ECE 662 Spring 2008
Homework 2 Report

April 16, 2008

1 Problem 1

In the Parametric Method section of the course, we learned how to draw a separation hyperplane between two classes by obtaining $\mathbf{w}_{opt}$, the argmax of the cost function $J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$. The solution was found to be $\mathbf{w}_{opt} = S_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$, where $\mathbf{m}_1$ and $\mathbf{m}_2$ are the sample means of each class, respectively. Some students raised the question: can one simply use $J(\mathbf{w}) = \mathbf{w}^T S_B \mathbf{w}$ instead (i.e. setting S_W as the identity matrix in the solution $\mathbf{w}_{opt}$)? Investigate this question by numerical experimentation.

1.1 Methodology

In this problem, we wish to compare the classification ability of the linear separating hyperplanes obtained by optimizing two different cost functions. The separating hyperplanes are to be used for two-class classification problem. One cost function simultaneously maximizes the between-class variance and minimizes the within-class variance for the two classes. This is the Fisher's Linear Discriminant (FLD) cost function. The other cost function just maximizes the between-class variance. For this problem, we have used a real dataset titled 'Pen-Based Recognition of Handwritten Digits' that was obtained from UCI Machine Learning repository [1]. The dataset is divided into a training set (7494 instances) and testing set (3498 instances). The dataset is described by 16 features, each of which are integers in the range 0-100. Based on the features, the dataset has been divided into 10 classes (class label 1-10). Since our problem is related to linear discriminant functions, we want a 2-class dataset that is approximately linearly separable. We investigated the dataset and found that for classes 1 and 4, the features 1 and 8 are approximately linearly separable in 2D and the features 1, 8 and 15 are approximately linearly separable in 3D (Figure 1(a) and (b)). Classes 1 and 4 have 780 and 719 training instances and 363 and 336 testing instances respectively.

First we considered the 2D case. We calculated the overall scatter matrix as

$$S_W = \sum_{\mathbf{x}\epsilon C_1} (\mathbf{x} - \mathbf{m}_1)(\mathbf{x} - \mathbf{m}_1)^T + \sum_{\mathbf{x}\epsilon C_2} (\mathbf{x} - \mathbf{m}_2)(\mathbf{x} - \mathbf{m}_2)^T$$

and the projection direction as $\mathbf{w_{opt}} = S_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$. $\mathbf{m}_1$ and $\mathbf{m}_2$ are the means of training data from the two classes. Optimizing the FLD cost function gives us the optimal weight vector but in order to derive the discriminant function $(\mathbf{w}^T\mathbf{x} + w_0)$, we also need the bias term w_0. For this, we posed the FLD problem as a linear least squares problem (see [2] Section 4.1.5) and obtained an expression for the bias of the form $w_0 = -\mathbf{w}_{opt}^T \mathbf{m}$ where $\mathbf{m}$ is the mean of the total training dataset, given by

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n = \frac{1}{N}(N_1\mathbf{m}_1 + N_2\mathbf{m}_2)$$

