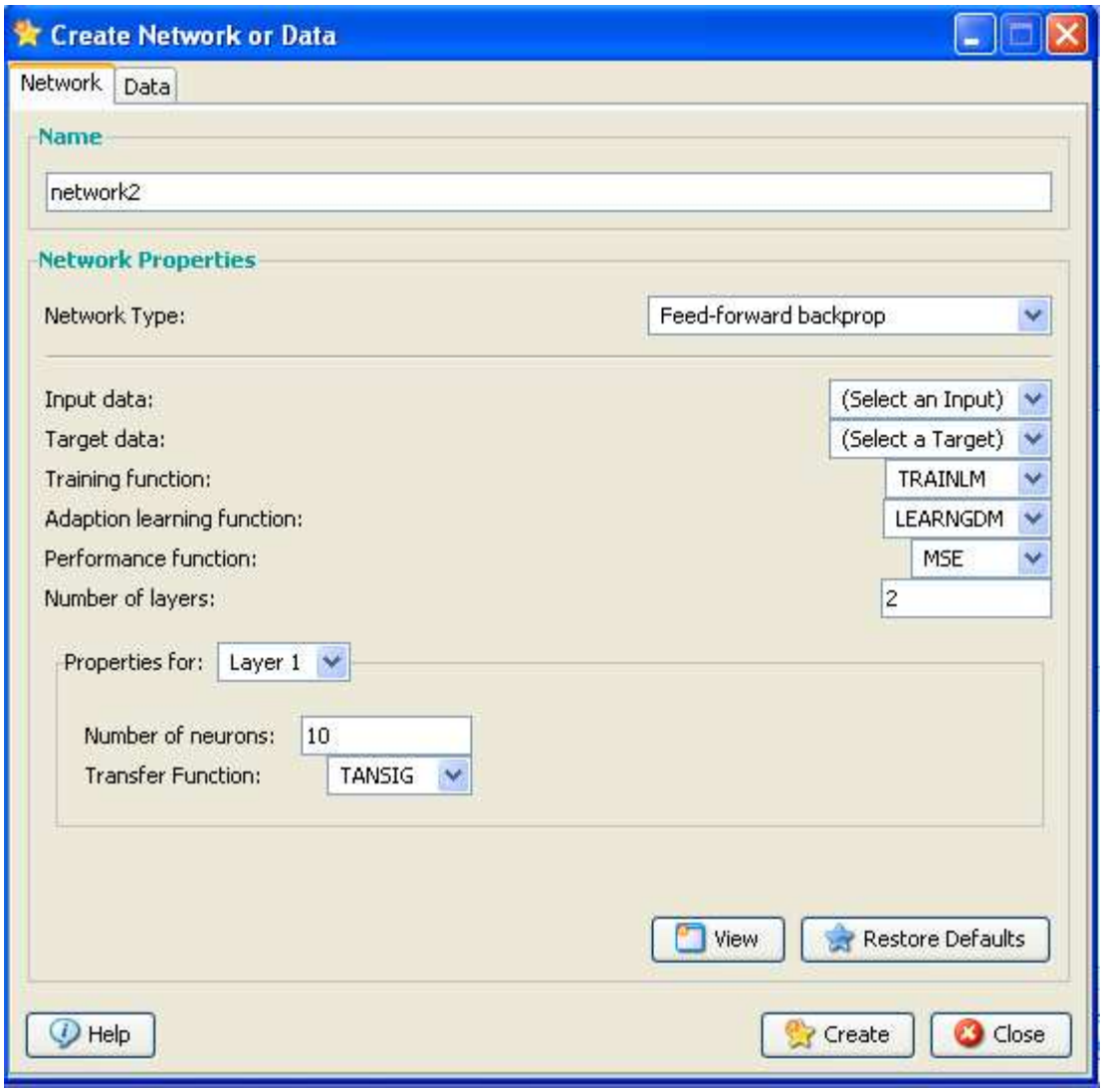
This tool is a Graphical User Interface Tool. The user can specify several parameters of the neural network such as the activation function, the initial weights, the training and test data, the number of epochs and the target data for the giving training set.

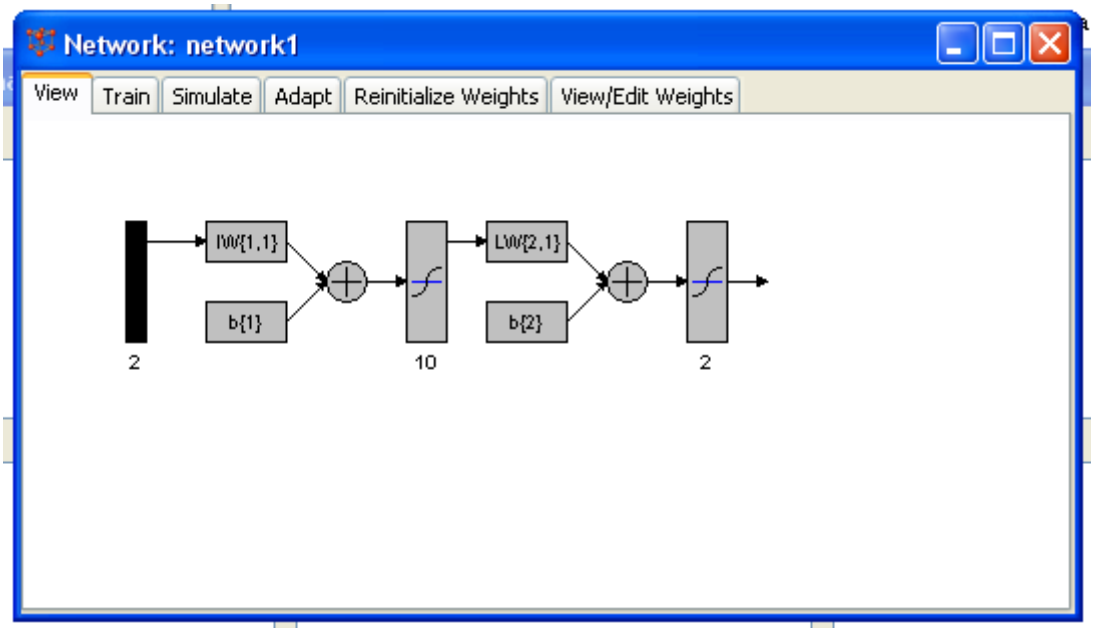
The tool expects the user to give the training data in the form of a d X n matrix where d is dimensionality of the data and n is the number of training vectors. It expects “target” data in the form of a 1 X n vector each entry specifying the class of the corresponding training sample. Once this data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

