ECE 662 Homework II

April 15, 2008

1 Fisher Linear Discriminant

In that qustion, we are going to implement Fisher's Linear Discrimainant using several kinds of data to compare the two approach. The idea is to project the data onto one dimensional unit vector **w** and use a bias point $\mathrm{w}_0$to discriminate the classes [3, 4]. At the first step, I have generated two classes coming from two Gaussian distributions in 2-D. There are 100 feature vectors in the first class and 150 in the second class. The mean and the covariance of the feature vectors coming from first class are

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 1 \\ 5 \end{bmatrix} \\ \Sigma_1 &= \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

and the second class has

$$\begin{aligned} \mu_2 &= \begin{bmatrix} 5 \\ -3 \end{bmatrix} \\ \Sigma_2 &= \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

then I have calculated the sample means of the classes using $m_k = \frac{1}{N_{1k}} \sum_{i \epsilon C_k} x_i$ and the results are:

$$m_1 = \begin{bmatrix} 0.7974 \\ 4.9891 \end{bmatrix}$$

and the second class has

$$m_2 = \begin{bmatrix} 5.1280 \\ -3.0156 \end{bmatrix}$$

Our goal is to maximize between-class variance while minimizing within-class variance. For that purpose, Fisher criterion is given by

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

where s_1 and s_2 is given by

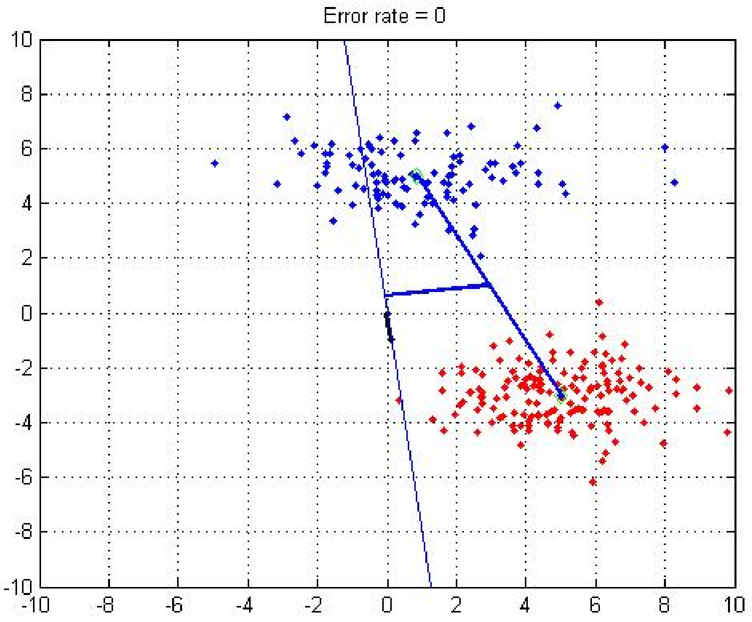
$$s_k^2 = \sum_{i \epsilon C_k} (y_i - m_k)^2$$

maximization of J($\mathbf{w}$) gives

$$\mathbf{w} \propto \mathbf{S_W^{-1}}(\mathbf{m_2} - \mathbf{m_1})$$

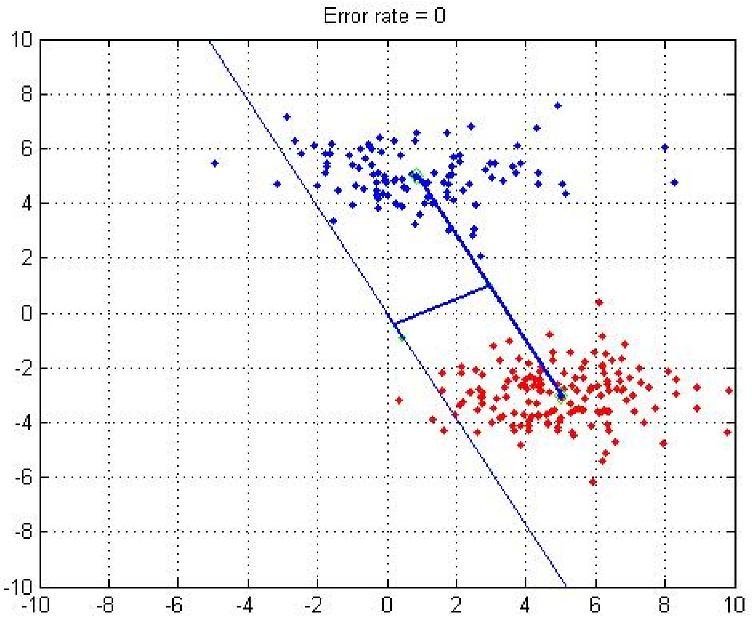
where $\mathbf{S_W} = \sum_{\mathbf{i \epsilon C_1}} (\mathbf{x_i} - \mathbf{m_1})(\mathbf{x_i} - \mathbf{m_1})^\mathbf{T} + \sum_{\mathbf{i \epsilon C_2}} (\mathbf{x_i} - \mathbf{m_2})(\mathbf{x_i} - \mathbf{m_2})^\mathbf{T}$.

If the data is linearly separable then projection of the data onto $\mathbf{w}$ using $\mathbf{w^T x}$gives the best separation according to the Fisher's Criterion. Here is the implementation results(the Matlab code is attached to the end of the report):



As can be seen from the figure, all the data is projected onto$\mathbf{w}$ which is the little black vector determining the reference line for projection. Separation line connects the mid-point of the vector determined by $(\mathbf{m_2} - \mathbf{m_1})$ and its projection onto the direction of $\mathbf{w}$. Because the data is separable, Fisher's discriminant function separates the data without error. The error rate is %0.

Next step is to take $\mathbf{w} \propto (\mathbf{m_2} - \mathbf{m_1})$ which means taking the within-class variance matrix as identity matrix: $\mathbf{S_w} = \mathbf{I_2}$. Using the same data with that approach gives the following results:



Error rate again %0. So we can conclude that in some cases there is no difference between these two approach. Let's think about the situation and try to decide when there is no difference between these two approach. Obviously, if we take the covariance matrices as isotropic matrices then $\mathbf{S_W}$ will be proportional to unit matrix, implying that the second approach is a special condition of the first approach which is Fisher's Criterion. Then, we should find some situations in which the second approach fails.

Next step is letting the two classes of the data come close to each other so that there are some misclassified points. We need to change the mean and maybe the covariance matrices to see what happens. Here is the results with two new classes coming from two Gaussian distribution:

Mean and the covariance matrix for the first class is

$$\mu_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$$

and the second class has

$$\mu_2 = \begin{bmatrix} 5 \\ -1 \end{bmatrix}$$

$$\Sigma_2 = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

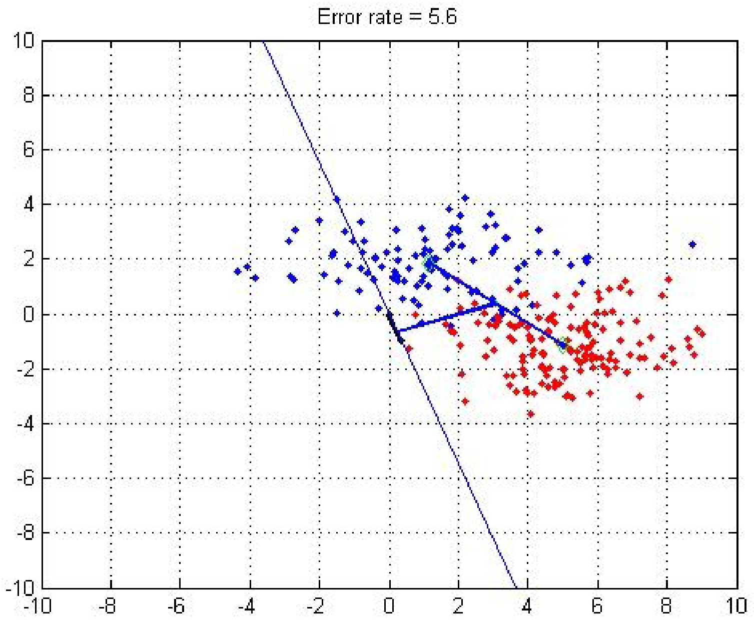
the calculated sample means of the classes are:

$$m_1 = \begin{bmatrix} 1.1308 \\ 1.8712 \end{bmatrix}$$

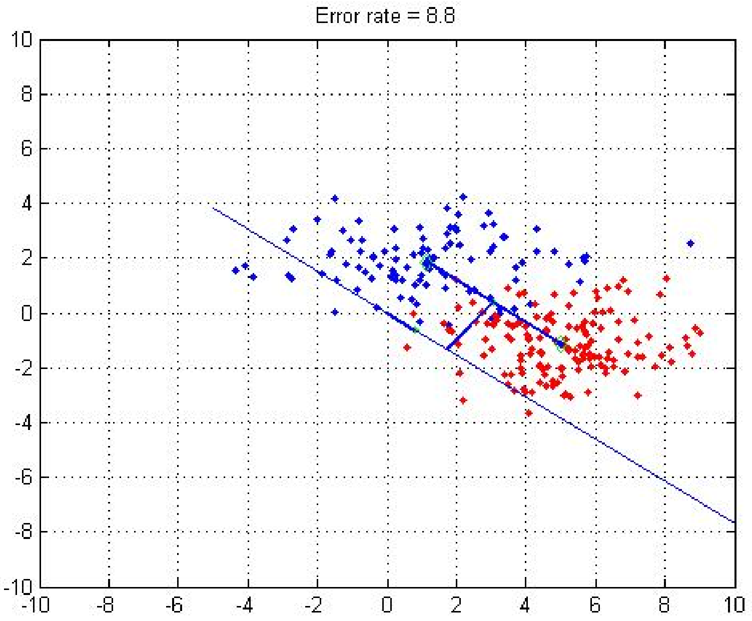
and the second class has

$$m_2 = \begin{bmatrix} 4.9989 \\ -1.1143 \end{bmatrix}$$

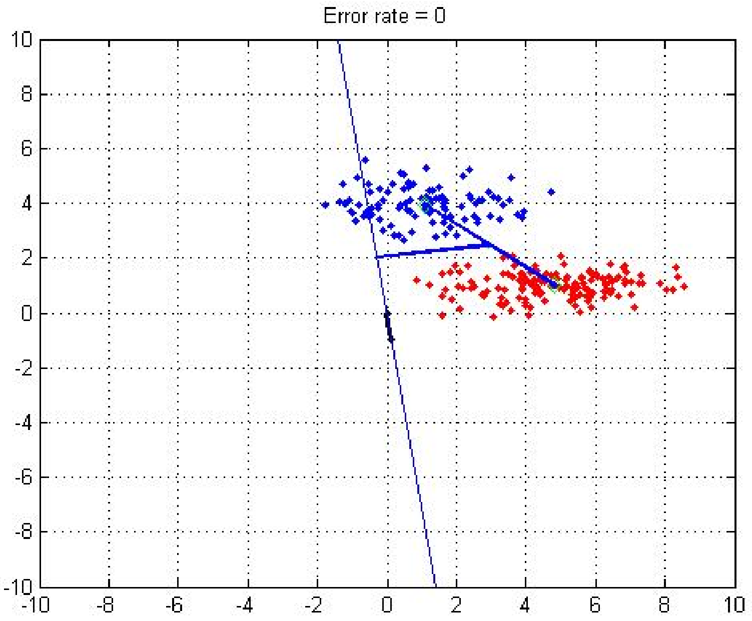
First the Fisher's Method:



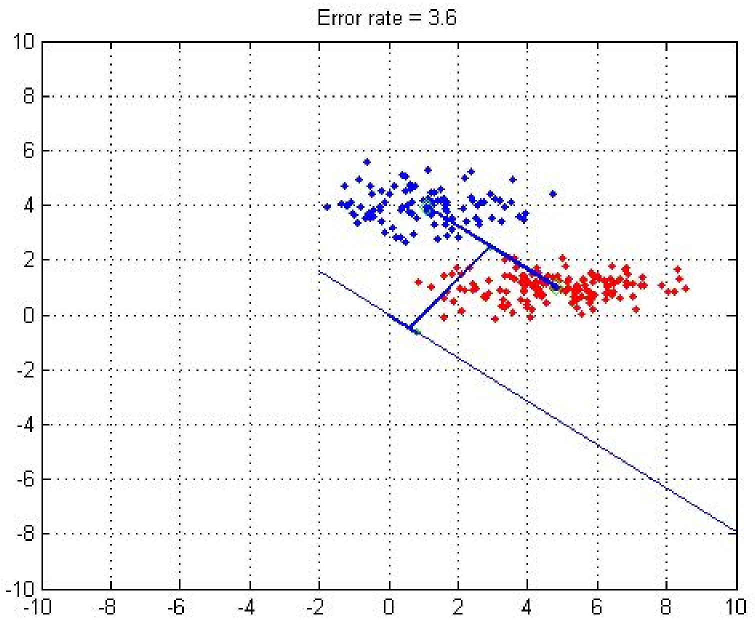
We see thatthe error rate is now %5.6 with Fisher's criterion. Now let's look at second approach:



In that case we have a different $\mathbf{w}$As can be seen the error rate increases if we take $\mathbf{S_W} = I_2$. From this example it is clearly understood that if we use Gaussian data Fisher's criterion catches the separation line better than second approach. Here is another experiment:



The data is linearly separable and Fisher's criterion separates perfectly. Here is the second approach:



Data is linearly separable but second approach fails to separate it. Error rate is %3.6. Next step is to taking outliers into account. What if there are some outliers.For that purpose, I have generated 10 outliers from the first class. I have 210 feature vectors from first class (together with outliers) and 200 feature vectors from second class. Here are the mean and covariance matrix for class 1(without outliers)

$$\mu_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$
$$\Sigma_1 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$

and the second class has

$$\mu_2 = \begin{bmatrix} 5 \\ -1 \end{bmatrix}$$
$$\Sigma_2 = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

and the outliers from class 1:

$$\mu_{10} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$
$$\Sigma_{10} = \begin{bmatrix} 1 & 0 \\ 0 & 0.4 \end{bmatrix}$$

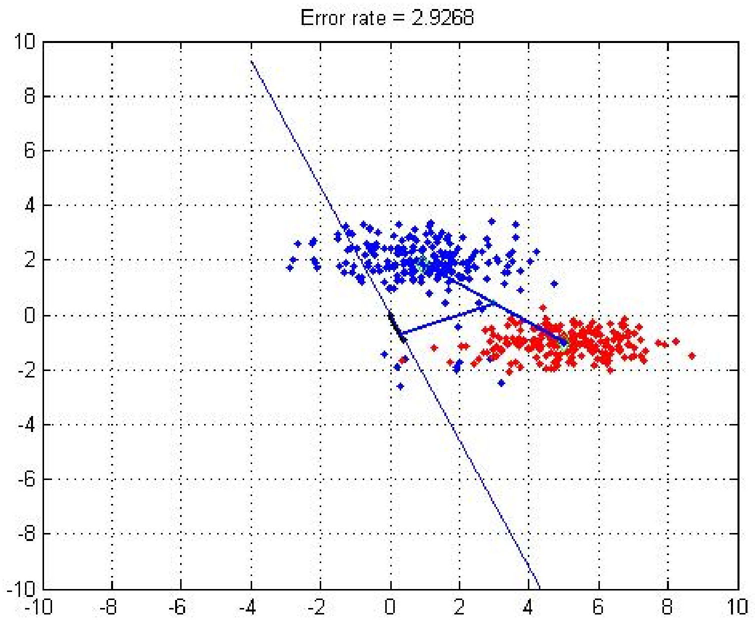
sample means of the classes are:

$$m_1 = \begin{bmatrix} 0.9298 \\ 1.8504 \end{bmatrix}$$

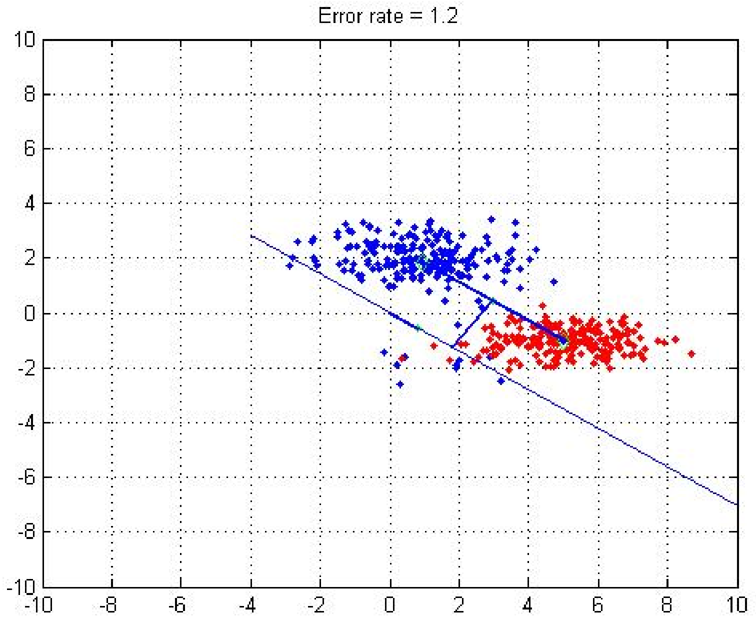
and the second class has

$$m_2 = \begin{bmatrix} 5.0062 \\ -1.0142 \end{bmatrix}$$

Here are the results. Fisher's:

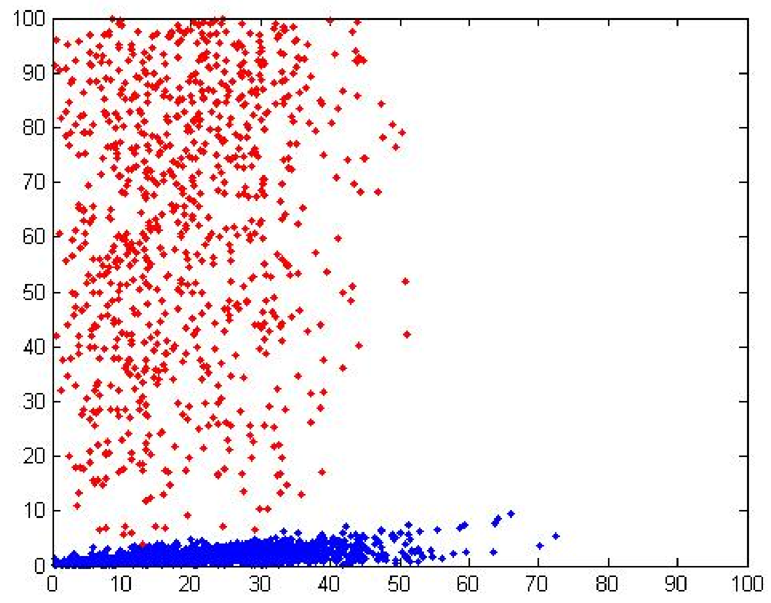


And the second approach:

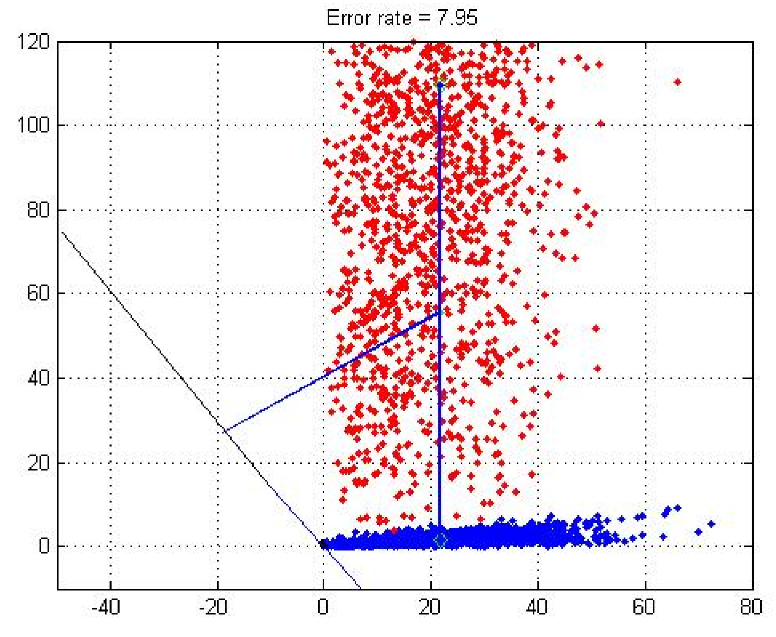


As can be seen, taking $\mathbf{S_w} = \mathbf{I_2}$generates slightly better results with the error rate %1.2 vs 2.9268. From that results, we can conclude that if the data is linearly separable, that is if we can draw a line without misclassifying any data points then Fisher's linear discriminant separates the data perfectly, whereas the second approach fails to separate. However, if the data is not linearly separable, that is we cannot draw a line without misclassifying some data points, second approach MAY generate slightly better results depending on the characteristics of the data.

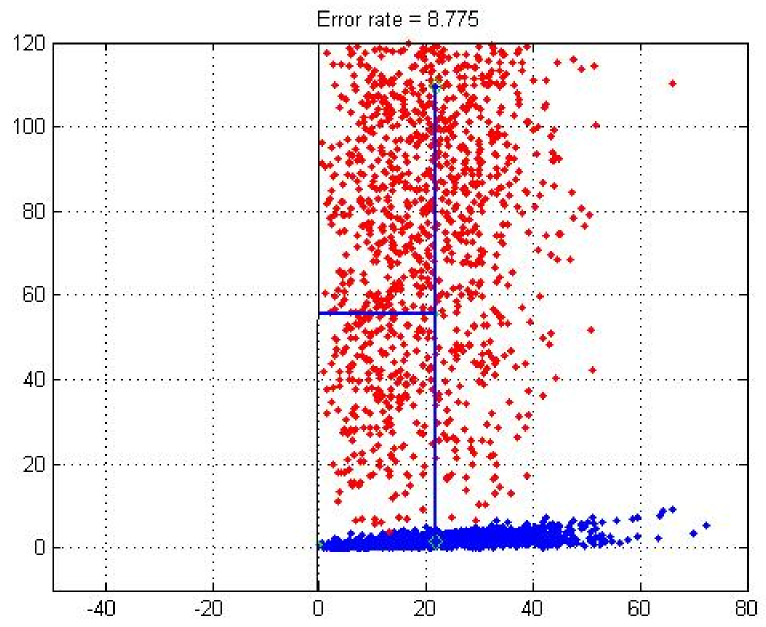
For the next step, I wanted to try Fisher's linear discriminant on some real data. I have used some portions of the real dataset called "Multiple Features" from the website: http://archive.ics.uci.edu/ml/datasets.htm Because this data set is multivariate, I have only used 2 specific dimensions of it, which are virtually linearly separable and can be plotted. First let's see how the data look like:



There are 2000 feature vectors from each classes. First I have applied Fisher's Linear Discriminant to it. Here are the results:



In order to see the projection clearly, I needed to scale the axis. The vertical line connects the sample means of the two classes. Thin, black line is the projection line. All the data is projected onto it. The Threshold point to separate the classes is chosen to be the projection of the midpoint onto $\mathbf{w}$(as can be seen on the figure). Here is the result if we take $\mathbf{S_W} = \mathbf{I_n}$:



All the data is projected onto the black line. Again Fisher's criterion performs better.

2 Support Vector Machines & Artificial Neural Networks

Let's start with Support Vector Machine(SVM). I have used Matlab's Toolbox functions to implement SVM. I have chosen the Glass Identification Database [2]to implement the two methods . The dataset has 214 instances, 9 attributes, and 7 classes. The classes are labeled as

– 1 building_windows_float_processed
– 2 building_windows_non_float_processed
– 3 vehicle_windows_float_processed
– 4 vehicle_windows_non_float_processed (none in this database)
– 5 containers
– 6 tableware
– 7 headlamps

The features to separate the data is given as:

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon

7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron

Because, we want to work on a two-class problem, I have labeled the dataset as being headlamps and non-headlamps. I have written a Matlab script to run Matlab's built-in functions. The script is attached to the end of the report.

2.1 Explanations of used built-in functions:

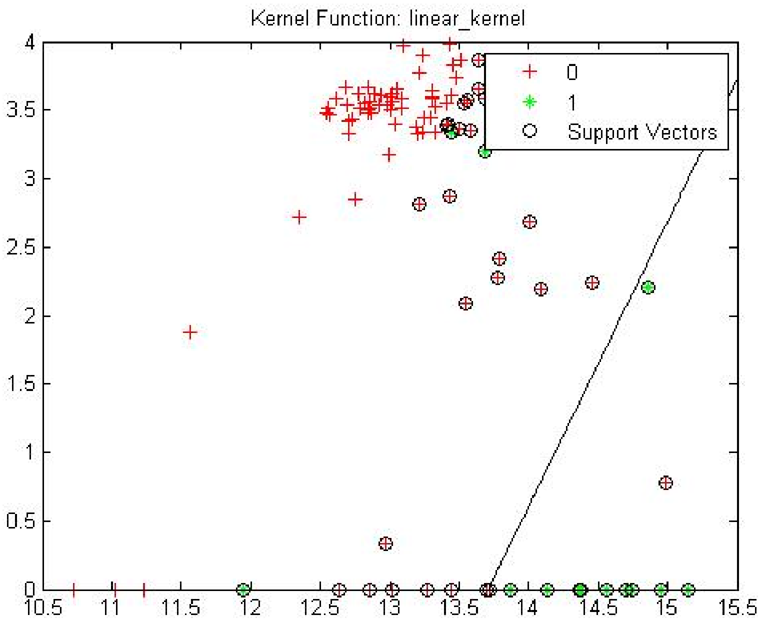
crossvalind: Used to split the data as train and test samples.

svmtrain: Used to train the SVM. It takes the training data, corresponding labels, type of kernel function to be used and some additional parameters as input. The output has the support vectors and the reqired parameters of SVM.

svmclassify: the input is the output of the function 'svmtrain', the test data, and some additional parameters regularizing plotting and so on.

