ECE 662:Pattern Recognition and Decision-Making Processes Homework Assignment Two

****

Collaborators: None

April 16, 2008

1 Question 1: Numerical Experiments with the Fisher Linear discriminant

Here the issue to address is the following: We consider observations $\mathbf{X}$ coming from two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ and a set of training data divided into the sets χ_1(training data from class 1) and χ_2(training data from class 2). We learned that an adequate hyperplane can be drawn to appropriately separate the observations into the classes $\mathcal{C}_1$ and $\mathcal{C}_2$ by using the vector ω_o which is the argmax of the cost function

$$J(\omega) = \frac{\omega S_B \omega}{\omega S_W \omega}$$

where

- S_B is the "between classes scatter matrix" and is defined in *Duda and Hart*.
- S_W is the "within classes scatter matrix" and is defined in *Duda and Hart*.

The solution ω_o is the Fisher linear discriminant and is defined as $\omega_o = S_W^{-1}(m_1 - m_2)$ where m_1 and m_2 are the (training) sample means of each class.

The question raised is the following: why not instead optimize the following cost function

$$J(\omega) = \omega S_B \omega$$

In this case the solution would be $\omega_o^{(modified)} = (m_1 - m_2)$, which we will refer to here as the *modified* discriminant. To demonstrate and explain the differences (if any) between the two approaches we implement an adequate numerical experiments to compare the two approaches.

1.1 Numerical Experiments

The cost function

$$J(\omega) = \frac{\omega S_B \omega}{\omega S_W \omega}$$

can be interpreted as maximizing the *projected difference in the means* while minimizing the *projected class variances*. In the above cost function it can be shown that the projected *difference in the means* is controlled by the term in the numerator $\omega S_B \omega$ while the *projected class variances* are controlled by the

term in the denominator $\omega S_W \omega$. Hence the choice of only using the *modified* cost function

$$J(\omega) = \omega S_B \omega$$

reduce to neglecting to minimize the *projected class variances* in the optimization procedure. That is we fails to take into consideration the effect "within classes scatter". Therefore the comparison of the two methods is done by evaluating the effect of "within classes scatter". To this end, we compare the two approaches with three cases. First, we compare the separation performance of the two methods when the two classes in the data have both small "within classes scatter". Second, we compare them when these two classes both have a large "within classes scatter". Finally, the performances are compared when one class has small "within classes scatter" while the other one has large "within classes scatter". In our experiment the data was chosen as follows: The original observation data $\mathbf{X}$ comes from two bi-variate Gaussian vector and the two classes have the same a-priori probabilities. Thus, the resulting observation data after the projection (i.e. $Y = \omega_o^T \mathbf{X}$ or $Y = \omega_o^{(modified)^T} \mathbf{X}$) might be realization from two Gaussian random variables (one for each 1-feature class) and the probabilities of misclassification can be easily calculated. In each case below we visualize $\mathbf{X}$ and Y. Moreover, the performance of the projection methods is quantified by computing the probability of misclassification from observing Y. The misclassification performances are with respect to a Bayesian Framework. For a more robust performances comparison, large number of data samples were used in testing the methods.

1.1.1 Case1: Small "within classes scatter" for both $\mathcal{C}_1$ and $\mathcal{C}_2$

When both classes have small "within classes scatter" it is observed in Fig.1 and Fig.2 that the observation data obtained after projection in both cases have good separation. However the performance is better when the cost function includes the "within classes scatter" information. Again, although both performances are acceptable, the probability of misclassification after the *modified* discriminant projection in Fig.2 is 3.6 times larger than when the Fisher cost function is utilized (i.e. the cost function leading Fisher Linear Discriminant) as shown in Fig.1.

1.1.2 Case2: Large "within classes scatter" for both $\mathcal{C}_1$ and $\mathcal{C}_2$

When both classes have large "within classes scatter" it is observed in Fig.3 and Fig.4 that the observation data obtained after projection in both cases have relatively poor separation. After the Fisher discriminant projection the

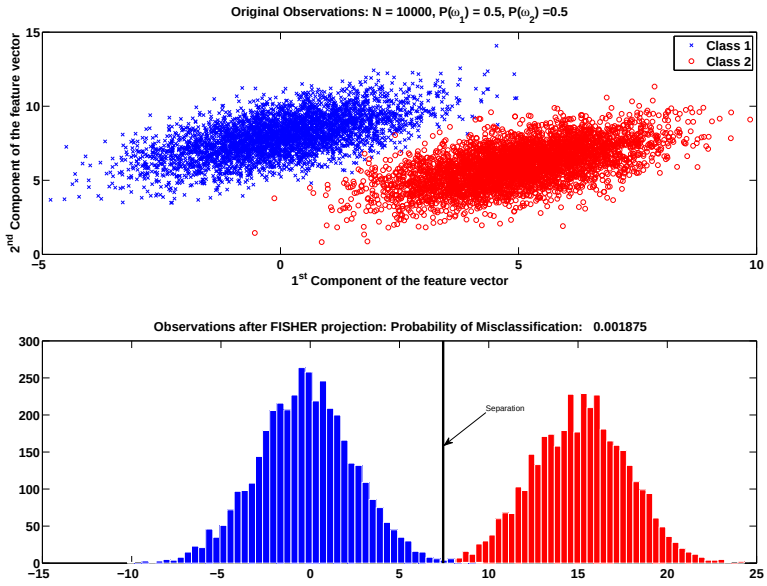

Figure 1: FISHER discriminant when both classes have small "within classes scatter"