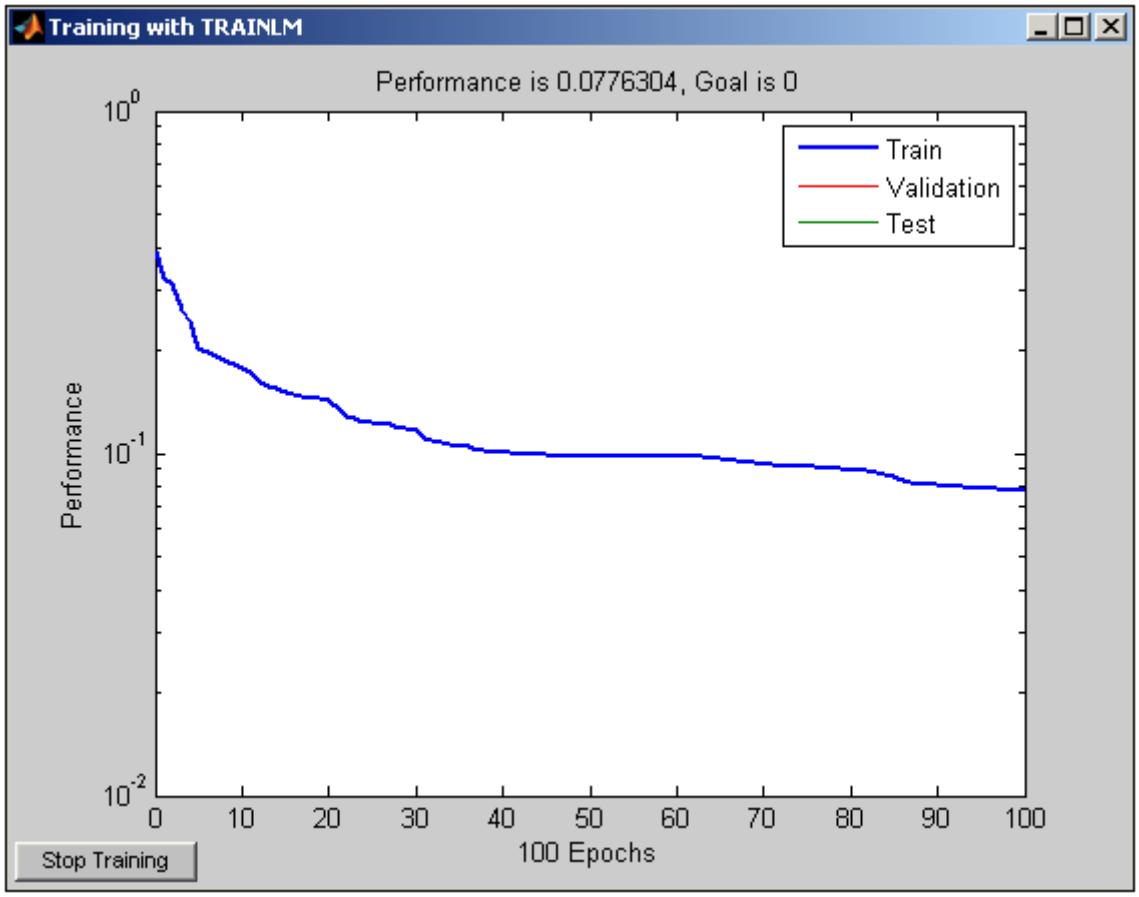
Following are some snapshots of the tool




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



We can also get sample error plots like the one shown below



In our experiments we have varied the number of hidden layer neurons and the number of epochs in the network. We have made use of a feed forward back propagation network

with 2 layers (excluding the input neurons). We have used the Levenberg-Marqardt function for training and the default value of LEARNGDM for learning function along with Mean Square Error for performance. The tansigmoid is used as a activation function for the neurons.

Experiments

We have conducted experiments to evaluate the performance of both classifiers. We use 1000 points for each class and vary the amount of training and test data in the proportion 10-90,35-65,65-35 and 90-10 respectively. We note the performance of SVM under different Kernels. We also experiment with number of neurons in hidden layer. The format is data followed by figures (for SVM) and then a table showing results for both SVM and Neural networks for different proportions of training and test data.

1) 2-D overlapping data

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

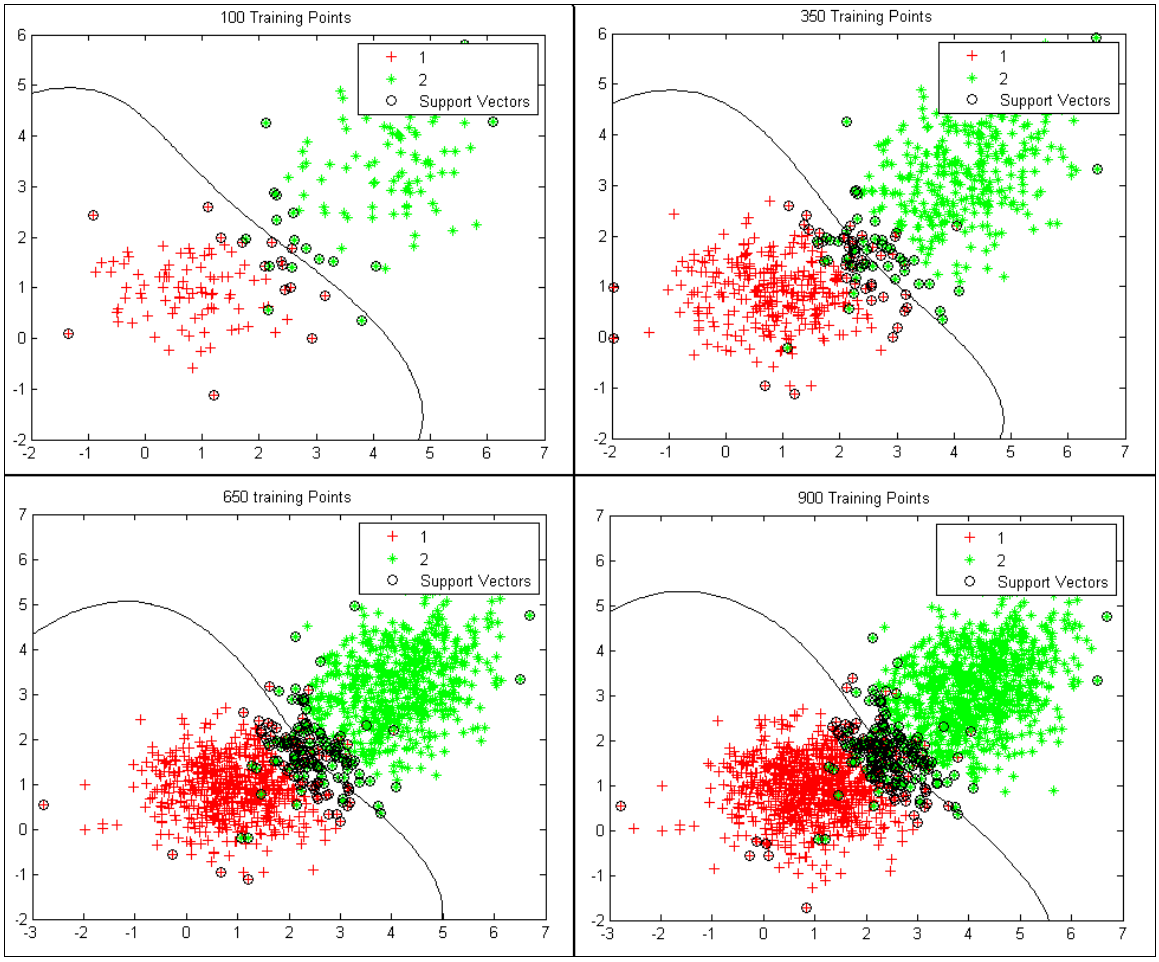


Figure (i) Training data and support vectors for different proportions of training data using ***radial basis function***

# of Training Samples	100	350	650	900
Accuracy Neural Network (10 neurons in hidden layer)	93.72	95.42	96.85	94.83
Accuracy RBF Kernel	94.9444	94.8462	95.4286	96
Accuracy 2^{nd} degree poly. Kernel	94.6667	94.7692	94.1429	94.5000
Accuracy 3^{rd} degree poly. Kernel	94.5556	94.6154	94.2857	95

Here we have chosen data which has separated means but the variances are such that there is a overlapping region. Here we observe that the accuracy of classification for the support vector machines is between 94 and 96%. We can observe this in the case of both the radial basis and the polynomial kernel (of order 2 and 3) ther performance remains similar. Thus SVM performs well with this type of data.