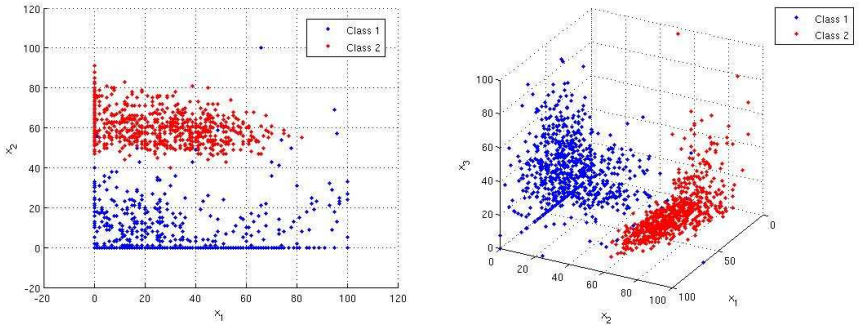
1.2 Results and Discussion

Figure 2(a) shows the $\mathbf{w}_{opt}$ (black line) and the separating hyperplane (magenta line) that are obtained by optimizing the FLD cost function. From the figure, it is apparent that this $\mathbf{w}_{opt}$ is infact a good choice for maximizing the between-class variance and minimizing the within-class scatter for the two classes. When we do not include the within-class scatter in the cost function, we are effectively setting S_W to identity matrix and in this case, $\mathbf{w}_{opt}$ is simply $(\mathbf{m}_1 - \mathbf{m}_2)$ i.e. the optimal weight vector is along the line joining the two class means. For this case, the weight vector $\mathbf{w}_{opt}$ and the separating hyperplane are shown in Figure 2(b). We refer to these as difference-of-mean weight vector and difference-of-mean hyperplane. From the figure, we can tell that in general these are not the best choices for the optimal weight vector and the separating hyperplane.

Interestingly enough, when we used these hyperplanes to classify the test data from the two classes, the results were a little counter intuitive. The difference-of-mean hyperplane performed slightly better than the FLD hyperplane. The numerical results are summarized in table 1. We had anticipated that FLD classifier will perform better than the difference-of-means classifier. The unexpected result may be due to the particular distributions of the two classes. We can definitely construct two class distributions where the FLD classifier performs better than the difference-of-means classifier. We demonstrate such a case later in our report. For right now, we proceed with discussing the performance comparison between the two classifiers on 3D dataset.

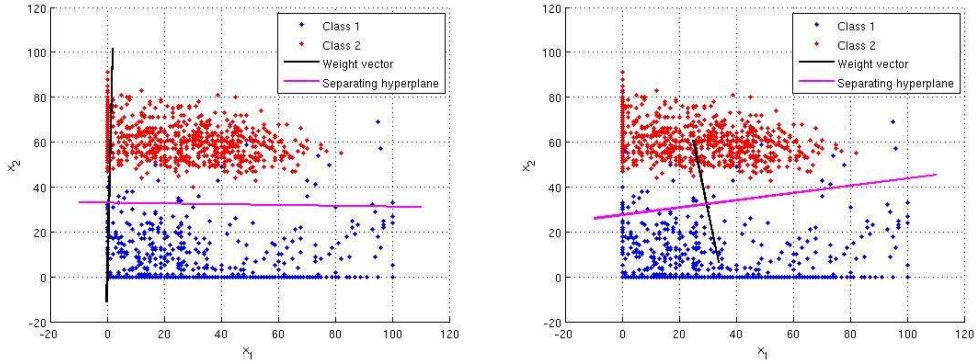
For the 3D case, we obtain $\mathbf{w}_{opt}$ and w_0 for both types of classifiers, similarly to the 2D case. Figures 3 (a) and (b) show the weight vector $\mathbf{w}_{opt}$ and the separating hyperplanes for the two classifiers. In 3D also, the difference-of-means classifier performs better than the FLD classifier as is apparent from table 2.

A little earlier in our report, we had commented that in general we expect FLD classifier to perform superior to the difference-of-means classifier. The reason is that FLD cost function simultaneously optimizes two different criteria (maximize between-class variance and minimize within-class variance) which is more logical. But there may be some data distributions for which the latter performs better. This is precisely the case with our selected real dataset. On `the` other hand, we can show that there exist data distributions for which FLD classifier will perform better. One such example is shown in figure 4(a) and (b), where we have shown the FLD classifier and difference-of-mean classifier results for 2 class Gaussian distributions. The means of the Gaussian distributions are horizontally displaced from each other and their axes of maximum variance are parallel to each other and inclined at an angle with the horizontal. From the numbers in table 3, we observes that FLD gives perfect classification (100% accuracy) for this dataset while difference-of-mean classifier yields 95.8% accuracy. From this exercise, we learned a good lesson that **it is better to use real life datasets to gauge the performance of different classifiers because an artificial dataset can always be generated to confirm the preconceptions that we may have regarding the relative classification abilities of different techniques.**