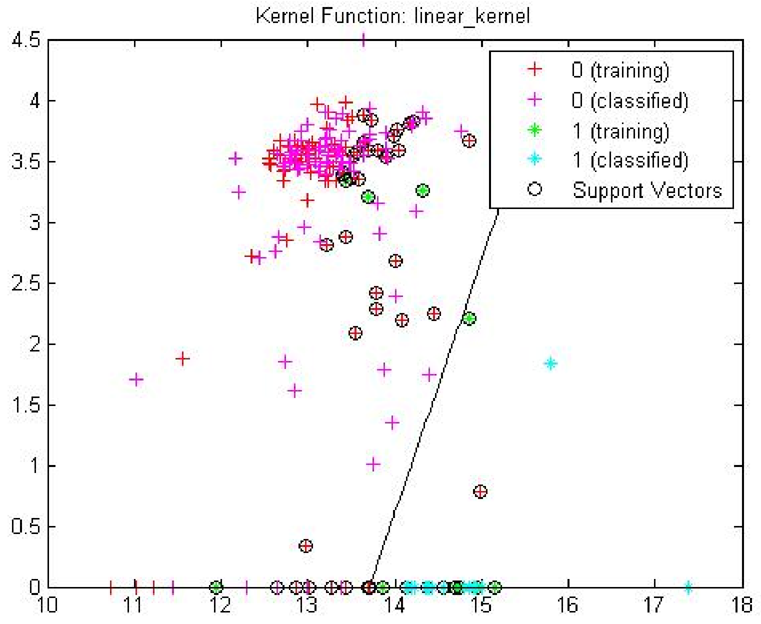
I have started to test the algorithm by using third(Na) and fourth(Mg) features of the data to separate the data as being headlamps and non-headlamps using linear kernel function. Below are the plots and the classification rate.

Training results:

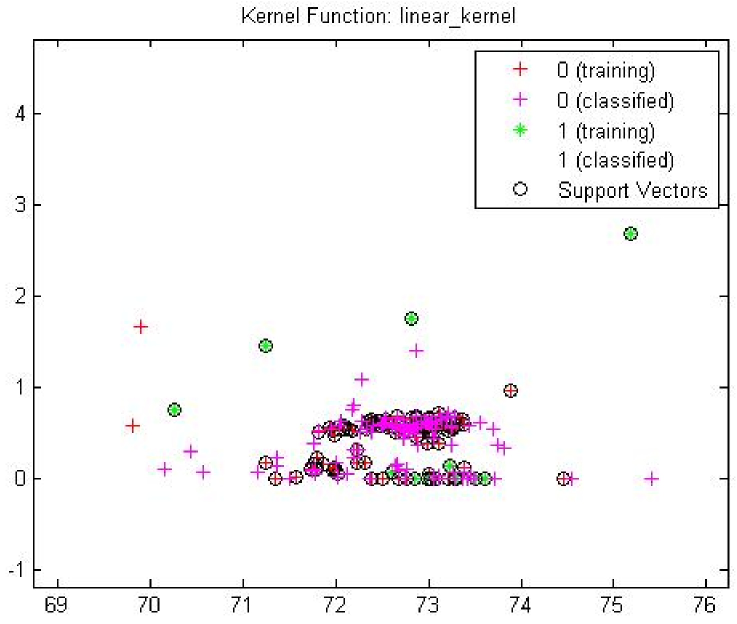


As can be seen on the figure, red-plus samples are non-headlamps samples, and green-star samples are headlamps samples. The support vectors are circled samples.

Test results:



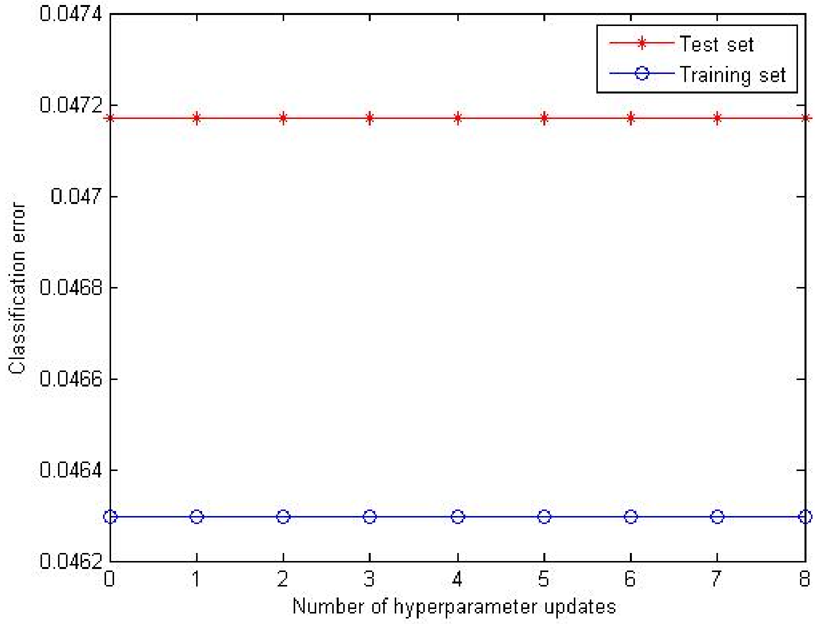
It seems that SVM draws the best separation line. The classification rate is 0.9528 which is pretty good, because we are using only two features out of 9 to separate the data. Is that because of the power of the features to separate the data? Let's try some other features. We can take Silicon and Potasium to separate the data.



These features obviously worse than the previous features. The classification rate is 0.8679. Before using more features, let us do the classification with the same data by using Artificial Neural Networks(ANN) to compare SVM and ANN.

For ANN, I have used the ANN code for binary classification [1]. The code is needed a little change, so that the Glass dataset can be used as input. The altered code is also attached to the report. The code simply takes the train and the test data, and inplements ANN algorithm. The m-file my_ANN is the altered script to run the related files of the algorithm. The main file is nc_main which trains the Network. Hidden unit number is chosen to be 10. The algorithm starts with randomly initializing the weights, then it creates outputs. Finally, backpropagation is done using BFGS gradient method to optimize the weights.

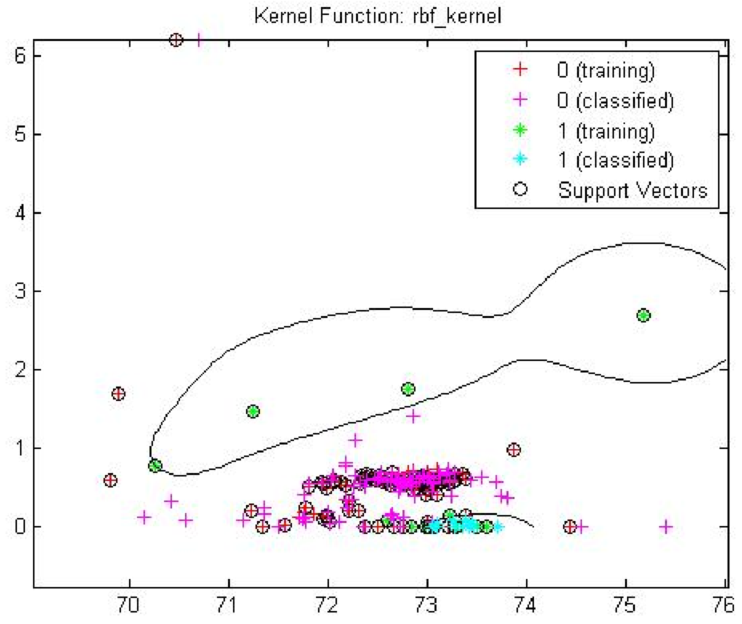
Firstly, Mg and Na is used to classify the data as we have done with the SVM. Interestingly, the classification rate is happen to be same: 0.9528.