For a 5 layer ANN ther performance varies from around 93% and increases as amount of training is increased from 10% to 65%. This is expected as neural networks perform better with more training as they become “more aware” of the ranges of data and how to react to it. A point to note is the drop in performance of the ANN when 900 of the 1000 are used for training. This is a good example of over training the network and this is seen consistently across all our experiments. Also the training time for neural networks was more than that of the SVM.(results are not quantified)

2) 2D more overlapping closer means

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

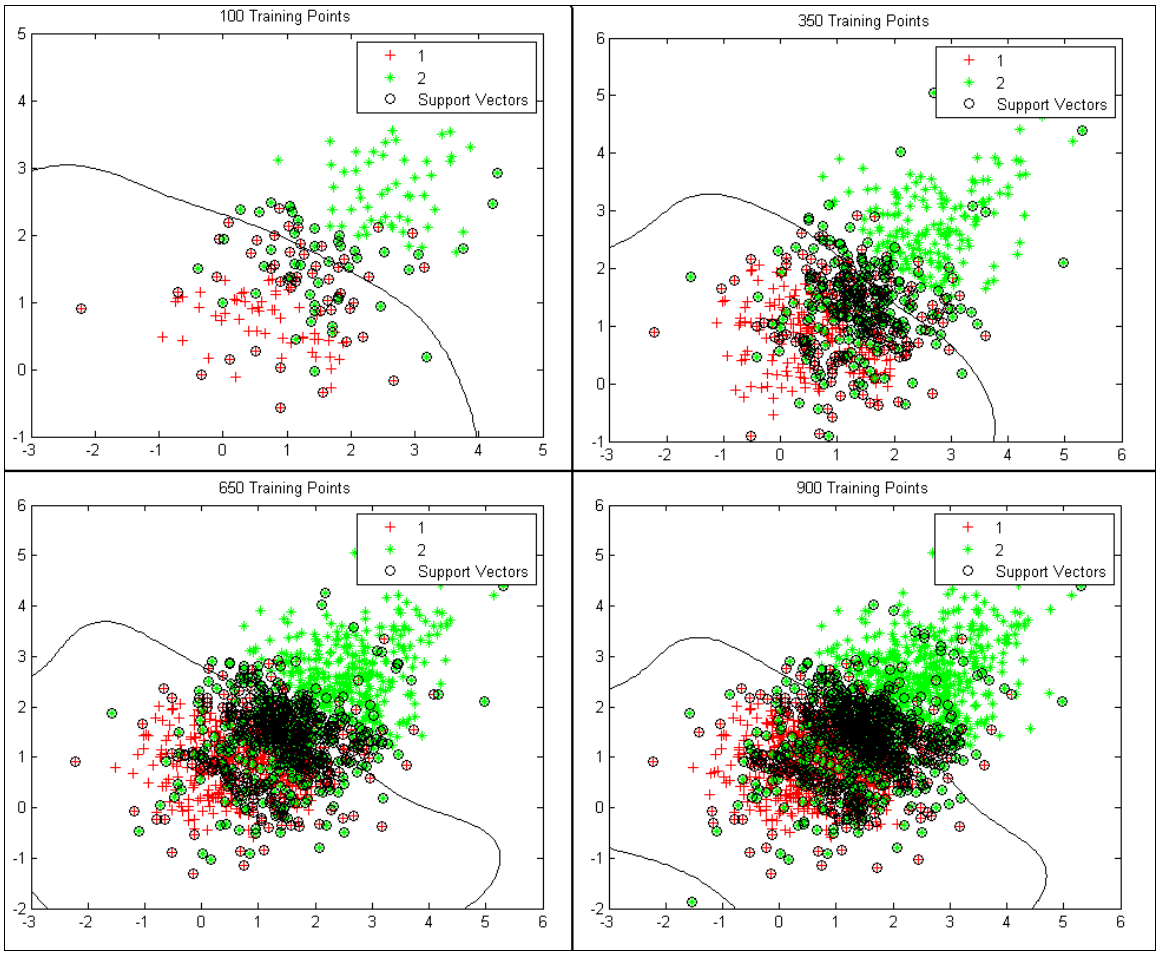


Figure (ii) Training data and support vectors for different proportions of training data using ***radial basis function***

# 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 3rd degree poly. kernel	74.0556	73.5385	NC	NC

Our next experiment was to see what happens when we have overlapping data sets. In this case support vector machines with radial basis functions have a accuracy between 74 and 78%. The polynomial kernels(using 2nd and 3rd order) have a similar performance. We were unable to get convergence for 3rd order polynomial for large amounts of training data.

In the case of neural networks with 5 neurons the performance increases as the training data increases till an extent but there is a decrease when we use 90% of the data for training due to overtraining.

As compared to support vector machines neural networks perform poorly when the amount of training data is 90%. However when 65% of the data is used neural networks perform better. The trade off appears to be that of time. Neural networks take time to train so as to achieve good results.

In this case the data is overlapping bringing down the performance of the classifiers quite drastically as expected.

3D data

The next set of experiments is with the objective of studying the performance of neural network and SVM on 3-D data.

1) 3D overlapping data

	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

2) 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

3) 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 the above three cases we explore data with certain means but with different variance. In the case of SVMs when it attains a accuracy of 92.5 % when 100 tests cases are taken from each class. In the case of data with same means but with more spread there is a decrease in the performance of SVM. Thus we can say that there is a general drop in performance of SVM when the data variances are increased.

Neural Networks in these cases give us poorer performance than SVMs but it is not too large to tip the scales in favour of SVMs when data is overlapped. The difference is not very large to clearly say that SVM's are better. Also if we let neural networks run for large number of epochs or vary the number of neurons we may be able to get better performance. In conclusion the results obtained in 3-D case are similar to those in 2-D case that is neural networks can do well if trained appropriately. However SVM's can do as well if not better in LESSER time.

SPIRAL DATA

We are going to generate spiral data. That is the data for the 2 classes are spirals which coil around each other. This presents one of the most challenging classification problems and is inspired from [1] and [2]. We wish to see how our SVM and Neural Networks perform with such spiral data. We have attached our code for generation of the spiral data at the end of this section.

1) Training using the initial set of points of the spiral

In this case we experimented with what happens when we take the first set of values of the spiral and use it to train a SVM and a neural Network and then test it with the next set of values.

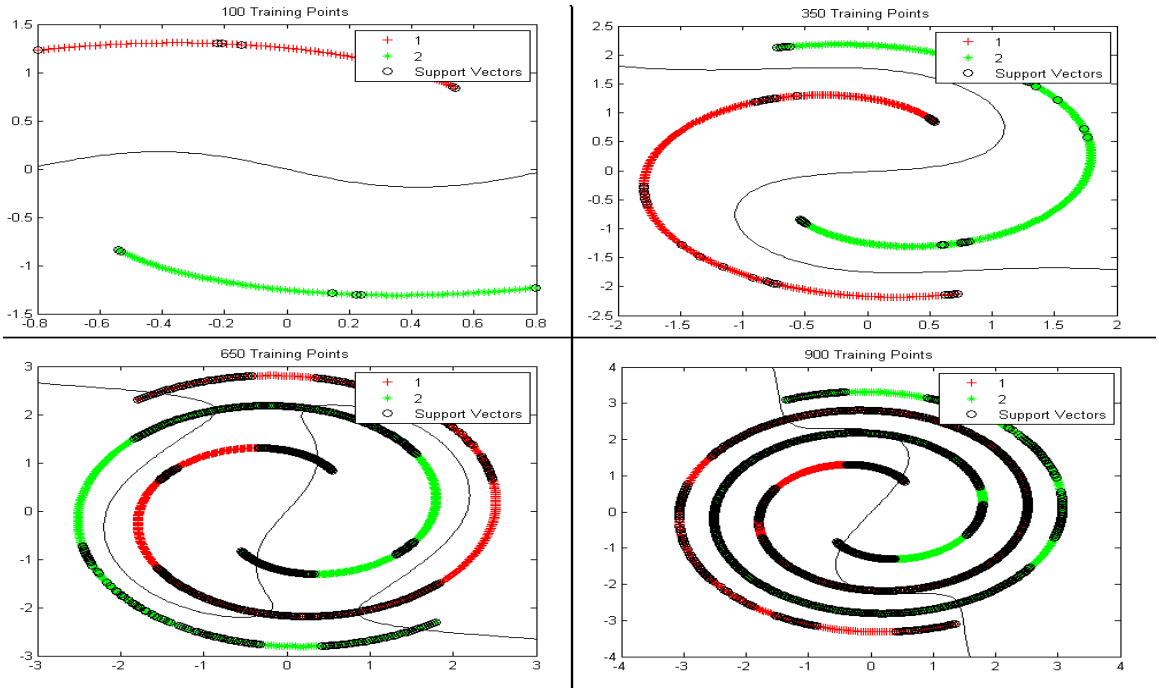


Figure (iii) ***Training data*** *and support vectors for different proportions of training data using* ***radial basis function***

# 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 the performance is very poor. This is because we are giving only the first set of samples to the SVN and neural network. These networks cannot really "predict" what will come after that and hence perform very poorly with the test data which is of not much resemblance to the original data in terms of its position in the 2-D space.