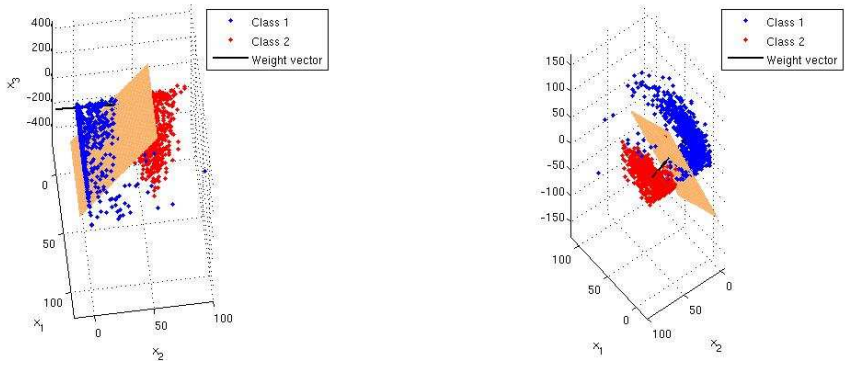
(a) Scatter plot of features 1 and 8 (b) Scatter plot of features 1, 8 and 15

Figure 1: Input data from 2 classes. The classes are nearly linearly separable in 2D as well as 3D



(a) (b)

Figure 2: Classification results in 2D



(a) (b)

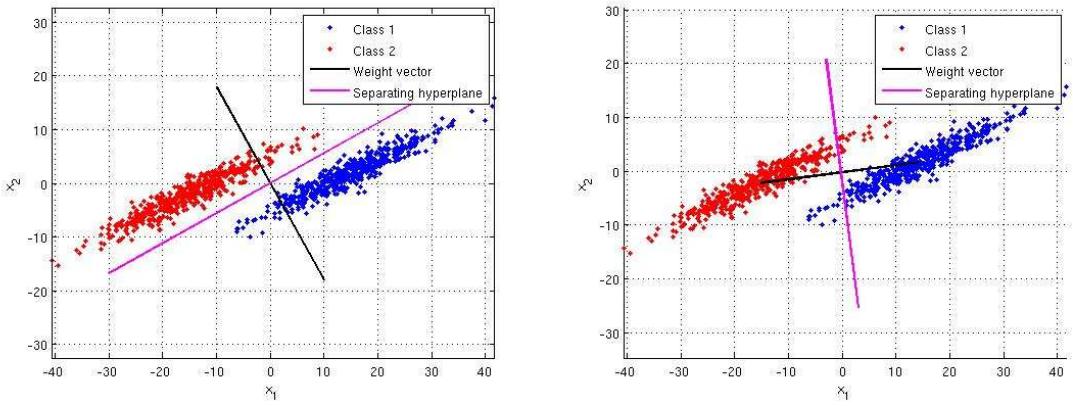
Figure 3: Classification results in 3D

	Fisher's Linear Discriminant	Difference-of-Means
# Test vectors	699	699
# Misclassifications	29	22
% Classification Accuracy	95.85	96.85

Table 1: Classification accuracy comparison in 2D

	Fisher's Linear Discriminant	Difference-of-Means
# Test vectors	699	699
# Misclassifications	24	13
% Classification Accuracy	96.57	98.14

Table 2: Classification accuracy comparison in 3D



(a) (b)

Figure 4: Classification results in 2D for Gaussian class distribution

	Fisher's Linear Discriminant	Difference-of-Means
# Test vectors	1000	1000
# Misclassifications	0	42
% Classification Accuracy	100	95.8

Table 3: Classification accuracy comparison in 2D for Gaussian class distribution

2 Problem 2

In this problem we had to experiment with designing a classifier using neural network and support vector machine approaches and then compare the two.