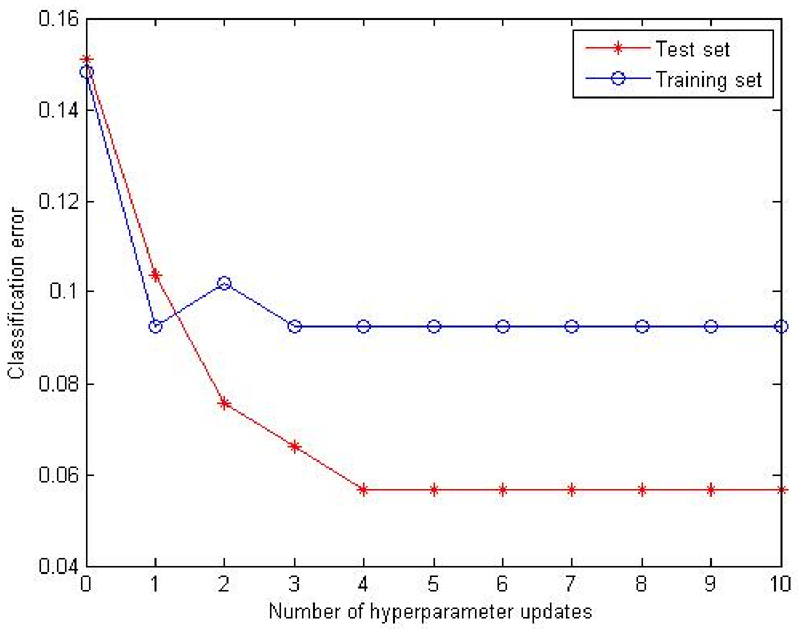
Obviously, selected feature vectors is so important that two methods give the same error rate. Let's try silicon and potasium to calssify the data, as we have done with SVM. Classification rate is 0.9340. Again, these features are worse than the first ones. However, Classification rate is better than the SVM's rate which was 0.8679 with these same features.

Obviously, difference is too much, We might think that SVM should have performed better than that. The key point in SVM is to choose the most suitable kernel function to the characteristics of the dataset. So, let's try famous Gausian Kernel to see if we can improve the classification rate 0.8679.

Indeed yes, if we use Radial Basis Function to classify the data using silicon and potasium with SVM, we outperform the linear kernel function. Here is the results:



SVM draws really interesting curves to separate the data, and classification rate is 0.9528 which is much better than 0.8679 and also better than ANN classification rate which was 0.9340. Immediate question is 'can we improve ANN performance by playing with the number of hidden units?'. The answer is 'perhaps we can.' Instead of using 10 hidden units, I have tried 5 hidden units. Here are the results:



The classification rate is 0.9434, improvement! 5 hidden units seem to be optimal for the data we use. Here is table showing different number of hidden units and their performance:

Number of Hidden units	2	3	4	5	6	11
Classification Rate	0.9245	0.9434	0.9434	0.9434	0.9340	0.9340

That is all we can do. The best classification rate is 0.9434, which is slightly worse than SVM's rate with Gaussian Kernel.

Here is a table showing SVM classification rates with different kernel functions(Features:Silicon, Potasium):

Kernel Function	Linear	Quadratic	Polynomial	Gaussian
Classification Rate	0.8679	0.8491	0.9623	0.9528

More interestingly, if we use polynomial kernel function, classification rate is even better than Gaussian Kernel's rate, proving the importance of choosing the right kernel function for a spesific dataset.

2.2 Using more features

Next step is to scrutinize the effect of using more than 2 features. Let's start with SVM. Using first three features which are refractive index, sodium, and magnesium, the classification rates are given below:

Kernel Function	Linear	Quadratic	Polynomial	Gaussian
Classification Rate	0.9340	0.9434	0.9434	0.9717

In 3-D Polinomial Kernel could not beat Gaussian kernel.

What about ANN? Here is a table showing the experiments using first three features to classify the data:

Number of Hidden units	2	3	4	5	6	20
Classification Rate	0.9340	0.9528	0.9528	0.9434	0.9434	0.9434

The best rate is 0.9528 which is worse than SVM's rate with Gaussian Kernel.

Finally, what if we use every feature in the dataset to separate the data?Below are the results for SVM using 9 features.

Kernel Function	Linear	Quadratic	Polynomial	Gaussian
Classification Rate	0.9717	0.9717	0.9717	0.9717

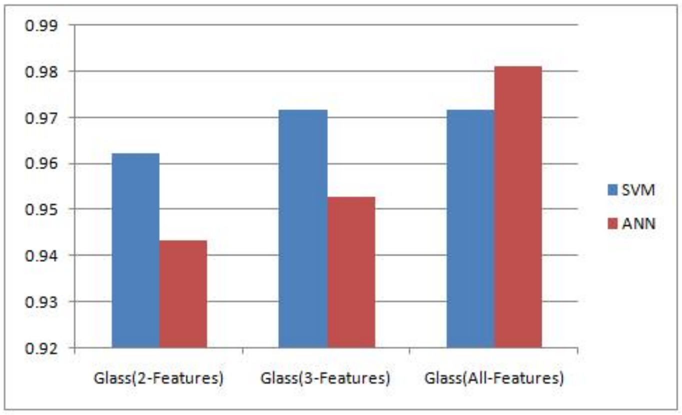
Obviously, 0.9717 is the best we can do with SVM. Here are the results for ANN using 9 features:

Number of Hidden units	2	3	4	5	6	20
Classification Rate	0.9811	0.9811	0.9811	0.9811	0.9811	0.9811

Regardless of the number of hidden units I used the classification rate is 0.9811 which is better than SVM's rate.

2.3 Discussions

This is really exciting. ANN outperforms SVM if we use all features. it had worse performance for less features. Obviously, it is not easy to say that one method is better than the other. Based on the chracteristics of the dataset, SVM and ANN generates slightly different results. For the dataset I have used, ANN has a better result with all features. Moreover, we need to remember that we have only used 4 different kernel functions for SVM. One can come up with a better kernel function and outperform ANN's classification rate. Moreover, if we use another dataset, results might also be different. In either case, both of the methods give a classification rate above 0.96. Here is a chart to show the best classification rates achieved by each method for my datasets.



3 Parzen Windows & Nearest Neighbors

3.1 Parzen Windows

For Parzen Window mathod, I have written my own code. It is attached to the report. The algorithm simply generates a window and for each test point, it counts the training points falling into that window.In d-dimensions, it is a hypercube. The algorithm generates a distance matrix which computes the distance of training samples to test samples such that

$$distance\,matrix(i,j) = distance\,between\,i-th\,test\,sample\,and\,j-th\,training\,sample$$

The class which has the most training points falling into the window determines the label of that test point. Obviously, my code copies each point to memory which can be infeasible for large datasets. We can have some additional processes to improve performance of the algorithm but it is out of the scope of this homework.

I have used the same dataset that I used for SVM and ANN However, in order to test my algorithm, I havefirstly used "fisheriris" data set which is linearly separable in 2-D. I captured the maximum classification rate with a window size of 1. Here is a table showing the results for "iris" data.

Window Length	0.2	0.3	0.35	0.5	1	1.5
Classification Rate	0.92	0.9733	0.9867	0.9867	1	0.8133

Because the function "crossvalind" randomly chooses the test and the training samples, I had slightly different classification rates for the same window length, when I run it several times.

Next, I have used the "glass" data which is the same dataset I used for SVM and ANN. Firsly, I have only used first 2 features which are refractive index and Sodium amount(Na) to separate the data as being headlamps and non-headlamps. Here is a table showing my experiments.

Window Length	0.2	0.3	0.5	1	1	1.5
Classification Rate	0.8868	0.9057	0.9057	0.9245	1	0.8874

Then I tried first 3 features to separate the data again. Results:

Window Length	0.2	0.5	1	1.5	2	3
Classification Rate	0.9340	0.9528	0.9528	0.9528	0.9434	0.8585

Finally, I used all 9 features to separate the data. Results:

Window Length	0.2	0.3	0.5	1	2	3	3.1
Classification Rate	0.8774	0.9057	0.9528	0.9717	0.9717	0.9906	0.9811

Obviously, adding more features increases the classification rate. On the other hand, window length is extremely important. I had a classification rate of 0.9906, when I used all of the features with a window length of 3.

3.2 Nearest Neighbors Methods

I have also written a matlab code for K-Nearest Neighbor. In my code, I have counted nearest k training points to each of the test points and the class which has the majority of the labels in that k points determines the label of the test point. The algorithm uses distance matrix again. As I did with Parzen Windows Method, I used fisheriris data to test my algorithm to see whether it can separate linearly separable data without error.

Here are the results:

k	1	5	10	15	20	25	50
Classification Rate	1	0.9867	1	1	1	0.9867	0.6667

Note that if k equals 1 then KNN becomes Nearest Neigbor Method.

I have run the code several times for several k values. When k equals 1, the algorithm has always had a classification rate of 1 regardless of the training and test samples. However, when I increased k, I had a classification rate which is less than 1 for different test and training samples. For example, for k=5, I had classification rate equal to 1, if training and test samples are exchanged slightly. Therefore, we can say that having a good training set of data is extremely important to have a good classifier.