Note especially that when 900 samples are used in SVM we get 2.5 % accuracy. This is because as seen in the figures the surface is drawn such that most of our test data comes in a region in which it is definitely going to get misclassified. As the test data arrives in order most points arrive in this "incorrect" region.

Another point of interest is that neural network does comparatively better in the 900 training samples case. This indicates that it draws the separation hyper plane differently as compared to SVM.

However we can see that this method is not very instructive in terms of telling us which classifier is better.

2) Spiral Sampled

In order to correct for the above experiment we "sample" the spiral at different equi-spaced points to obtain the training data set. Then we give the points we did not consider in training, for testing. We also have varied the terminating angle of the spiral.

For angle 2.5 pi

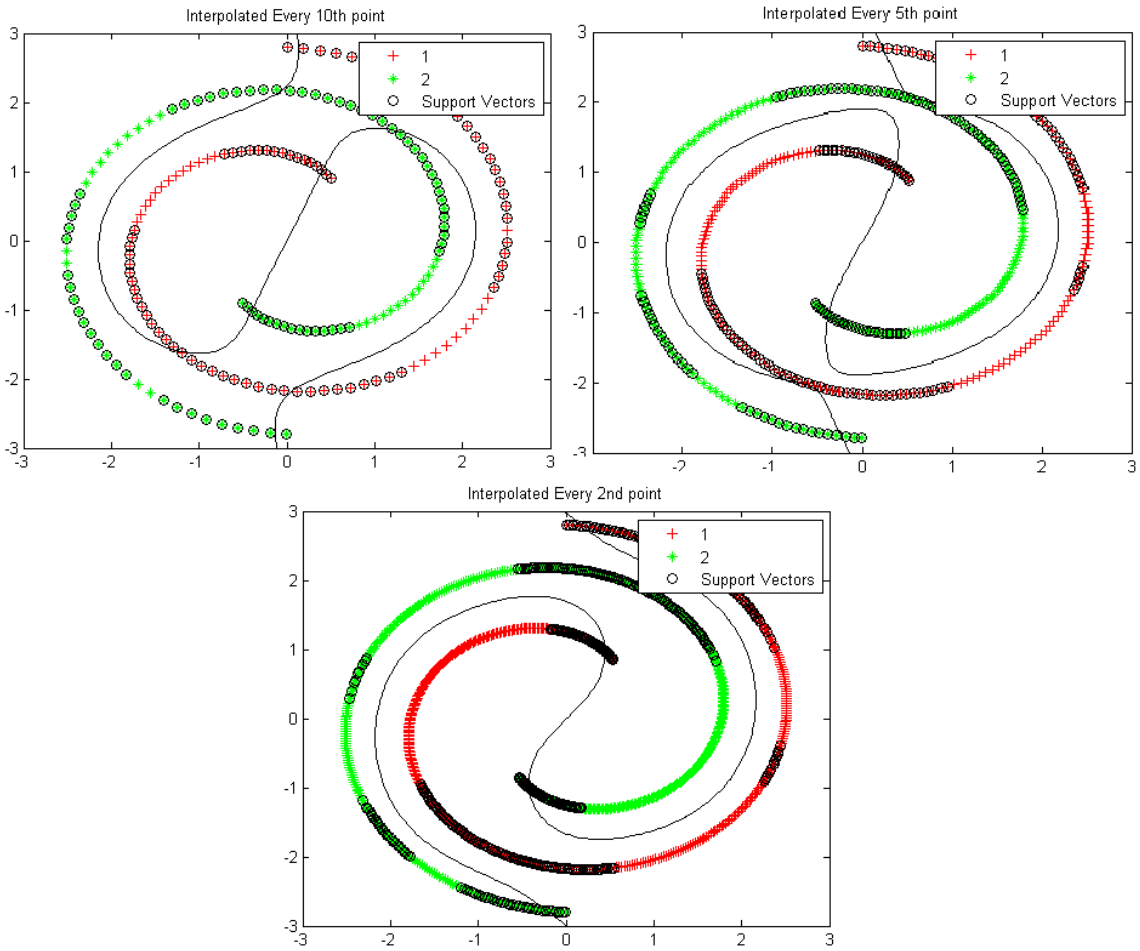


Figure (iv) ***Training data*** *and support vectors for different proportions of training data using* ***radial basis function..***

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

For angle 4*pi

Fraction of samples	1/10	1/20
Accuracy NN 25 hidden neurons	99.9444	99.9474
Accuracy Neural Network 10 hidden neurons	82.77	75.47
Accuracy with 8^{th} degree poly Kernel	99	94.2105
Accuracy with Rbf Kernel	55.7778	56.6316

7^{th} degree polynomial kernel accuracy =98.7778

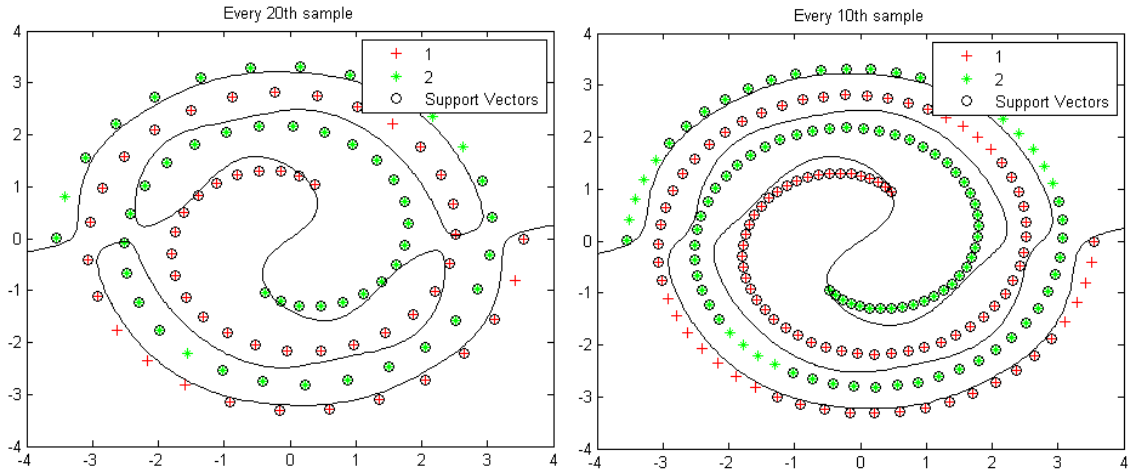


Figure (v) Training data and support vectors for different proportions of training data using 8^{th} order polynomial function using every 20^{th} and 10^{th} sample.

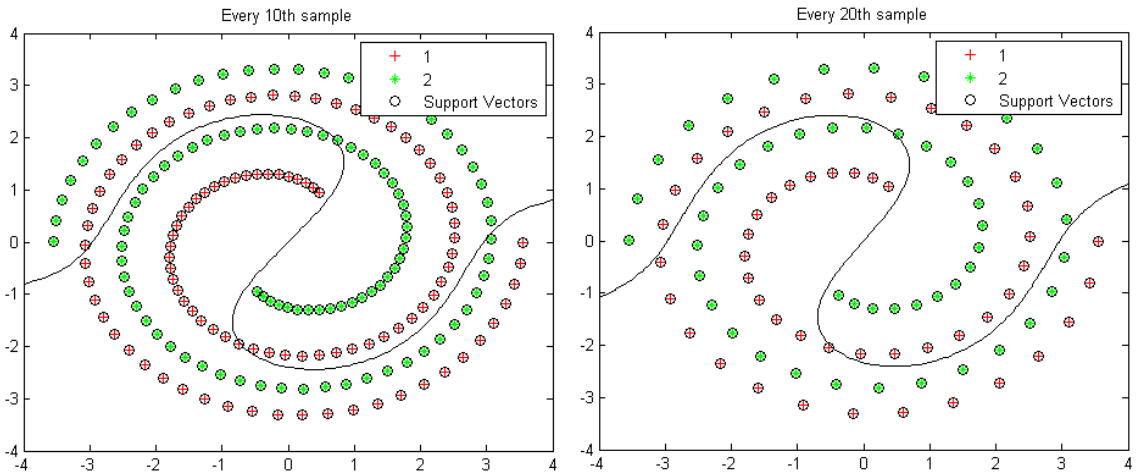


Figure (vi) Training data and support vectors for different proportions of training data using radial basis function using every 10^{th} and 20^{th} sample.