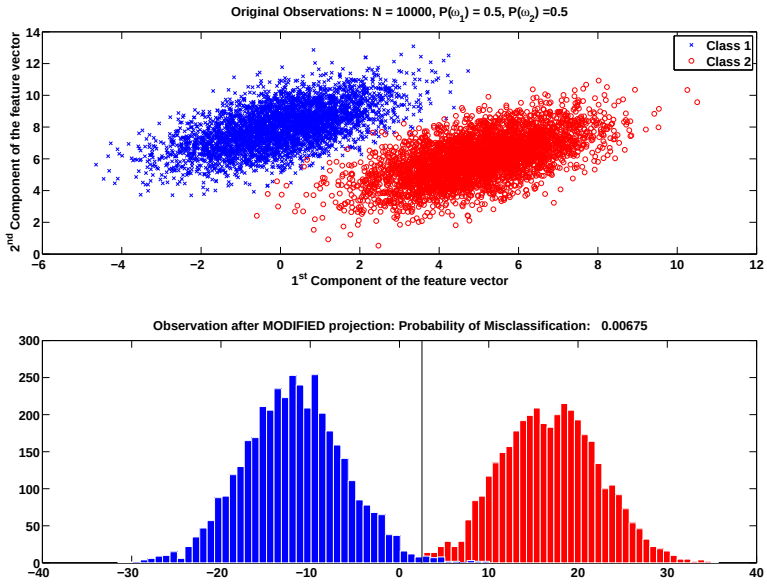

Figure 2: MODIFIED discriminant when both classes have small "within classes scatter"

probability of misclassification was 17.8% and after the projection with the modified discriminant, the probability of misclassification is 22.9 %. Then although both separation are relatively poor, the Fisher discriminant method performs better in this case also.

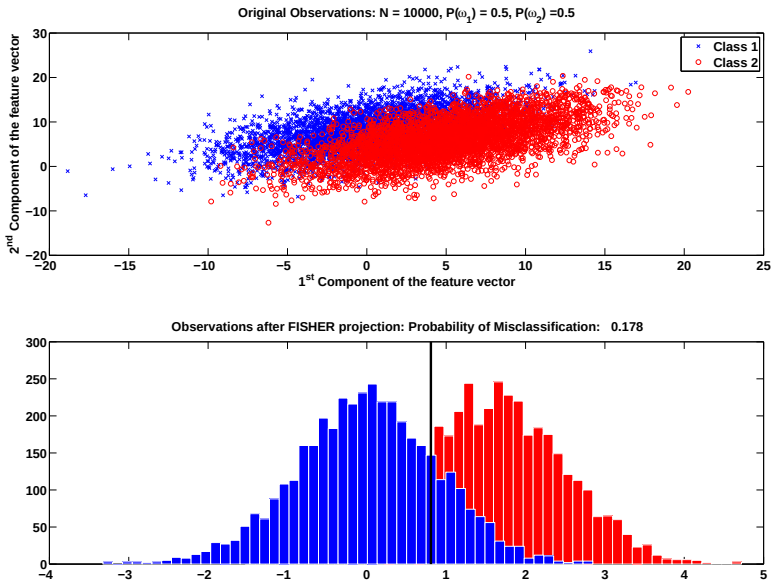

Figure 3: FISHER discriminant when both classes have large "within classes scatter"

1.1.3 Case3: Small "within classes scatter" for both $\mathcal{C}_1$ and large "within classes scatter" $\mathcal{C}_2$

Here the original data is such that one class have a small "within classes scatter" while the other have a large "within classes scatter". We still observe in the performance results in Fig.5 and Fig.6 that better performance are achieved when the Fisher linear discriminant as compared to the so-called modified discriminant.

In all the above cases the Fisher discriminant have better performance than the *modified* discriminant in terms of classes separability after projection. In fact the these numerical experiments are just supporting what one would expect from a qualitative analysis of the two methods. The Fisher discriminant is attained by optimizing a cost function that encapsulates two criterion that are both crucial to separability of classes after projection. These are criterion are 1) Maximizing the *projected difference in the means* and 2)

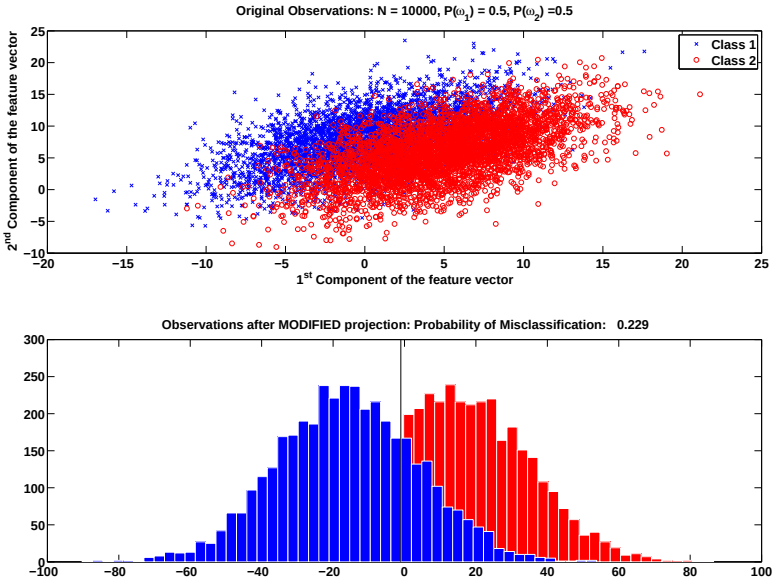

Figure 4: MODIFIED discriminant when both classes have small "within classes scatter"

Minimizing the *projected class variances*. By using the *modified* discriminant one neglect the second criterion in the cost function hence performance is bound to degrade a certain amounts. The amount of performance degradation depends on the datasets under consideration.