Next, I have tried first 2 features of the "glass" data to classify the data as being headlamps and non-headlamps again. Here are the results:

k	1	5	10	15	20	25	30
Classification Rate	0.8868	0.8962	0.9057	0.8962	0.9151	0.8679	0.8679

I have reached the best classification rate with k=20 which is nearly $\frac{1}{5}$ of the number of training samples. Let's add one more feature:

k	1	5	10	15	20	25	30
Classification Rate	0.9340	0.9434	0.9434	0.9528	0.9057	0.8585	0.8679

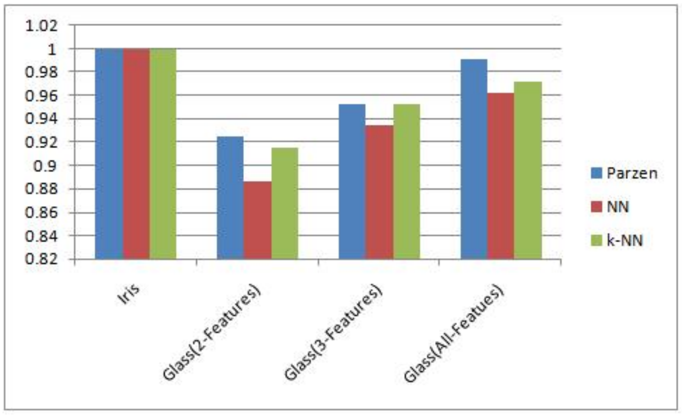
It increased the rates a little as expected. Finally, let's stick in all features:

k	1	5	10	15	20	25	30
Classification Rate	0.9623	0.9623	0.9716	0.9717	0.8774	0.8679	0.8679

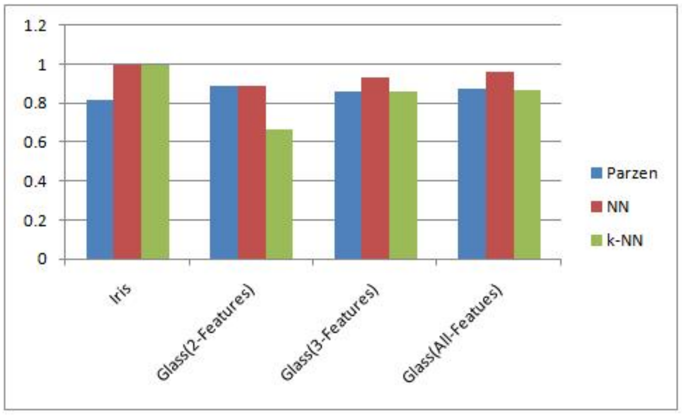
As can be seen, I had a classification rate above 96% with all features used. The value of k is obviously very important.

3.3 Discussions

From the experiments, it seems that parzen windows generated slightly better results then the nearest neighbor and k-Nearest Neighbor did for the "Glass" dataset. Here is a chart showing best classification rates achieved by the three methods with different datasets.



We might also want to see the worst classification rates generated by the 3 approach.



Obviosly, we can make the algorithms perform much worse than that by changing window size or the k value to extreme points. For Parzen windows if we take the window length long enough in order the window to include every sample point we have then the classification rate for Glass data with all features used is 0.8679. On the other hand, if we take the value of k so large as to include all the point we have in the dataset then classification rate with the same dataset with all features used is 0.8679 which is same as the Parzen window rate. We can conclude that in extreme situations, Parzen windows and k-Nearest Neighbor Methods are equivalent.

References

[1] Ann:dtu toolbox, 2002.

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] C.M. Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[4] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* Wiley-Interscience, 2000.