This experiment yields some interesting results. Firstly the SVM with a 8^{th} order polynomial kernel performs very well with a accuracy of 99% with every 10^{th} sample and 94% with every 20^{th} sample respectively. This is a very high accuracy and is similar to those reported in [1] (though they have used a different kernel). The polynomial have several degrees of freedom is accurately able to trace out the spiral. Below 7^{th} order the results are not good. As we increase the angle of the spiral, we need higher and higher degree polynomial. The radial basis function is not able to match the performance of the polynomial kernels for an angle of 4pi. However for 2.5 pi radial basis gives a performance of upto 96.5% which is quite high.
Using similar training procedures for a neural network with 10 hidden neurons we get a poor performance. However when we increased the number of neurons in hidden layer to 25 we get a 99.94 % accuracy as in [1]. This is observed across both cases 2.5 pi and 4 pi.

In conclusion in the spiral problem both ANN and SVMs can give a similar recognition accuracy but ANN's take a large amount of time to achieve the same result.

Conclusion

We have experimented with SVM and Neural Networks for overlapping data and the challenging spiral classification problem. It is difficult to emphatically say that one is better or worse than the other. However we can at least conclude that neural networks take more time to achieve similar results as SVM. The performance of SVM also does not appear to depend on the amount of training data while neural networks have suffer from a problem of overtraining. So when we are willing to forego some amount of accuracy for faster training, SVM appears to be the more attractive option. Moreover kernel's like the polynomial can be made to have high degrees of freedom which can trace even complex surfaces like the spiral. However this is at the cost of computational cost and time complexity. The computational complexity of SVMs does not depend on the dimensionality of the input space unlike in ANN.

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

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 neighbor technique

In this part we wish to experiment with three non parametric pattern classification techniques viz. Parzen Windows, K nearest neighbors and nearest neighbor technique.

The data used for these set of experiments is same as that used for section II of the report. For all parts we present the data followed by a table of results for varying amounts of training and some figures. **It is to be noted that the figures contain all the points (training and test points in blue color). Misclassified samples are indicated by a red x mark**.In the tables R is for Rectangular and G is for Gaussian.

I) Parzen Window Technique

We have developed Matlab code for Parzen Window technique using the rectangular window and Gaussian Window function. We encountered the problem in rectangular window of having no points in the window around the test point. In such case we resolved it by using the knowledge of the priors. We have assumed equal priors. So in case of ties or no points in rectangular window we have generated a random number which gives 1 50% of the time and 2 the other 50% of the time. The output of this random number is assigned as class label to out point under consideration.

In the next section we will present our results for the Parzen Window Technique.

1) **For 2-D well separated data**

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

Experiments for different values of “window” size were conducted for both Gaussian and Rectangular windows. Following are the results

%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

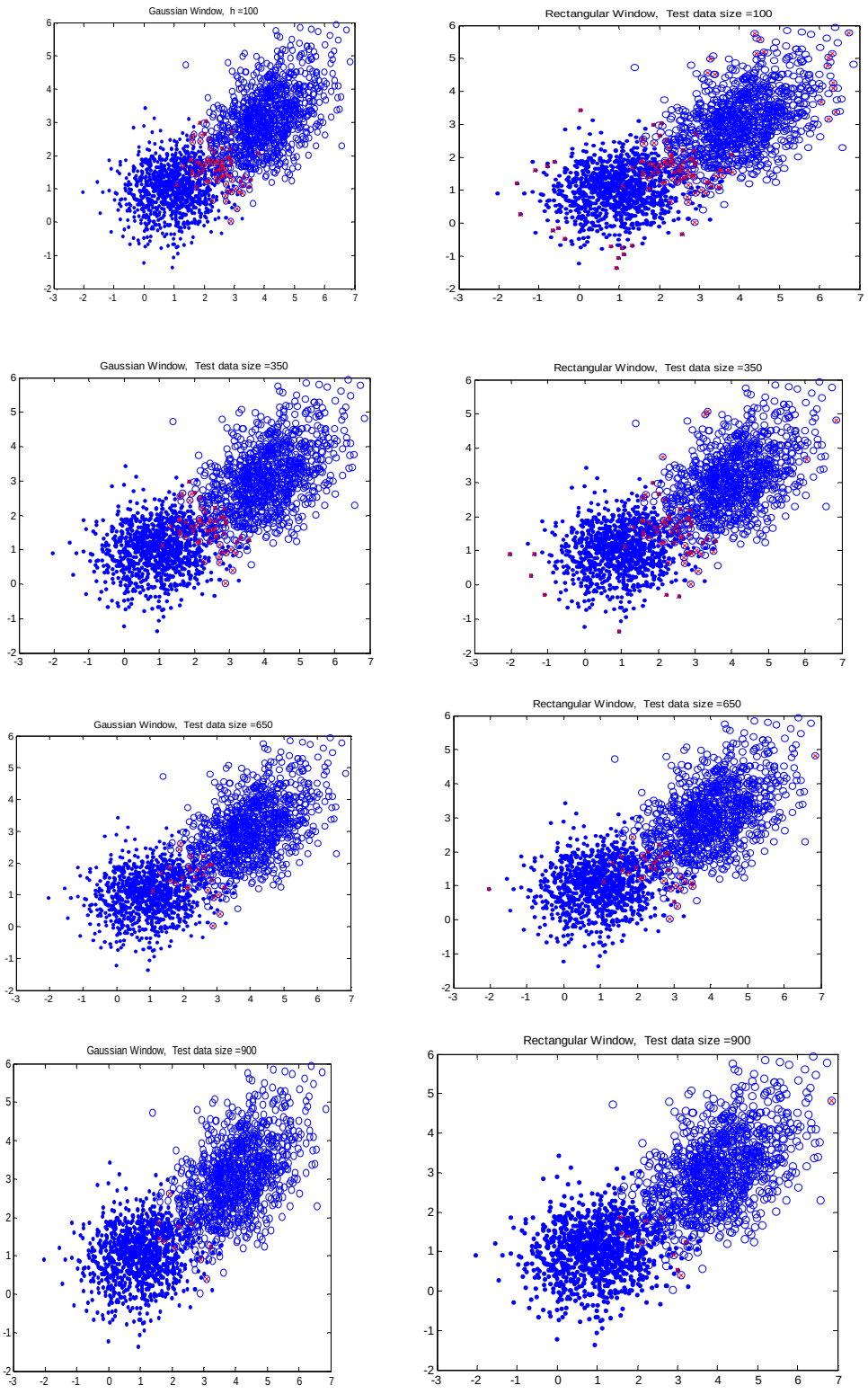


Figure (i) showing the plots of Parzen Window technique using h=0.75.The left half contains results for classification using Gaussian window and the right half contains rectangular window

From the tables and the figures we can observe that performance of Parzen Window technique is better as is expected when we use a Gaussian window. The reason is that it's a smoother estimate of the probability at a given point. 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 decided by the flip of a coin method. 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 impact of the classifier.