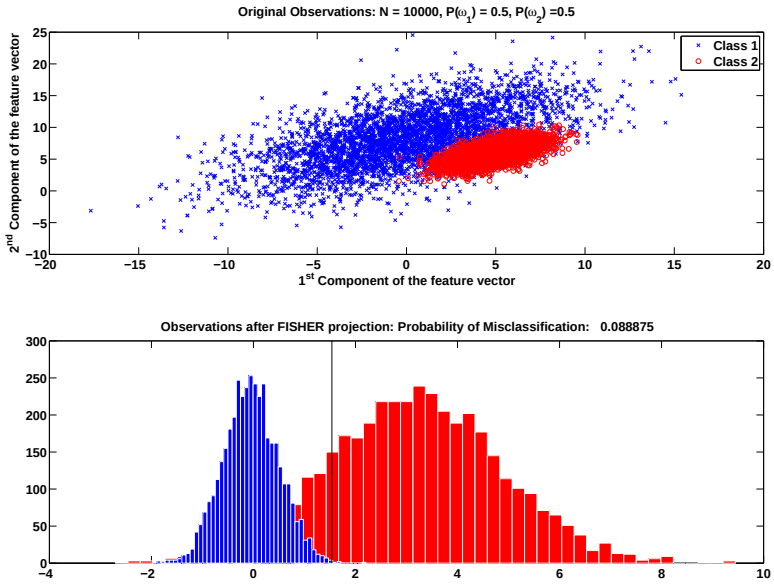

Figure 5: FISHER discriminant when one class has large "within classes scatter" and another a small "within classes scatter"

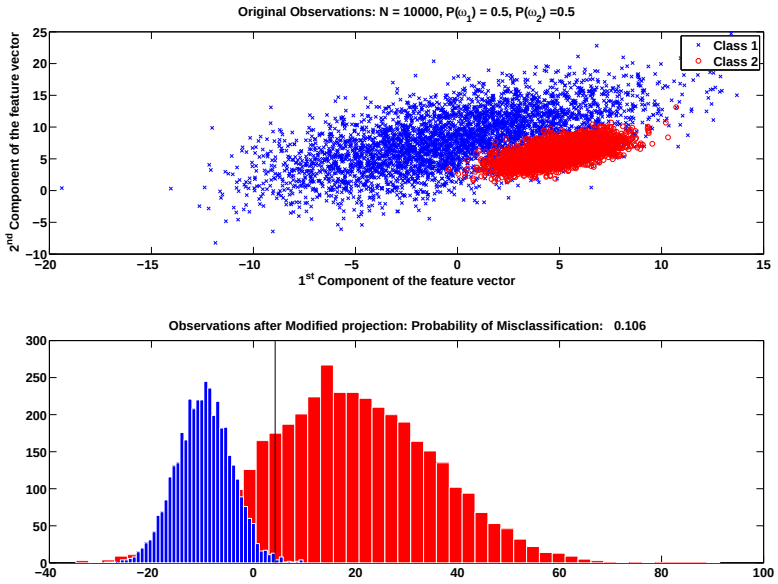

Figure 6: MODIFIED discriminant when one class has large "within classes scatter" and another a small "within classes scatter"

1.2 Codes for the problem one.

```
HOMEWORK TWO - PROBLEM 1
%% Author: ************************

%% GENERATION  OF DATASETS
clear all;
clc
% Correlation coefficent of features within a class
rho1 = [1];
rho2 = [1];
% Total number of observations N.
N = rand(1,10000);
%The number of samples from class 1 with prior probability P(w1)=0.5.
N1 = sum(N < 0.5);
%The number of samples from class 1 with prior probability P(w1)=0.5.
N2 = sum(N > 0.5);
%Mean of Class 1.
u1 = [0 8];%u1 = [1 2];
%Mean of Class 2.
u2 = [5 6];
%Covariance matrices of Class 1
Sigma1 = 20*[rho1 0.6 ; 0.6 rho1];
%Covariance matrices of Class 2
Sigma2 = 2*[rho2  0.6;  0.6 rho2];
%Generation of N1 class 1 data
r1 = mvnrnd(u1,Sigma1,N1);
%Generation of N2 class 2  data
r2 = mvnrnd(u2,Sigma2,N2);
% Training data (class 1)
r1_t = r1(1:1000,:);
% Training data (class 2)
r2_t = r2(1:1000,:);
% Observed data (class 1)
r1_o = r1(1001:end,:);
% Observed data (class 2)
r2_o = r2(1001:end,:);
subplot(2,1,1);
plot(r1_o(:,1),r1_o(:,2),'x'); hold on;
plot(r2_o(:,1),r2_o(:,2),'or');
title('Original Observations: N = 10000, P(\omega_1) = 0.5, P(\omega_2) =0.5');
```

```
legend('Class 1','Class 2');
xlabel('1^{st} Component of the feature vector');
ylabel('2^{nd} Component of the feature vector');

%% FISHER LINEAR DISCRIMINANT ANALYSIS AND VARIATIONS
%Mean and Scatter matrices for classes
%Mean and Scatter matrices
m1 = mean(r1_t);
m2 = mean(r2_t);
% Scatter matrix: class 1
%------------------------------------------------------------------------
S1_11 = zeros(length(r1_t),1);S1_12 = zeros(length(r1_t),1);
S1_21 = zeros(length(r1_t),1);S1_22 = zeros(length(r1_t),1);
for k = 1:1:length(r1_t)
S1.(['M',num2str(k)]) = (r1_t(k,:)-m1)'*(r1_t(k,:)-m1);
S1_11(k) = S1.(['M',num2str(k)])(1,1);S1_12(k) = S1.(['M',num2str(k)])(1,2);
S1_21(k) = S1.(['M',num2str(k)])(2,1);S1_22(k) = S1.(['M',num2str(k)])(2,2);
end
Sc1 = [mean(S1_11) mean(S1_12); mean(S1_21) mean(S1_22)];
% Scatter Matrix: Class 2
%------------------------------------------------------------------------
S2_11 = zeros(length(r2_t),1);S2_12 = zeros(length(r2_t),1);
S2_21 = zeros(length(r2_t),1);S2_22 = zeros(length(r2_t),1);
for k = 1:1:length(r2_t)
S2.(['M',num2str(k)]) = (r2_t(k,:)-m2)'*(r2_t(k,:)-m2);
S2_11(k) = S2.(['M',num2str(k)])(1,1);S2_12(k) = S2.(['M',num2str(k)])(1,2);
S2_21(k) = S2.(['M',num2str(k)])(2,1);S2_22(k) = S2.(['M',num2str(k)])(2,2);
end
Sc2 = [mean(S2_11) mean(S2_12); mean(S2_21) mean(S2_22)];
%------------------------------------------------------------------------
%Computation of within class scatter matrix.
%S_w = (Sc1 + Sc2); % for the Fisher Linear Discriminant
S_w =eye(2); % for the Modified Linear Discriminant
S_B = (m1-m2)'*(m1-m2);
% Resulting projection vector
w_o = inv(S_w)*(m1'-m2');
%% ViSUALIZATION AND PERFORMANCE
subplot(2,1,2);
y1 = r1_o*w_o;
y2 = r2_o*w_o;
hist(y1,50); hold all ;
```

```
h1 = findobj(gca,'Type','patch');
set(h1,'FaceColor','r','EdgeColor','w')
hist(y2,50)
h2 = findobj(gca,'Type','patch');
set(h2,'FaceColor','b','EdgeColor','w')
%Computation of Classification Error
mm1 = mean(y1);
mm2 = mean(y2);
mm = mean([mm1  mm2]);
P_error = (sum(y2>mm)+sum(y1<mm))/(length(y1)+length(y2))
plot([mm mm],[0 300],'k');
title(['Probability of Misclassification:',num2str(P_error)]);
```