MATLAB SCRIPTS

1)

```
%---------------------------------------------------------------------------
% ECE 662
% Homework 2
% Question 1
% Fisher's Linear Discriminant
%---------------------------------------------------------------------------
clear
num_sample_1 = 100;
num_sample_2 = 150;
class1 = zeros(num_sample_1,1);%we have 100 data points
class2 = zeros(num_sample_2,1);%we have 150 data points
within_class1 = [0 0;0 0];
within_class2 = [0 0;0 0];
mu1 = [1 2];
Sigma1 = [2 0; 0 0.4];
mu2 = [5 -1];
Sigma2 = [3 0; 0 0.2];
temp1=[0;0];
temp2=[0;0];

%Generate sample points
F1 = mvnrnd(mu1,Sigma1,num_sample_1);
F2 = mvnrnd(mu2,Sigma2,num_sample_2);
%Save sample points to use later
csvwrite('class_1.csv',F1);
csvwrite('class_2.csv',F2);
projection_1 = zeros(num_sample_1,1);
projection_2 = zeros(num_sample_2,1);
plot(F1(:,1),F1(:,2), 'b.');
hold on;
plot(F2(:,1),F2(:,2), 'r.');

class1_mean = mean(F1)
class2_mean = mean(F2)
midpoint = ((class1_mean+class2_mean)/2)
plot(class1_mean(1),class1_mean(2),'gd')
plot(class2_mean(1),class2_mean(2),'gd')

%Find S_W and w_0
for i=1:num_sample_1
    within_class1 =within_class1 + (F1(i,:)'-class1_mean')*(F1(i,:)'-
class1_mean')';
end
for i=1:num_sample_2
    within_class2 =within_class2 + (F2(i,:)'-class2_mean')*(F2(i,:)'-
class2_mean')';
end
S_W=within_class1+within_class2

w_0= inv(S_W)*(class2_mean'-class1_mean');
w_0=w_0/norm(w_0)
```

```
%m=(1/(num_sample_1+num_sample_2))*(num_sample_1*class1_mean'+num_sample_2*cl
ass2_mean')
bias=-w_0'*midpoint'

%Project The data
for i=1:num_sample_1
    projection_1(i)=w_0'*F1(i,:)';
    if(projection_1(i)>=-bias)
        misclass_1=misclass_1+1;
    end
end
for i=1:num_sample_2
    projection_2(i)=w_0'*F2(i,:)';
    if(projection_2(i)<-bias)
        misclass_2=misclass_2+1;
    end
end
misclass_1
misclass_2
%Find the classification ratio and plot results
ratio=100*(misclass_1+misclass_2)/(num_sample_1+num_sample_2)
%Plot----------------------------------------------------------------------
plot(w_0(1),w_0(2), 'g.');
plot(midpoint(1),midpoint(2), 'g.');
temp1(1) = class1_mean(1);
temp1(2) = class2_mean(1);
temp2(1) = class1_mean(2);
temp2(2) = class2_mean(2);
plot(temp1,temp2,'b.-','LineWidth',2);
temp1(1) = w_0(1);
temp1(2) = 0;
temp2(1) = w_0(2);
temp2(2) = 0;
slope1(1)=w_0(2)/w_0(1);
slope1(2)=0;
%plot W_0
plot(temp1,temp2,'k.-','LineWidth',2.5);
refline(slope1);
mid_proj=(w_0'*midpoint')*w_0;
temp1(1) = mid_proj(1);
temp1(2) = midpoint(1);
temp2(1) = mid_proj(2);
temp2(2) = midpoint(2);
line(temp1,temp2,'LineWidth',2);
grid on;
title(['Error rate = ' num2str(ratio) ]);
axis([-10 10  -10  10]);
hold off;
%--------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% ECE 662
% Homework 2
% Question 1
% S_W equals Identity
%--------------------------------------------------------------------------
clear
num_sample_1 = 100;
num_sample_2 = 150;
class1 = zeros(num_sample_1,1);%we have 100 data points
class2 = zeros(num_sample_2,1);%we have 150 data points
within_class1 = [0 0;0 0];
within_class2 = [0 0;0 0];

temp1=[0;0];
temp2=[0;0];

%Load saved data
F1 = load('class_1.csv');
F2 = load('class_2.csv');
[num_sample_1 coll1]= size(F1)
[num_sample_2 coll2]= size(F2)
projection_1 = zeros(num_sample_1,1);
projection_2 = zeros(num_sample_2,1);
%plot data
plot(F1(:,1),F1(:,2), 'b.');
hold on;
plot(F2(:,1),F2(:,2), 'r.');

class1_mean = mean(F1)
class2_mean = mean(F2)
midpoint = ((class1_mean+class2_mean)/2)
plot(class1_mean(1),class1_mean(2),'gd')
plot(class2_mean(1),class2_mean(2),'gd')

%S_W is identity
S_W=[1 0;0 1]
w_0= inv(S_W)*(class2_mean'-class1_mean');
w_0=w_0/norm(w_0)
%m=(1/(num_sample_1+num_sample_2))*(num_sample_1*class1_mean'+num_sample_2*cl
ass2_mean')
bias=-w_0'*midpoint'
misclass_1=0;
misclass_2=0;
%Project the data
for i=1:num_sample_1
    projection_1(i)=w_0'*F1(i,:)';
    if(projection_1(i)>=-bias)
        misclass_1=misclass_1+1;
    end
end
for i=1:num_sample_2
    projection_2(i)=w_0'*F2(i,:)';
    if(projection_2(i)<-bias)
        misclass_2=misclass_2+1;
    end
```

```
end
%Find the classification ratio and plot
misclass_1
misclass_2
ratio=100*(misclass_1+misclass_2)/(num_sample_1+num_sample_2)

plot(w_0(1),w_0(2), 'g.');
plot(midpoint(1),midpoint(2), 'g.');
temp1(1) = class1_mean(1);
temp1(2) = class2_mean(1);
temp2(1) = class1_mean(2);
temp2(2) = class2_mean(2);
plot(temp1,temp2,'b.-','LineWidth',2);
temp1(1) = w_0(1);
temp1(2) = 0;
temp2(1) = w_0(2);
temp2(2) = 0;
slope1(1)=w_0(2)/w_0(1);
slope1(2)=0;

line(temp1,temp2,'LineWidth',2);
refline(slope1);

mid_proj=(w_0'*midpoint')*w_0
temp1(1) = mid_proj(1);
temp1(2) = midpoint(1);
temp2(1) = mid_proj(2);
temp2(2) = midpoint(2);
line(temp1,temp2,'LineWidth',2);
title(['Error rate = ' num2str(ratio) ]);
grid on;
axis([-50 80  -10 120]);
hold off;
%-------------------------------------------------------------------------
```

```matlab
%---------------------------------------------------------------------------
% ECE 662
% Homework 2
% Question 1
% Add outliers to the data
%---------------------------------------------------------------------------
clear
num_sample_1 = 200;
num_sample_11 = 10;%outliers for class 1
num_sample_2 = 200;
class1 = zeros(num_sample_1,1);%we have 200 data points
outliers1 = zeros(num_sample_11,1);%outliers
class2 = zeros(num_sample_2,1);%we have 200 data points
within_class1 = [0 0;0 0];
within_class2 = [0 0;0 0];
mu1 = [1 2];
Sigma1 = [2 0; 0 0.4];
mu11 = [1 -2];
Sigma11 = [1 0; 0 0.4];
mu2 = [5 -1];
Sigma2 = [2 0; 0 0.2];
temp1=[0;0];
temp2=[0;0];

%Generate the samples
F1 = mvnrnd(mu1,Sigma1,num_sample_1);
F11= mvnrnd(mu11,Sigma11,num_sample_11);
F2 = mvnrnd(mu2,Sigma2,num_sample_2);
F1 = [F1;F11];
num_sample_1=num_sample_1+num_sample_11;
%save the samples to use later
csvwrite('class_1.csv',F1);
csvwrite('class_2.csv',F2);
projection_1 = zeros(num_sample_1,1);
projection_2 = zeros(num_sample_2,1);
%Plot the data
plot(F1(:,1),F1(:,2), 'b.');
hold on;
plot(F2(:,1),F2(:,2), 'r.');




class1_mean = mean(F1)
class2_mean = mean(F2)
midpoint = ((class1_mean+class2_mean)/2)
plot(class1_mean(1),class1_mean(2),'gd')
plot(class2_mean(1),class2_mean(2),'gd')

%Find S_W and w_0
for i=1:num_sample_1
    within_class1 =within_class1 + (F1(i,:)'-class1_mean')*(F1(i,:)'-
class1_mean')';
end
for i=1:num_sample_2
    within_class2 =within_class2 + (F2(i,:)'-class2_mean')*(F2(i,:)'-
class2_mean')';
```

```
end
S_W=within_class1+within_class2

w_0= inv(S_W)*(class2_mean'-class1_mean');
%normalize w_0
w_0=w_0/norm(w_0)
%m=(1/(num_sample_1+num_sample_2))*(num_sample_1*class1_mean'+num_sample_2*cl
ass2_mean')
bias=-w_0'*midpoint'
misclass_1=0;
misclass_2=0;
%Project the data
for i=1:num_sample_1
    projection_1(i)=w_0'*F1(i,:)';
    if(projection_1(i)>=-bias)
        misclass_1=misclass_1+1;
    end
end
for i=1:num_sample_2
    projection_2(i)=w_0'*F2(i,:)';
    if(projection_2(i)<-bias)
        misclass_2=misclass_2+1;
    end
end
misclass_1
misclass_2
%Find the classification ratio and plot results
ratio=100*(misclass_1+misclass_2)/(num_sample_1+num_sample_2)

plot(w_0(1),w_0(2), 'g.');
plot(midpoint(1),midpoint(2), 'g.');
temp1(1) = class1_mean(1);
temp1(2) = class2_mean(1);
temp2(1) = class1_mean(2);
temp2(2) = class2_mean(2);
plot(temp1,temp2,'b.-','LineWidth',2);
temp1(1) = w_0(1);
temp1(2) = 0;
temp2(1) = w_0(2);
temp2(2) = 0;
slope1(1)=w_0(2)/w_0(1);
slope1(2)=0;
%plot W_0
plot(temp1,temp2,'k.-','LineWidth',2.5);
refline(slope1);
mid_proj=(w_0'*midpoint')*w_0;
temp1(1) = mid_proj(1);
temp1(2) = midpoint(1);
temp2(1) = mid_proj(2);
temp2(2) = midpoint(2);
line(temp1,temp2,'LineWidth',2);
grid on;
title(['Error rate = ' num2str(ratio) ]);
axis([-10 10  -10  10]);
hold off;
%-------------------------------------------------------------------------
```

```matlab
% ---------------------------------------------------------------------------
% ECE 662
% Homework 2
% Question 1
% Repeat process with real data
%----------------------------------------------------------------------------
clear
within_class1 = [0 0;0 0];
within_class2 = [0 0;0 0];
temp1=[0;0];
temp2=[0;0];
%Load the data and take wanted dimensions
D = load ('mfeat.csv');
F1 = [D(:,3) D(:,2)];
F2 = [D(:,3) D(:,5)];
[num_sample_1 coll1]= size(F1)
[num_sample_2 coll2]= size(F2)

%Save the data to use later
csvwrite('class_1.csv',F1);
csvwrite('class_2.csv',F2);
projection_1 = zeros(num_sample_1,1);
projection_2 = zeros(num_sample_2,1);
plot(F1(:,1),F1(:,2), 'b.');
hold on;
plot(F2(:,1),F2(:,2), 'r.');

class1_mean = mean(F1)
class2_mean = mean(F2)
midpoint = ((class1_mean+class2_mean)/2)
plot(class1_mean(1),class1_mean(2),'gd')
plot(class2_mean(1),class2_mean(2),'gd')

%Find S_W and w_0
for i=1:num_sample_1
    within_class1 =within_class1 + (F1(i,:)'-class1_mean')*(F1(i,:)'-
class1_mean')';
end
for i=1:num_sample_2
    within_class2 =within_class2 + (F2(i,:)'-class2_mean')*(F2(i,:)'-
class2_mean')';
end
S_W=within_class1+within_class2

w_0= inv(S_W)*(class2_mean'-class1_mean');
w_0=w_0/norm(w_0)
m=(1/(num_sample_1+num_sample_2))*(num_sample_1*class1_mean'+num_sample_2*cla
ss2_mean')
bias=-w_0'*m
misclass_1=0;
misclass_2=0;
%Project The data
for i=1:num_sample_1
    projection_1(i)=w_0'*F1(i,:)';
    if(projection_1(i)>=-bias)
        misclass_1=misclass_1+1;
    end
end
for i=1:num_sample_2
    projection_2(i)=w_0'*F2(i,:)';
    if(projection_2(i)<-bias)
        misclass_2=misclass_2+1;
    end
end
misclass_1
misclass_2
ratio=100*(misclass_1+misclass_2)/(num_sample_1+num_sample_2)

plot(w_0(1),w_0(2), 'g.');
plot(midpoint(1),midpoint(2), 'g.');
temp1(1) = class1_mean(1);
temp1(2) = class2_mean(1);
temp2(1) = class1_mean(2);
temp2(2) = class2_mean(2);
plot(temp1,temp2,'b.-','LineWidth',2);
temp1(1) = w_0(1);
temp1(2) = 0;
temp2(1) = w_0(2);
temp2(2) = 0;
slope1(1)=w_0(2)/w_0(1);
slope1(2)=0;
%plot W_0
plot(temp1,temp2,'k.-','LineWidth',2.5);
refline(slope1);
mid_proj=(w_0'*midpoint')*w_0;
temp1(1) = mid_proj(1);
temp1(2) = midpoint(1);
temp2(1) = mid_proj(2);
temp2(2) = midpoint(2);
line(temp1,temp2,'LineWidth',2);
grid on;
title(['Error rate = ' num2str(ratio) ]);
axis([-50 80  -10 120]);
hold off;
%--------------------------------------------------------------------------
```