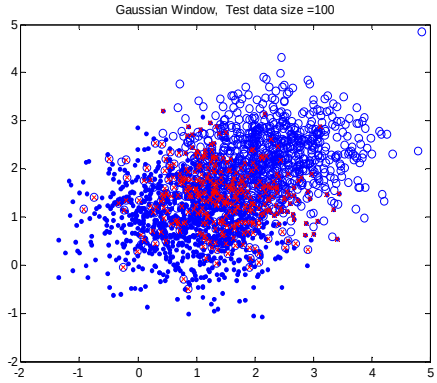
2) Overlapping 2-d Data

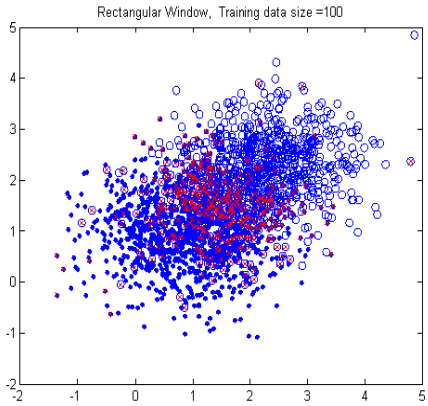
	Class I	Class II
Mean	[1 1]	[2 2]
Covariance	$\begin{pmatrix} 0.8 & 0.07 \\ 0.07 & 0.5 \end{pmatrix}$	$\begin{pmatrix} 0.7 & 0.21 \\ 0.21 & 0.56 \end{pmatrix}$

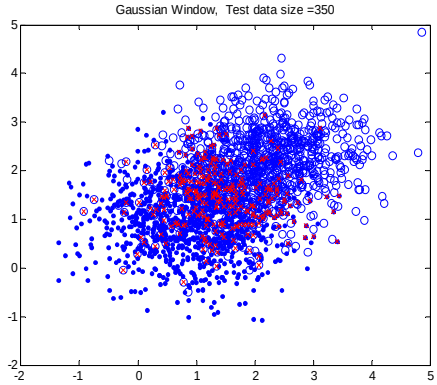
Following is the results for accuracy for rectangular and Gaussian windows of varying size

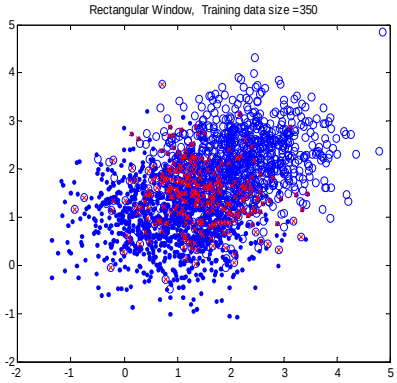
%training / H	10		35		65		90	
	G	R	G	R	G	R	G	R
2	78.56	78.83	79.0	79.07	79.29	78.42	75.0	74.0
1.25	80.17	78.94	78.85	78.31	78.29	78.14	81.5	80.0
0.75	79.72	77.95	78.54	77.54	77.86	77.29	81.0	81.0
0.50	79.76	74.44	78.62	76.77	78.0	76.58	81.0	79.0

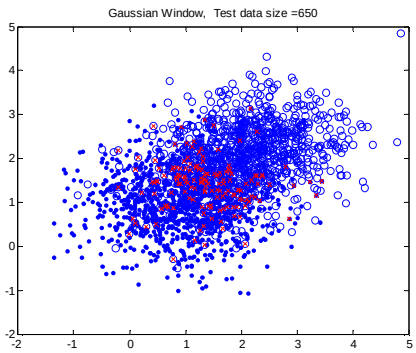
Figure (ii)Following are the visualizations of the above for h=1.25

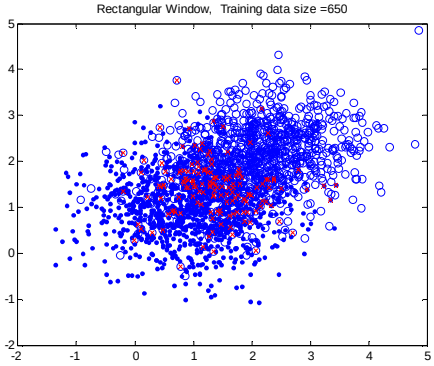


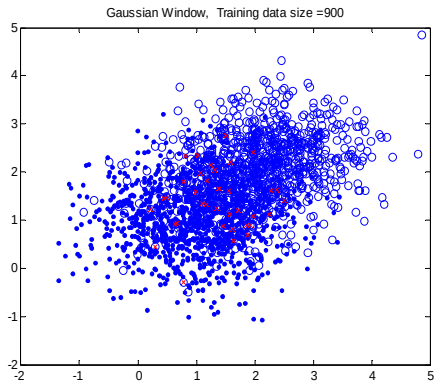


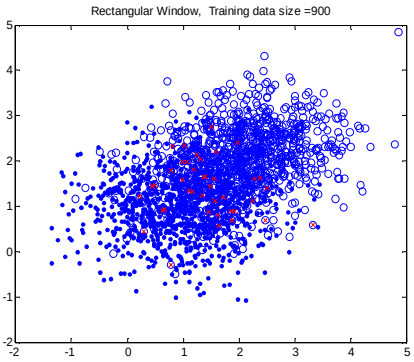












In the above case we have chose overlapping data samples. As expected we start getting incorrect estimates of the density at the test points leading to large error as show in the tabular columns. However even in this case we notice that the Gaussian window function still gives a better estimate as it 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.

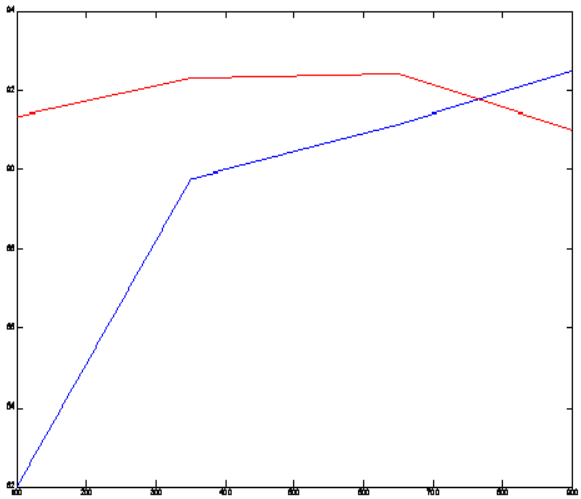
3)3-D overlapping data

In this section we explore the performace of Parzen Window in 3-D space.

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

Following is the results for accuracy for rectangular and Gaussian windows of varying size

%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



Figure(iii) Error as a function of number of training points per class for Gaussian window(red) and rectangular window(blue)

3-D overlapping data II

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

The following is the result of varying training data and window size

%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

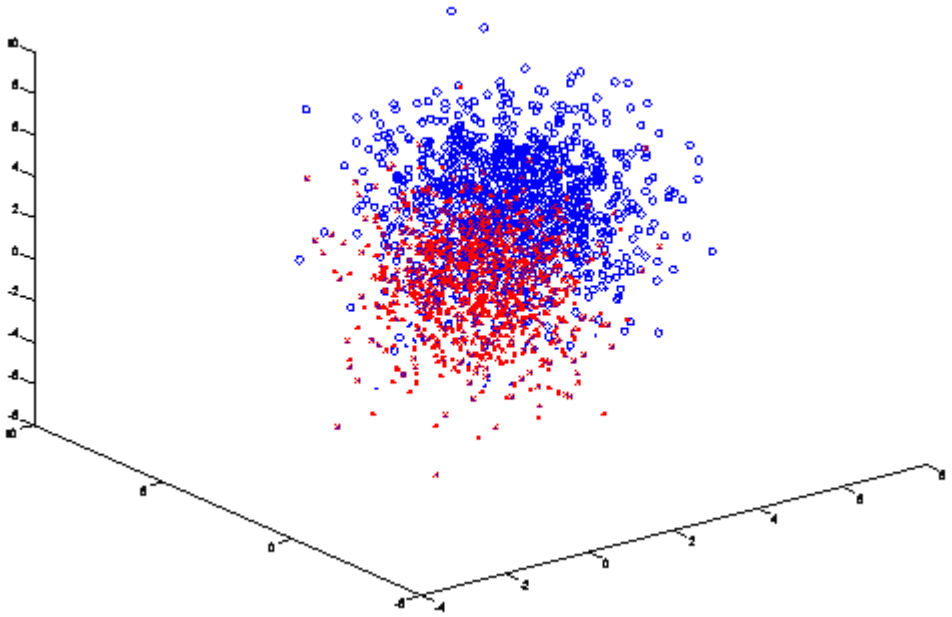
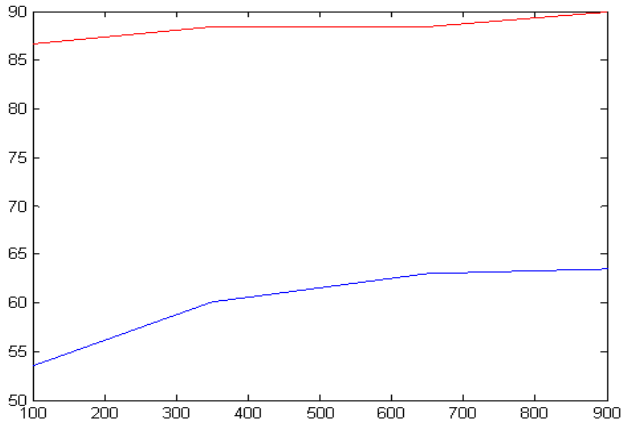