2 Question 2: Neural Network and Support Vector Machine Classifier.

In this problem the objective are the following:

- Perform numerical experiments to investigate selected aspects of classification via the Neural Network approach.
- Perform numerical experiments to investigate selected aspects of classification via Support Vector Machine.
- Design an experiment to compare the approaches for a selected dataset.

2.1 Part 1: Experiments with Neural Network classification

2.1.1 Experiment 1: Neural Network and Classes separation

This first experiment is targeted at evaluating the performance of Neural Networks as a function of the separation of the classes in the data. For better visualization, we use here two classes. The two classes in the observation data are observations from bivariate Gaussian vectors with covariance matrices Σ_1 and Σ_2. We start with a dataset with good separation corresponding to bivariate Gaussian vectors with covariance matrices $\Sigma_1^{(o)}$ and $\Sigma_2^{(o)}$ as shown in the top left plot in Fig.7. Separation between classes is induced by generating new datasets that comes from bivariate Gaussian vectors with covariance matrices $a\Sigma_1^{(o)}$ and $a\Sigma_2^{(o)}$ where $a > 1$ is a scaling parameter that indicate the level of classes separation (i.e. separation parameter). This is illustrated in Fig.7 where several datasets with varying degree of classes separation are shown. As expected classes separation decrease as the value of the scaling parameter a increases.

In our implementation script we adapted a Neural Network classifier that was downloaded from the location *http://isp.imm.dtu.dk/toolbox/ann/*. This program implements a Neural Network binary classifier using a logistic sigmoidal function. The network is trained using a BFGS optimization algorithm. For each dataset (i.e. different value of a which means different level of classes separation), we use training sample ($N_{training} = 50$ samples) to train the network and then use a test sample ($N_{test} = 200$ samples) to assess classification performance by computing the misclassification error rate from the output of the network. The number of hidden layers was set to 10 here. The results are plotted in Fig.8. The key observation for this experiment is

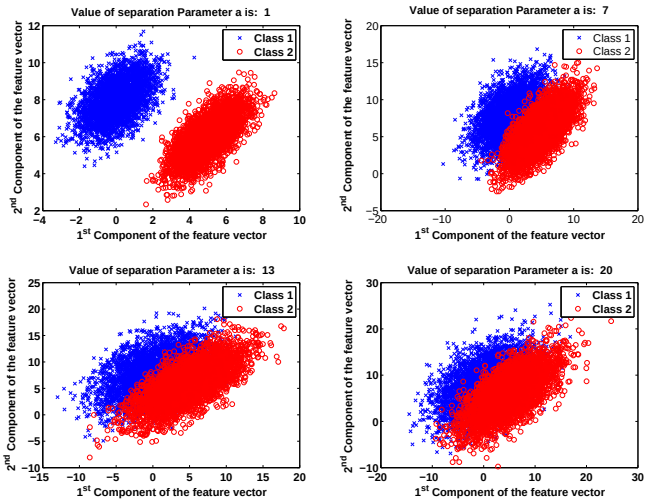

Figure 7: Illustration of class separation with the separation parameter a. Large a means less separation

to notice that the misclassification error rate of the Neural Network increase linearly with classes separation.

2.1.2 Experiment 2: Neural Network and number of hidden layers

Now we seek to use the same data set and assess the effect of the number of hidden layers on classification performance. We choose a dataset with separation corresponding to a separation parameter of a=5 (cf. Fig.7) for visualization. We then vary number of hidden layers from 1 to 20 and compute the misclassification rate each time. The results are plotted in Fig.9. From the figure it seems that the number of hidden layers did not have a major effect on classification performance in this case. The experiment was repeated for different level of separation and similar results were obtained. We postulate here that for this dataset a limited number of layers were sufficient to attain the upper bound on what a Neural Network is able to do in turn of classifying the data. Thus, there is no added gain with more hidden layers. Since adding more layers increase computational cost, it seems that one should consider (if possible) the minimal number of layers that is sufficient for maximum possible performance with a given type of dataset.