2.1 Methodology

We considered a 2-class classification problem and used the same two classes as in problem 1. For the neural network classification, we used the MATLAB Neural Network toolbox. In particular, we used 3 MATLAB functions: `newff`, `train` and `sim`, whose functionalities are briefly explained below:

- `newff` - create the neural network by specifying the number of hidden layers, number of neurons in each hidden layer and the transfer functions to be used in each layer. The number of neurons in the output layer is automatically determined from the target vector.
- `train` - train the neural network using the training data.
- `sim` - classify the testing data using the trained network.

In our experiments, we designed 3 neural networks. We call them NN_1, NN_2 and NN_3. We specified the number of layers as follows:

1. NN_1 had 1 hidden layer with 5 neurons and 1 output layer with 1 neuron.
2. NN_2 had 2 hidden layers with 5 neurons each and 1 output layer with 1 neuron.
3. NN_3 had 3 hidden layers with 5, 10 and 5 neurons respectively and 1 output layer with 1 neuron.

Note that each network has only 1 neuron in the output layer since we want the network output to be class labels which are single integers. The transfer function was chosen as arctan() for the hidden layers and linear for output layer. In our experiments, we were using the class labels '1' and '4'. Since the output of network was a real number for every test vector, we thresholded the output as:

$$Class\,Label = \begin{cases} 1 & network\,output \leqslant 2.5 \\ 4 & network\,output > 2.5 \end{cases}$$

Since our dataset has 16 features for the 2 classes, we tried to classify the data with the neural network approach by varying the number of features and evaluate which configuration of the network and how many features resulted is a good classifications accuracy.

For the SVM approach, we used the LIBSVM software v2.85 [3] which is a command line tool for SVM classification. The source code is available in C++ and Java and the software has interfaces for MATLAB, R, Python and so on. We used the C++ binaries for running our experiments. The software also comes with 'A Practical Guide to Support Vector Classification' which suggests some easy but significant steps to get good SVM classification on our data.

In order to understand some of the parameters used in our experiments, we delve briefly into the theory behind SVM. The goal of SVM is to produce a model which predicts target value of data instances in the testing set, when only the features are known. Given a training set of instance-label pairs $(\mathbf{x}_i, y_i), i = 1, \ldots, N$ where $\mathbf{x}_i \in R^n$ and $\mathbf{y} \in \{-1, 1\}^N$, the support vector machines (SVM) require the solution of the following optimization problem:

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{min} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ subject\,to \quad & y_i\left(\mathbf{w}^T\phi\left(x_i\right)+b\right) \geqslant 1-\xi_i \\ & \xi_i \geqslant 0 \end{aligned}$$

Here training vectors $\mathbf{x}_i$ are mapped into a higher (maybe infinite) dimensional space by the function ϕ. Then SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. $C > 0$ is the penalty parameter of the error term. Furthermore, $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ is called the kernel function.

The procedure suggested in this software was as follows:

1. Conduct simple scaling of the data. The main advantage is to avoid features in greater numeric ranges dominate those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Since all our feature values were integers in the range 0 to 100, we scaled that data to 0-1 range.
2. Consider the Radial Basis Function (RBF) kernel $K\left(\mathbf{x},\mathbf{y}\right) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}$
3. Use cross-validation to find the best parameters C and γ
4. Use the best parameters C and γ to train the whole training set
5. Test on the testing data

2.2 Results and Discussion

Table 4 summarizes the numerical classification accuracy percentages for the 3 neural networks when the number of features were varied from 1 to 16. In table 5, we present the results of SVM training and testing. The software finds the best values of C and γ and evaluates the 5-fold cross validation accuracy for training set classification. Then this (C, γ) pair is used on the testing data. Figure 5 depicts the performance comparison between the neural network and SVM approaches. From the figure, we can make several observations. First is that the data is not well separated in the first 2 dimensions because both neural network and SVM give $\sim$65% and $\sim$75% accuracy when using only the first or first two dimensions for classification. Starting from 3 features onwards, both types of classifiers give $\gtrsim$96% accuracy. So we do not need to use all 16 features for classification. The first 3-5 features should be sufficient. We also note that NN_1, NN_2 and SVM have relatively similar performances while NN_3 performs inferior to them. This suggests that increasing the number of hidden layers or the number of neurons in a hidden layer may not necessarily improve performance.