Figure (iv) Sample plot for h=0.50 and 10% training data



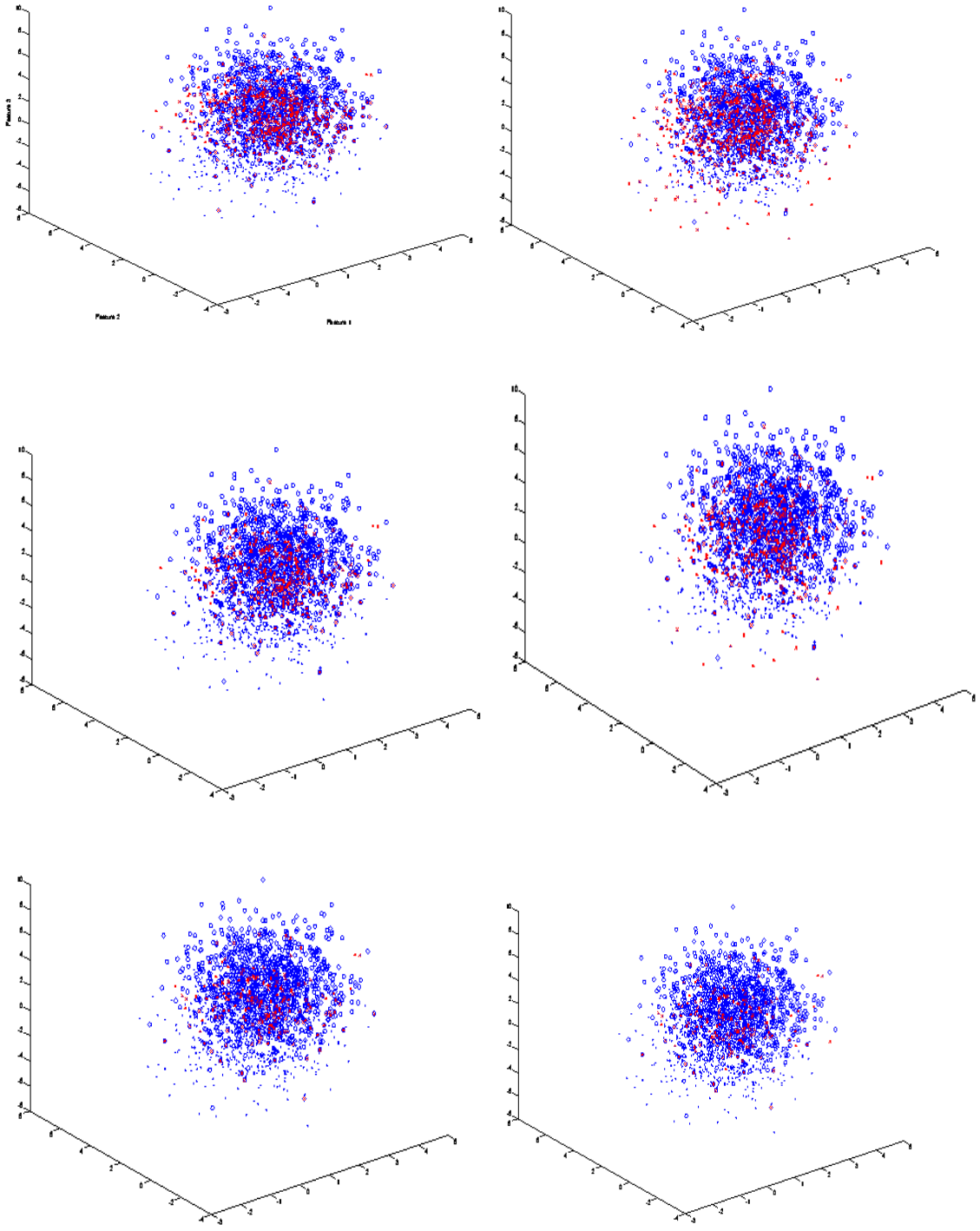
Fig(v) The error plot for h=0.50

3-D more overlapping with shifted mean

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

The results for the above are as follows:

%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


Feature 3
Feature 2
Feature 1

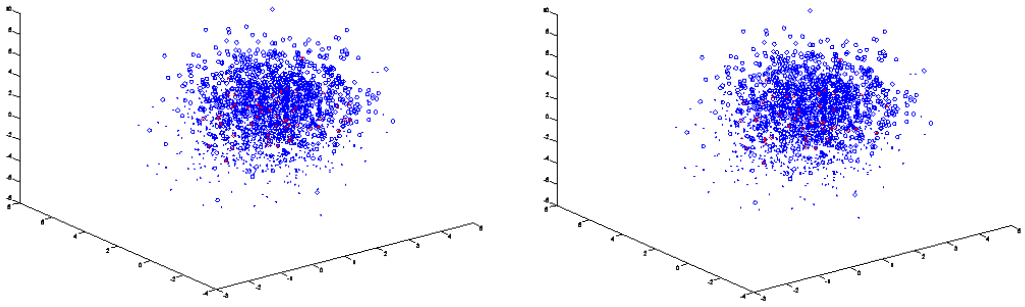


Figure (vi) showing the plots for h=1 for Gaussian and rectangular window for 900,650,350 and 100 training samples respectively.

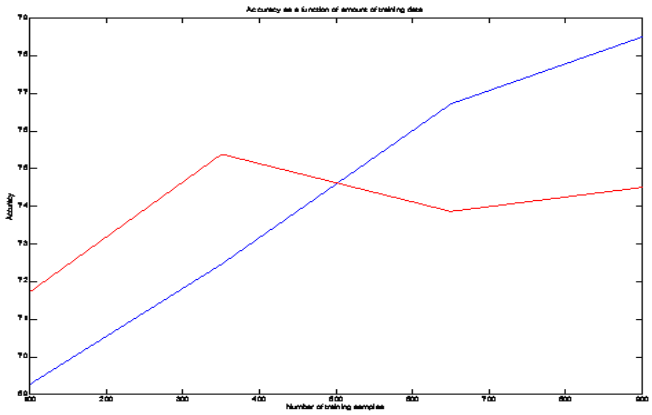


Figure (vii) showing the accuracy as a function of training samples

In the case of 3-D also Parzen window technique yields similar results as in 2-D case. When there is overlap of data there is significant decrease in performance of the classifier. The Gaussian window performs better in almost all cases and there is not much variation as a function of % of training especially when the data is separable which in quite intuitive. Differently distributed data perform better with different window sizes. The size of window is particularly important when we use rectangular window than in the case of Gaussian window. Hence we need to pick the right window size to optimize the performance of our classifier based on Parzen Window. We also observe that most of the error in the rectangular technique is because of outlier points which have no neighbors and hence are classified based on the priors(=0.5 in our case).

II) K nearest Neighbors and Nearest Neighbor Technique

We have operated on exactly the same data as in Parzen Window technique. The organization of this part is as follows. For each experiment we present the data and the results of classification using different values of K and by varying the amount of data points available in advance (training). The programs are implemented with the help of Matlab and the code is attached at the end of the section.