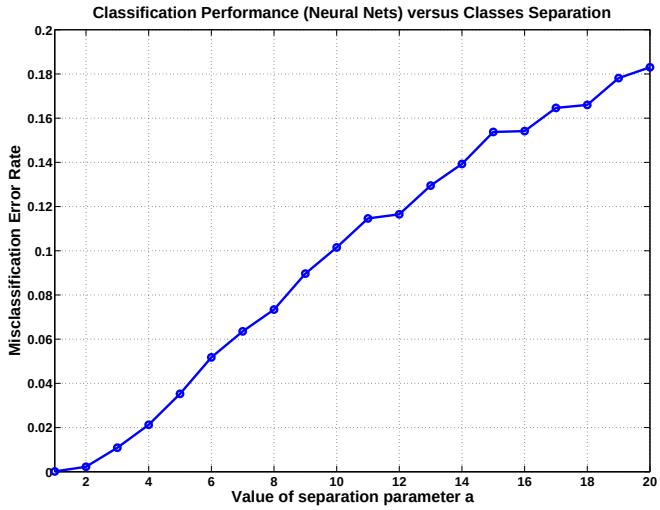

Figure 8: Neural Network classification and classes separation experiment. Number of hidden layers used Nh=10

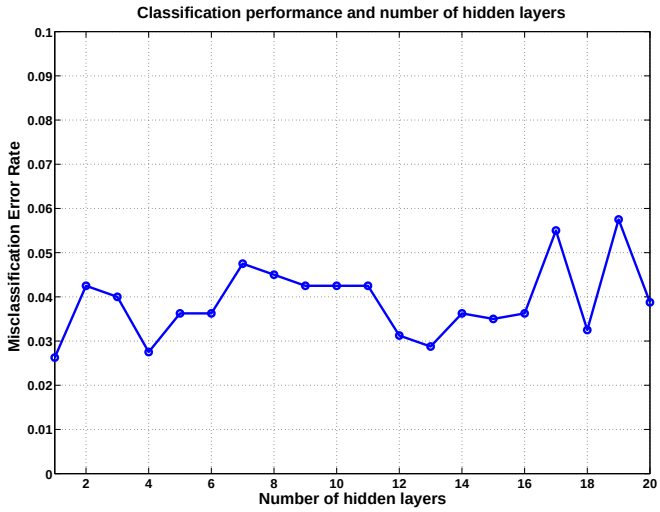

Figure 9: Classification performance versus number of hidden layers

2.2 Experiments with the Support Vector Machine

2.2.1 Illustration of Support Vector Machine Classification

Here we discuss the support vector machine classifier that is used in the problem. In our implementation script for this portion we have used the MATLAB functions *svmtrain* and *svmclassify*. *svmtrain* is used with the training dataset and generate the a set of parameters that are used by *svmclassify* to separate the test dataset. The *svmtrain* was set to use a linear kernel (i.e. dot product). More information on the functions can be found through MATLAB help. We use similar data set as above. The specific dataset (among the above variations) is the one corresponding to a separation parameter of a=5 (cf. Fig.7). The size of training sample is $N_{training}$ = 50 samples and the size of the test sample is N_{test} = 200 samples. The results of the classification via Support Vector Machine are shown in Fig.10. The misclassification error rate in this case was equal to 0.04.

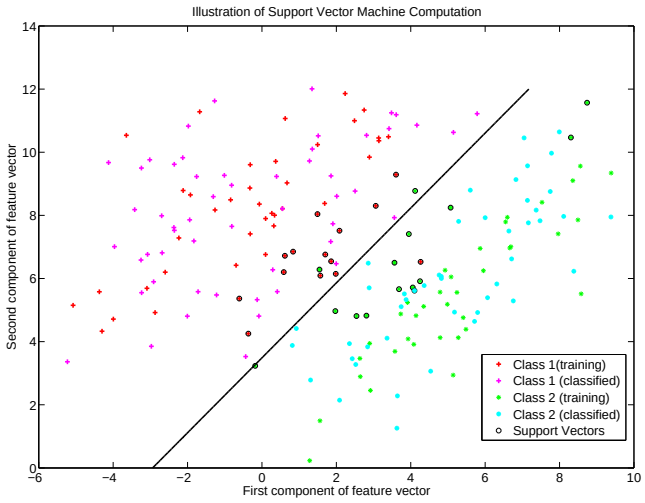

Figure 10: A Support Vector Machine classification example. Misclassification Error is 0.04.

2.2.2 Support Vector Machine and Classes separation

Using the same approach outlined with the Neural Network case, we week to evaluate the performance of Support Vector Machine classification as a function of class separation. The same settings and sample sizes used for the Neural Network classifier is also used here. The results attained are outlined

in Fig.11. As in the case of Neural Network, it can be observed from the plot that here also the misclassification rate increase linearly with classes separation. For the datasets we used the two methods seems to show similar behaviors. To verify this we design a more robust experiment for a more direct comparison.