#Features	NN_1	NN_2	NN_3
1	63.23	62.52	61.66
2	75.54	75.68	74.96
3	97	98	95.57
4	95.71	97	95.71
5	99.71	99.57	89.41
6	97	99	98.43
7	99.86	99.57	97.42
8	98.43	99.57	98
9	99.71	98.86	98.43
10	100	99.86	75.82
11	99.86	99.28	98.86
12	97.85	96.71	99.86
13	99.71	97.85	98
14	99.86	98.14	97.85
15	99.86	99.86	97.57
16	99.28	99	98.57

Table 4: Classification accuracy percentages for Neural Network approach

#Features	Best C	Best γ	Cross Validation Accuracy (%)	Accuracy on test data (%)
1	2048	0.5	59.31	63.23
2	8	8	74.85	75.39
3	0.13	8	97.87	98.43
4	8192	0.13	99.6	97.57
5	2	0.01	99.87	99
6	0.03	8	99.87	99.57
7	0.03	8	99.93	99.57
8	0.03	8	100	99.57
9	0.03	8	99.93	99.57
10	0.5	8	100	99.71
11	0.13	2	100	99.86
12	0.03	2	100	99.86
13	0.03	2	100	99.86
14	0.03	2	100	99.86
15	0.13	0.01	99.93	97.14
16	0.13	0.01	99.93	97.57

Table 5: Classification accuracy for SVM approach

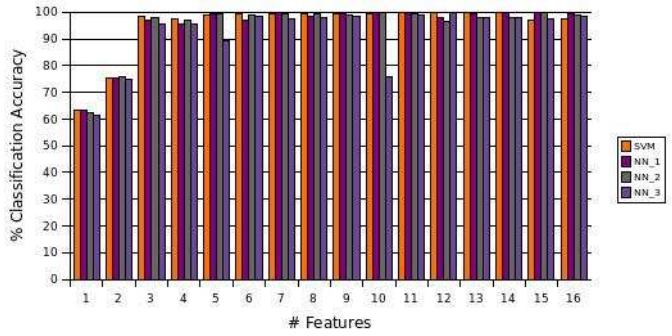


Figure 5: Classification Accuracy comparison for Neural networks and SVM

3 Problem 3

In this problem we had to design classifiers using Parzen window, K-nearest neighbour and nearest neighbour techniques and then compare the three approaches.

3.1 Methodology

For this problem, I implemented the classifiers in MATLAB. The dataset used was same as for problem 2. For Parzen window technique, I used a Gaussian kernel. The probability that a given test point $\mathbf{x}$ belongs to class C_i was calculated as:

$$p\left(\mathbf{x}|C_i\right) = \frac{1}{N^i}\sum_{j=1}^{N^i} K_{h_i}(\mathbf{x}-\mathbf{x}_j^i)$$

where $K(.)$ defines the Gaussian kernel: $K_\Sigma(\mathbf{x}-\mathbf{x}_j) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}_j)^T\Sigma^{-1}(\mathbf{x}-\mathbf{x}_j)\right)$.