1) **For 2-D well separated data**

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

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

K \ %training	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 also conducted experiments on the same data using **Manhattan distance**

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

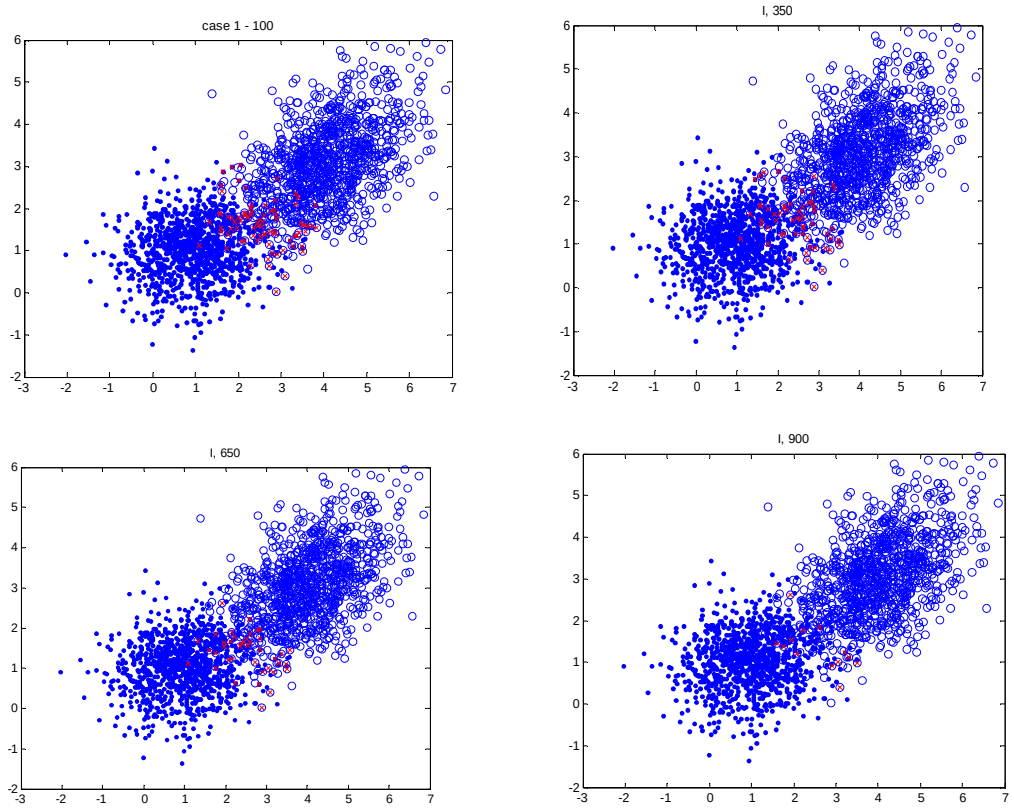


Figure (viii) the plots of Nearest neighbour with eucledian distance metric

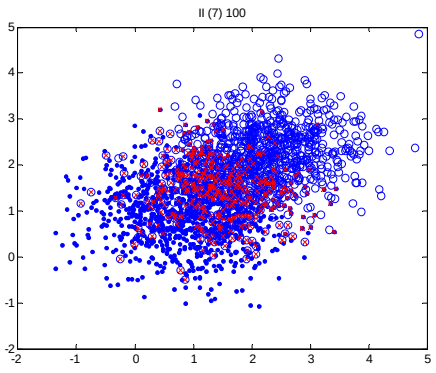
In the case of K nearest neighbors we observe that classification accuracy varies as K is changed. We observe that K=1 does not perform as well as K=3, 5 or 7. This indicates that K=1 is not as robust as higher values of K. We have chosen odd values of K in order to prevent ties from occurring when we check which class samples are in a majority around our test points. 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.

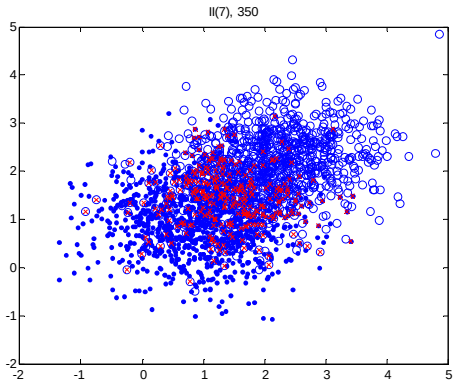
Data Set II

	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

The results for the above data for K=7 are as follows:





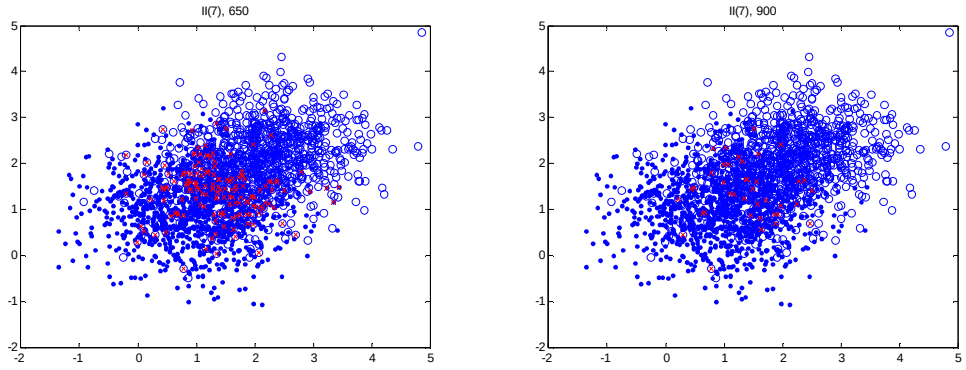


Figure (ix) showing K=7 classification for overlapping data

In this case also we observe that K=1 is not as good as higher values for K. K=7 in this case yields us better results than other values of K. In this case the data is highly overlapped and hence we cannot be drawing definite conclusions about which classifier is better. But we can say that it is better to use a value of K greater than one to have a better accuracy of the class of our test samples.

For 3-D data I

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

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

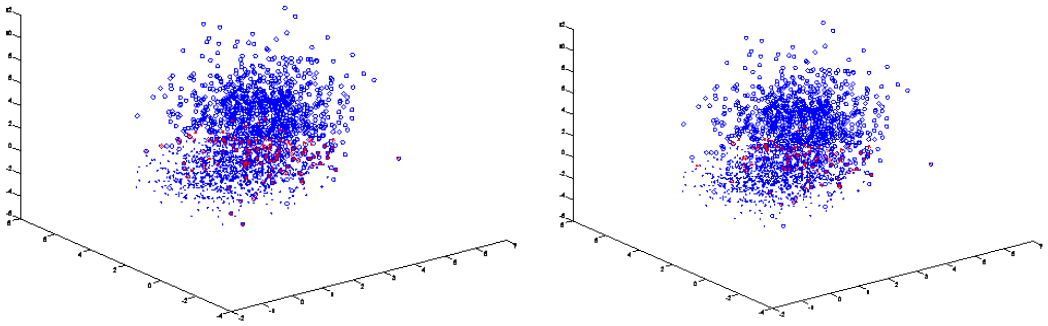


Figure (x) showing K=7 classification for overlapping data with 100 and 350 training samples

For 3-D data II

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

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

3-D data set III

	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

In this case the percentage accuracy of classification is very poor as the data is highly overlapped. The K=9 case gives a comparatively better performance than other cases once again reinforcing the fact that checking the nearest neighbor alone can lead to incorrect classification.

Conclusions

In this section we have performed experiments for classifying overlapping data in 2-D and 3-D for varying amounts of overlap using the Parzen Window and K nearest neighbor technique. We find that for Parzen Window using a Gaussian window function results in better performance than rectangular window. K nearest neighbors out performs nearest neighbor technique and is more robust. We also observe that KNN does not suffer from the problem that Parzen window(rectangular kernel) has, that points of a class which are very far from its means are not classified based on priors in KNN. So these points are well classified by KNN. Comparing Parzen window 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 this is because we experimented with mainly overlapping data in our quest 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.

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
```