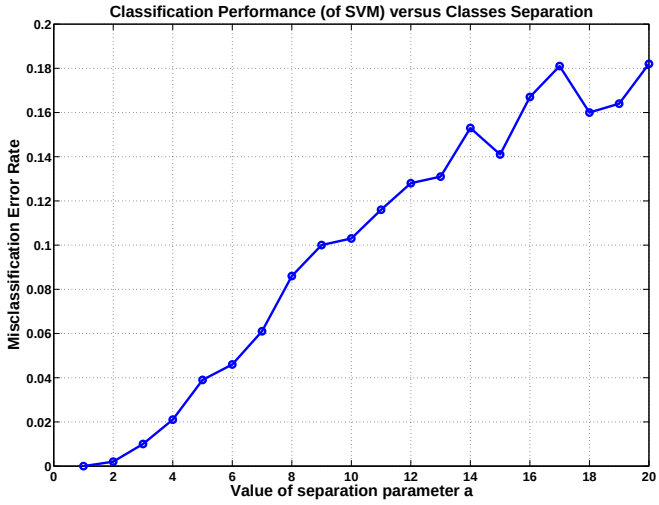

Figure 11: Support Vector Machine performance as a function of Neural as a function of classes separation

2.2.3 Comparison of Support Vector Machine and Neural Networks

To compare the two methods in a more robust manner. The Neural Network here has 10 hidden layers. We compare the methods by using datasets at different separation. At each separation, we generate a training dataset and testing dataset that are used by both methods. A misclassification error rate is computed for each method. To guarantee robustness in the comparison, the separation is maintained and the test is repeated 20 times. The 20 misclassification error rates are averaged. The following procedure is done for separation level. Therefore we can robustly compare the methods at each separation (at least for this type of datasets). The results are plotted in Fig.12. The following is observed: When the two classes are relatively well separated (i.e. small values for a), the Support Vector Machine performs slightly better with this dataset. As the separation decreases, it appears that the two methods perform equally. It was however noticed in this comparison

that the computation of the Neural Network was faster than that of the Support Vector Machine classifier. Hence here one would clearly choose the Neural Network Classifier if computational time was an issue.

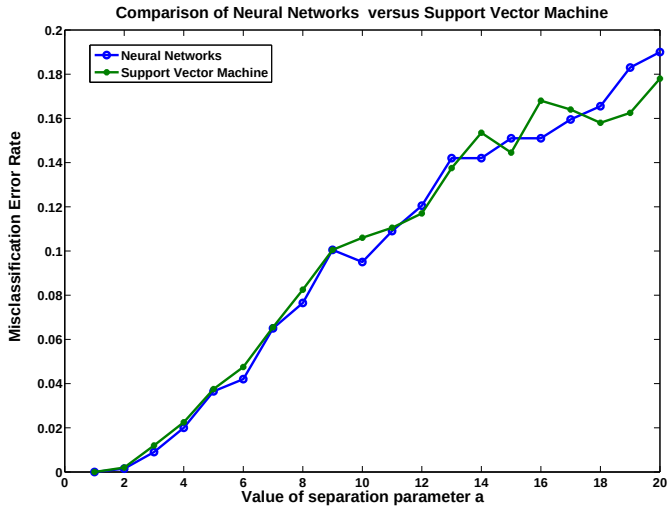

Figure 12: Comparison of Support Vector Machine and Neural Network classifiers

2.3 Scripts for problem 2

1. Script for SVM experiment

```
%% HOMEWORK TWO - PROBLEM 2
%Author: **********************
%% GENERATION  OF DATASETS
clear all;
clc
%% THE DATASET
%Training dataset.
x = [r1_t ; r2_t];
%Labels of training dataset.
t =[zeros(length(r1_t),1); ones(length(r2_t),1)];
% Testing dataset.
x_test = [r1_o ; r2_o];
% Labels of testing dataset (used to compute Performance).
t_test =[zeros(length(r1_o),1); ones(length(r2_o),1)];

%% USING MATLAB SVM FUNCTIONS
svmStruct = svmtrain(x,t,'showplot',true);
%Performing the classification (MATLAB FUNCTION)
classes = svmclassify(svmStruct,x_test,'showplot',true);
%Misclassification rate Computation
P(k)= sum(abs(classes-t_test))/length(t_test)
end
```

3 Question 3: Comparison of Parzen Window, K-Nearest Neighbor and Nearest Neighbor techniques.

The objective here is to design three classifiers using 1) the Parzen Window, 2) The K-nearest neighbor and 3) the Nearest Neighbor. The dataset used above (i.e. comparison of Neural Network and Support Vector Machine) is also used in this problem. Although for better visualization of the parameter estimation via Parzen window, the data has been projected to 1-D by using the Fisher discriminant vector discussed in problem 1. The original data and the projected 1-D data are shown in Fig.13. First, we design a Parzen window density estimation method using a trained set which was also obtained from projection of an original (2-D) training set. The estimated densities of the two classes are then used for classification. The classification here hinges on the accuracy of the density estimation which in turns depends on the size of the training set and the scaling parameter h. Since the training set is fixed here we look only look at here the effect of h on the density estimation. We list below many plots (i.e. 14,15,16,17,18, ,19,20,21,) of the density estimation for the two classes along with the associated misclassification error rate for different value of h, the scaling parameter in the Parzen window method. There is no consistency in the density estimation for the various values of h although for the values of h=0.6,0.7,0.8, the density estimation and the corresponding probability of error seems to converge. This density they converges is the correct here (we know that since was the data was synthesized). In practice however we do not have this information and the above dilemma persists. If large training data is not available, the Parzen window does not seems to be a stable method. For the purpose of comparing the Parzen window technique to the k-Nearest Neighbor(kNN) and Nearest Neighbor(NN) we choose h=0.8 and then the correct misclassification error rate is 0.06. To compare the above method to the k-NN and the NN we utilize the code from *www.mathworks.com/matlabcentral/files/15562/kNearestNeighbors*. When the the code is applied to the same training set and test set, we have a misclassification rate of 0.03 for the k-NN. For the NN case we have a misclassification rate of 0.05. Therefore for our data set the k-NN and the NN algorithm performed better than the Parzen window method.