The parameter h_i depends on the number of samples in class C_i and is given by $h_i = \frac{h}{\sqrt{N^i}}$ where N^i is the number of samples in C_i. h is a proportionality constant depending on the range of input data. Since our datapoints were in the range 0-100, we found out by simple calculation that h=1200 is a good value to use for our case. For the 2-class problem, we computed $p\left(\mathbf{x}|C_1\right)$ and $p\left(\mathbf{x}|C_2\right)$ and assigned the labels as:

$$Predicted\,Class\,Label\; = \begin{cases} 1 & p\left(\mathbf{x}|C_1\right) \geqslant p\left(\mathbf{x}|C_2\right) \\ 4 & p\left(\mathbf{x}|C_1\right) < p\left(\mathbf{x}|C_2\right) \end{cases}$$

The nearest neighbour technique was implemented as a special case of k-nearest neighbours technique for $k = 1$. For any test point, we considered its k nearest neighbours and depending on which class had majority of neighbouring points, that class label was assigned to the test point. Ties were avoided by choosing k to be odd. The performance of these classifiers was evaluated on the dataset by varying the number of features from 1 to 16 and recording the classification accuracy.

3.2 Results and Discussion

Table 6 gives the numerical classification results for Parzen window, Nearest neighbour and K-nearest neighbour methods where for the last method, values of k equal to 5 and 9 were used. Figure 6 presents the same results in graphical form for visual comparison. From the figure, we can deduce that for 3 or more features, the k-nearest neighbour technique has the best performance and

the nearest neighbour technique is closely second to it. Parzen window method consistently performs the lowest among the three methods. For 1 and 2 features, the two classes are highly overlapping and so it is difficult to assess the relative performance of the classifiers using only these two features.

Since we used density estimation for the Parzen window estimator, for demonstration purposes, we present in figure 7, the estimated densities for the two classes based only on feature 8. This feature was chosen simply to show how the density estimation leads to classification, because the two classes were well separated in this feature.

#Features	Parzen window	Nearest neighbour	KNN (k=5)	KNN (k=9)
1	56.94	51.93	51.93	51.65
2	58.94	63.66	71.24	73.25
3	94.13	97	97.28	96.85
4	93.42	97.71	97.28	97.14
5	97.14	99.57	99.57	99.43
6	96.71	99.57	99.57	99.28
7	97.71	99.57	99.57	99.57
8	97.57	99.57	99.57	99.43
9	98.14	99.57	99.86	99.86
10	98.43	99.57	99.71	99.57
11	98.43	99.57	99.57	99.57
12	98.43	99.57	99.71	99.71
13	98.71	99.57	99.71	99.71
14	98.43	99.57	99.71	99.71
15	99.28	99.71	99.71	99.71
16	99.71	99.71	99.86	99.86

Table 6: Classification accuracy percentages for non-parametric techniques

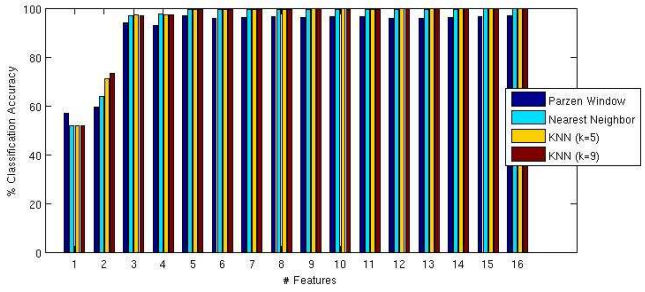


Figure 6: Classification accuracy comparison for Parzen window, nearest neighbour and k-nearest neighbour approaches

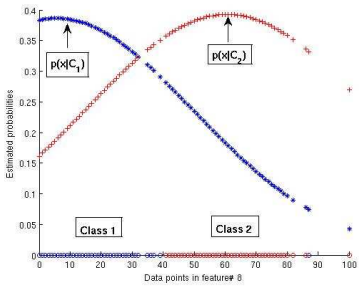


Figure 7: Density estimation on feature# 8 of the dataset

4 Comparing all the classifiers

In problem 1, we compared the performance of Fisher's linear discriminant and difference-of-means classifiers on 2- and 3-dimensional data which was approximately linearly separable. We performed more experiments to investigate how do the other classifiers perform on the same data.