Q.2) SVM&ANN

```
% -----------------------------------------------------------------------
% ECE 662
% Homework 2
% Question 2
% Support Vector Machines
% -----------------------------------------------------------------------
clear;
%load the data and prepare for SVM
S=load ('glass.data');
%We have 9 features
data = [S(:,2),S(:,3),S(:,4),S(:,5),S(:,6),S(:,7),S(:,8),S(:,9),S(:,10)];
%class 7 is headlamps
groups = ismember(S(:,11),7);
K=load('svm.csv');
train=logical(K(:,1));
test=logical(K(:,2));
%Do crossvalidation only once to use the same datasets as train and test
%[train, test] = crossvalind('holdOut',groups)
%csvwrite('svm.csv',[train,test]);

%start training
cp = classperf(groups);
svmStruct =
svmtrain(data(train,:),groups(train),'showplot',true,'Kernel_Function',
'polynomial');
title(sprintf('Kernel Function: %s',   func2str(svmStruct.KernelFunction)),
'interpreter','none');
%classify test data
figure (2)
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
classperf(cp,classes,test);
cp.CorrectRate
% -----------------------------------------------------------------------
% ------------------------------------------------------------------------
% ECE 662
% Homework 2
% Question 2
% Artificial Neural Network(Binary classification)
% This script runs related .m files which can be found in references
% ------------------------------------------------------------------------
clear;
%load the data and prepare it for ANN
S=load ('glass.data');
K=load('svm.csv');
train=logical(K(:,1))
test=logical(K(:,2));
%We have 9 features
data = [S(:,2),S(:,3),S(:,4),S(:,5),S(:,6),S(:,7),S(:,8),S(:,9),S(:,10)];
%class 7 is headlamps
groups = ismember(S(:,11),7)

x=data(train,:);
t=groups(train);
x_test=data(test,:);
t_test=groups(test);
% Set the number of hidden units
Nh = 7;

% Train the network
disp('Network training, this will not take long...')
results = nc_main(x,t,x_test,t_test,Nh);

% Plot the error
figure(1)
x_axis = 0:length(results.Etest)-1;
plot(x_axis,results.Etest,'r*-',x_axis,results.Etrain,'bo-')
xlabel('Number of hyperparameter updates')
ylabel('Average cross-entropy error')
legend('Test set','Training set')

% Plot the classification error
figure(2)
1-results.Ctest
plot(x_axis,results.Ctest,'r*-',x_axis,results.Ctrain,'bo-')
xlabel('Number of hyperparameter updates')
ylabel('Classification error')
legend('Test set','Training set')

% Plot the evolution of the hyperparameters
figure(3)
plot(x_axis,results.alpha,'b*-')
xlabel('Number of hyperparameter updates')
ylabel('alpha value')
% ------------------------------------------------------------------------
```

Q.3) Parzen Windows & Nearest Neigbours

```
%--------------------------------------------------------------------------
%ECE 662
%Homework 2
%Question 3.a
%Parzen windows
%Fisheriris dataset
%------------------------------------------------------------------
%load the data and prepare for Parzen
load fisheriris
w_length=1.5;
data=[meas(:,1), meas(:,2)];
l=length(data);

groups = ismember(species,'setosa');
[train, test] = crossvalind('holdOut',groups);
train_labels=groups(train);
test_labels=groups(test);

train_data=data(train,:);
test_data=data(test,:);
[train_size x]=size(train_data);
[test_size x]=size(test_data);
distance=zeros(train_size,test_size);
count=0;%for correctly classified data
%create distance matrix
for i=1:test_size
    for j=1:train_size
        distance(i,j)=norm(test_data(i,:)-train_data(j,:));
    end
end
%Find the data falling into the window and classify
for i=1:test_size
    neigbors=0;
    neigbors_tot=0;
    for j=1:train_size
        if(distance(i,j)<=w_length)
            %training point is in the window
            neigbors=neigbors+1;
            %sum the labels of training points
            neigbors_tot=neigbors_tot+train_labels(j);
        end
    end
    %classification starts here for i-th test data
    if(neigbors_tot<=(neigbors/2))
        if(~test_labels(i))
            count=count+1;
        end
    else
        if(test_labels(i))
            count=count+1;
        end
    end
end
classification_rate=count/test_size
```

```
% ------------------------------------------------------------------------
%ECE 662
%Homework 2
%Question 3.a
%Parzen windows
%Glass Identification
% ------------------------------------------------------------------------
%load the data and prepare for Parzen
S=load ('glass.data');
%window length
w_length=20;
%We have 9 features
data = [S(:,2),S(:,3),S(:,4),S(:,5),S(:,6),S(:,7),S(:,8),S(:,9),S(:,10)];
l=length(data);
groups = ismember(S(:,11),7);
K=load('svm.csv');
train=logical(K(:,1));
test=logical(K(:,2));
train_labels=groups(train);
test_labels=groups(test);

train_data=data(train,:);
test_data=data(test,:);
[train_size x]=size(train_data);
[test_size x]=size(test_data);
distance=zeros(train_size,test_size);
count=0;%for correctly classified data
%create distance matrix
for i=1:test_size
    for j=1:train_size
        distance(i,j)=norm(test_data(i,:)-train_data(j,:));
    end
end
%Find the data falling into the window and classify
for i=1:test_size
    neigbors=0;
    neigbors_tot=0;
    for j=1:train_size
        if(distance(i,j)<=w_length)
            %training point is in the window
            neigbors=neigbors+1;
            %sum the labels of training points
            neigbors_tot=neigbors_tot+train_labels(j);
        end
    end
    %classification starts here for i-th test data
    if(neigbors_tot<=(neigbors/2))
        if(~test_labels(i))
            count=count+1;
        end
    else
        if(test_labels(i))
            count=count+1;
        end
    end
end
classification_rate=count/test_size
```

```
%---------------------------------------------------------------------------
%ECE 662
%Homework 2
%Question 3.b
%K-Nearest Neighbour
%Fisheriris data
%---------------------------------------------------------------------------
clear;
%load the data and prepare for k-NN
load fisheriris
k=20;
data=[meas(:,1), meas(:,2)];
l=length(data);
groups = ismember(species,'setosa');
[train, test] = crossvalind('holdOut',groups);
train_labels=groups(train);
test_labels=groups(test);

train_data=data(train,:);
test_data=data(test,:);
[train_size x]=size(train_data);
[test_size x]=size(test_data);
distance=zeros(train_size,test_size);
count=0;%for correctly classified data
%create distance matrix
for i=1:test_size
    for j=1:train_size
        distance(i,j)=norm(test_data(i,:)-train_data(j,:));
    end
end
%Find the k nearest data point and classify
for i=1:test_size
    [sorted index]=sort(distance(i,:));
    %classification
    if(sum(train_labels(index(1:k)))>(k/2))
        if(test_labels(i))
            count=count+1;
        end
    elseif(~test_labels(i))
            count=count+1;
    end
end
classification_rate=count/test_size
%---------------------------------------------------------------------------
```

```
%-------------------------------------------------------------------------
%ECE 662
%Homework 2
%Question 3.b
%K-Nearest Neighbour
%Glass Identification
%-------------------------------------------------------------------------
clear;
%load the data and prepare for k-NN
S=load ('glass.data');
k=20;
%We have 9 features
data = [S(:,2),S(:,3),S(:,4),S(:,5),S(:,6),S(:,7),S(:,8),S(:,9),S(:,10)];
l=length(data);
groups = ismember(S(:,11),7);
K=load('svm.csv');
train=logical(K(:,1));
test=logical(K(:,2));
train_labels=groups(train);
test_labels=groups(test);

train_data=data(train,:);
test_data=data(test,:);
[train_size x]=size(train_data);
[test_size x]=size(test_data);
distance=zeros(train_size,test_size);
count=0;%for correctly classified data
%create distance matrix
for i=1:test_size
    for j=1:train_size
        distance(i,j)=norm(test_data(i,:)-train_data(j,:));
    end
end
%Find the k nearest data point and classify
for i=1:test_size
    [sorted index]=sort(distance(i,:));
    if(sum(train_labels(index(1:k)))>(k/2))
        if(test_labels(i))
            count=count+1;
        end
    elseif(~test_labels(i))
            count=count+1;
    end
end
classification_rate=count/test_size
%-------------------------------------------------------------------------
```