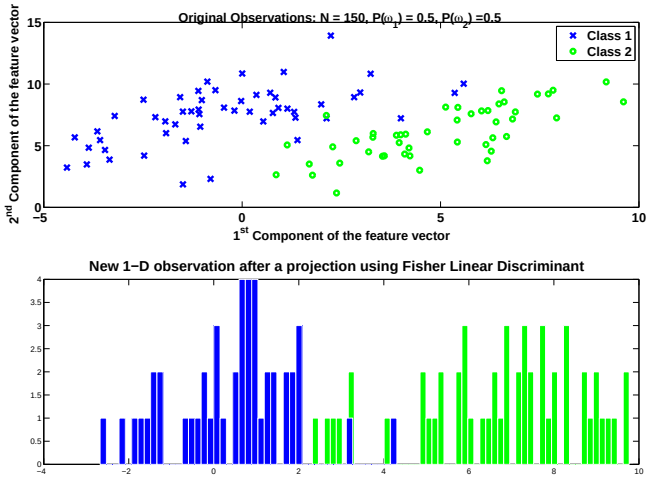

Figure 13: Original observed data and projected 1-D observed data

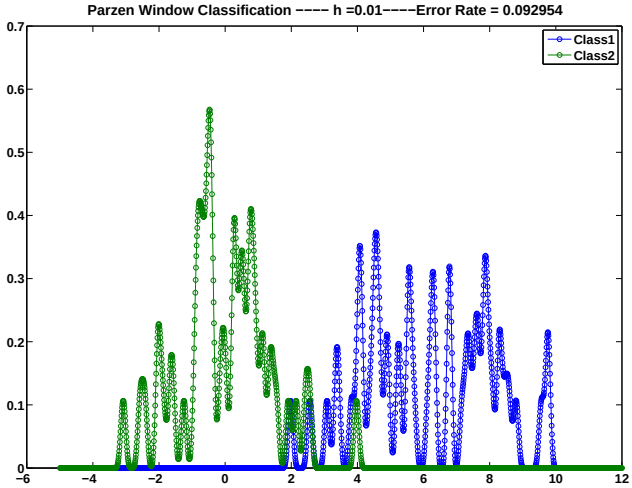

Figure 14: Parzen window method with h=0.01

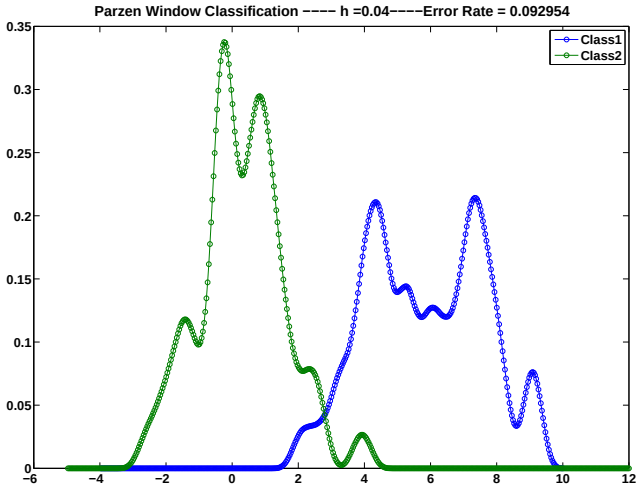

Figure 15: Parzen window method with h=0.04

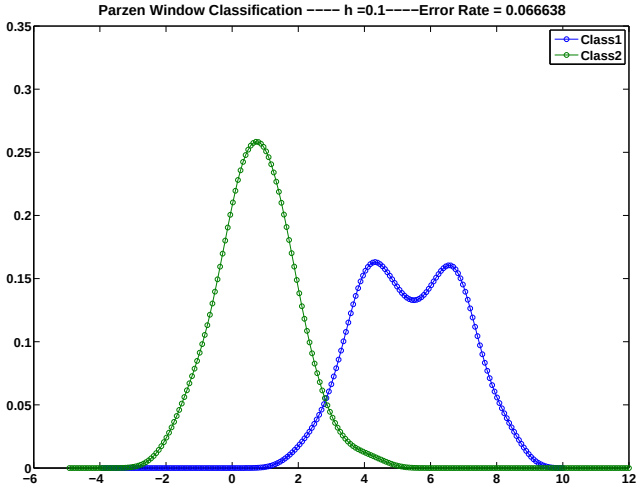

Figure 16: Parzen window method with h=0.1

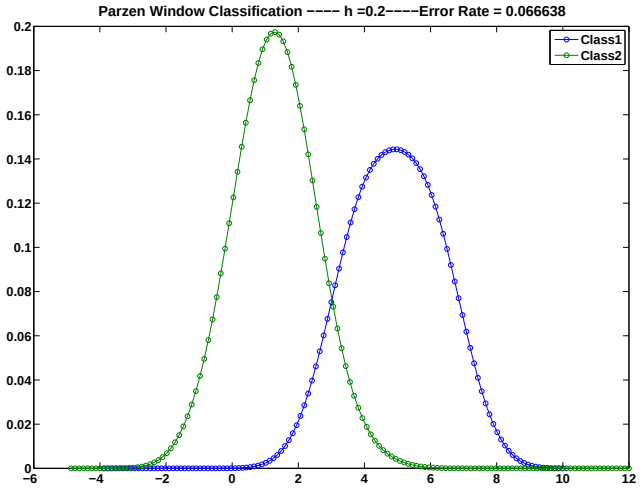

Figure 17: Parzen window method with h=0.2

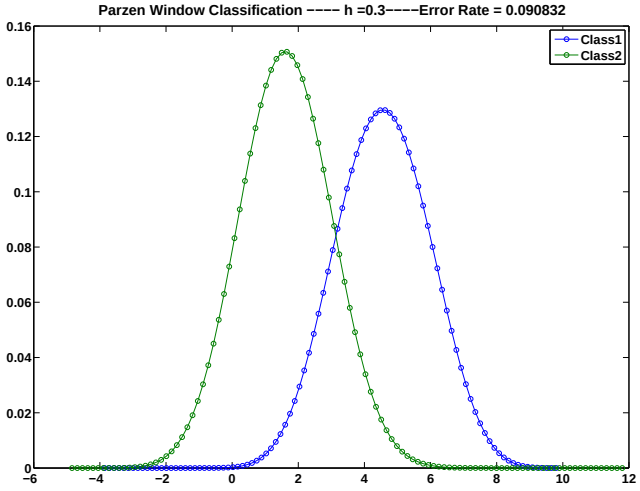

Figure 18: Parzen window method with h=0.3

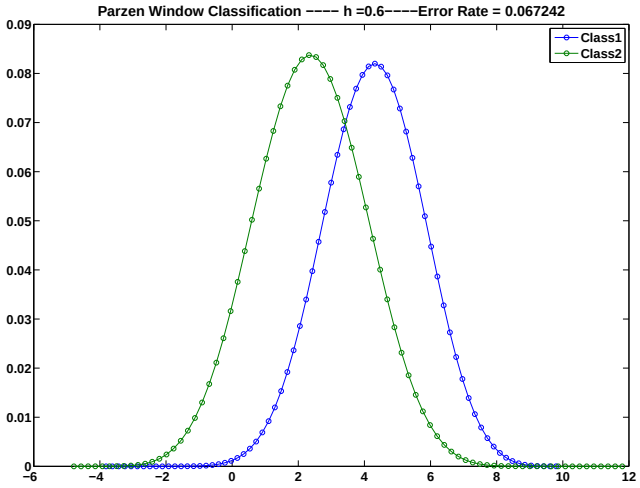

Figure 19: Parzen window method with h=0.6

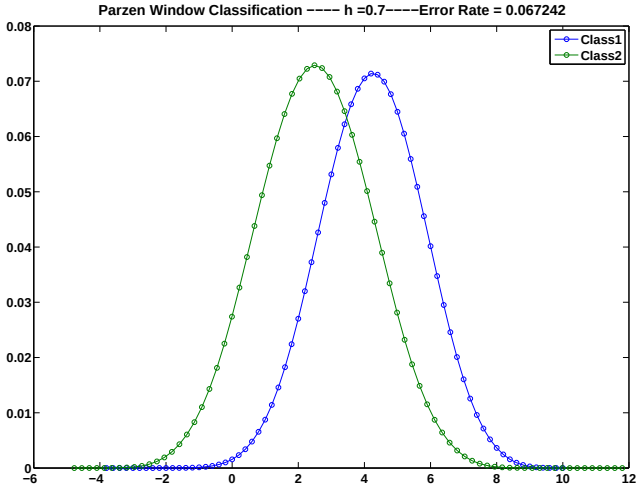

Figure 20: Parzen window method with h=0.7

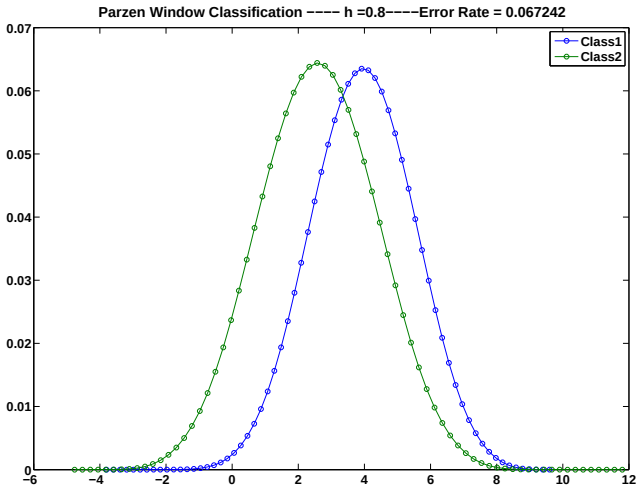

Figure 21: Parzen window method with h=0.8

3.1 Code for Problem 3.

Code for Parzen Window Classification

```
%DATA SET FOR PARZEN WINDOW ESTIMATION
%Training data
y1_t = r1_t*w_o;
y2_t = r2_t*w_o;
% Test dataset
y1 = r1_o*w_o;
y2 = r2_o*w_o;
%Window width
h=0.8;
%Estimating density of class 1;
xh = [-4:h:10];
z1=hist(y1_t,xh);
T1 = conv(z1,parzenwin(50));
x1 = [1:1:length(T1)];
x1 = ((x1/max(x1))*(max(xh)-min(xh)))+min(xh);
%z2 = hist(y2_t,length(y2_t));
T1 = (T1/sum(T1))*inv(h);
plot(x1,T1,'o-')
```

```
hold all;
%Estimating density of class 2;
xh = [-5:h:12];
z2=hist(y2_t,xh);
T2 = conv(z2,parzenwin(50));
x2 = [1:1:length(T2)];
x2 = ((x2/max(x2))*(max(xh)-min(xh)))+min(xh);
%z2 = hist(y2_t,length(y2_t));
T2 = (T2/sum(T2))*inv(h);
plot(x2,T2,'o-')
legend('Class1','Class2')
title(['Parzen Window Classification ---- h =',num2str(h),'----Error Rate = ' num2s
%Class2 classification error;
T1new1 = spline(x1,T1,y1);
T1new2 = spline(x2,T2,y1);
P1_error = (sum(T1new1 < T1new2))/length(T1new2)
%Class2 classification error;
T2new1 = spline(x1,T1,y2);
T2new2 = spline(x2,T2,y2);
P2_error = (sum(T2new1 > T2new2))/length(T2new2)
P_error = mean([P1_error P2_error])
```