4.1 Methodology

The different classifiers that were used for this experiment were Fisher's Linear Discriminant, Difference-of-Means, Neural Network (1 hidden layer with 5 neurons), SVM, Parzen Window, Nearest Neighbour and k-Nearest Neighbour (k=5,9). Table 7 presents the numerical classification accuracy percentages for these classifiers, on 2- and 3-dimensional datasets and figure 8 shows the same results in graphical form. For the sake of clarity on the figures, the classifier names have been abbreviated as $FLD, DoM, NNET, SVM, NN$,KNN_5 and KNN_9.

4.2 Results and Discussion

From the results, it is obviously clear that the classifiers resulting in non-linear classification boundaries (e.g. SVM, Neural network, Parzen window, KNN etc.) perform better than linear classifiers (Fisher's Linear discriminant and Difference-of-Means classifiers). For the 2-d dataset, SVM and neural network give the best accuracy followed by non-parametric techniques (KNN, Parzen window). On the other hand, for the 3-d dataset, K-nearest neighbour and SVM give practically the best accuracy and they are followed by nearest neighbour and neural network approaches.

Type of classifier	% classification accuracy (2-d data)	% classification accuracy (3-d data)
Fisher's Linear Discriminant	95.85	96.57
Difference-of-means	96.85	98.14
Neural network	98.14	98.14
SVM	98.99	99.33
Parzen window	96.28	97.85
Nearest neighbour	96.85	98.57
K-nearest neighbour (k=5)	97.85	99.57
K-nearest neighbour (k=9)	97.85	99.57

Table 7: All classifier performance on 2-d and 3-d datasets

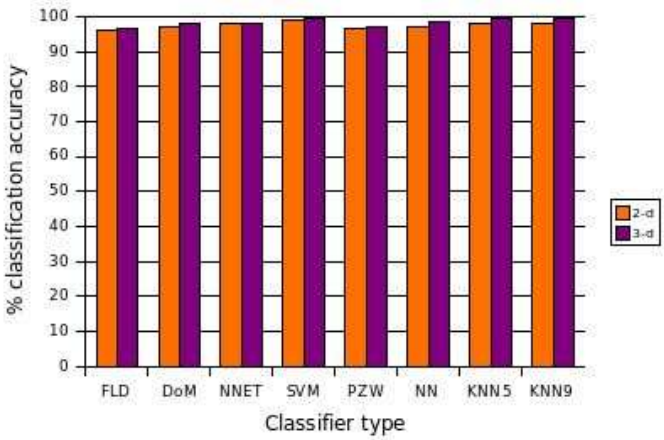


Figure 8: All classifier performance comparison for 2-d and 3-d datasets

References

[1] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*. [Online]. Available: http://www.csie.ntu.edu.tw/∼cjlin/libsvm

```
% ECE 662 HW2 Problem 1 : Linear discriminant classification for 2-d dataset
close all;

global training_data test_data

% Fisher's Linear Discriminant in 2D
class1_data = [training_data{1,1} training_data{1,8}];
class2_data = [training_data{4,1} training_data{4,8}];

figure;hold
on;plot(class1_data(:,1),class1_data(:,2),'.');plot(class2_data(:,1),class2_data(:
,2),'r.');
axis([-20 120 -20 120]);

m1=mean(class1_data);
m2=mean(class2_data);

S_W = zeros(2,2);

for i=1:size(class1_data)
    S_W = S_W + (class1_data(i,:)-m1)'*(class1_data(i,:)-m1);
end

for i=1:size(class2_data)
    S_W = S_W + (class2_data(i,:)-m2)'*(class2_data(i,:)-m2);
end

% determining optimal weight vector
w = inv(S_W)*((m1-m2))';

% plotting the weight vector
slope = w(2)/w(1);
x1=1.8;y1=slope*x1;
x2=-.2;y2=slope*x2;
plot([x1;x2],[y1;y2],'k-','LineWidth',2);

% for i=1:size(class1_data)
%     l = (class1_data(i,:)*w)/(w'*w);
%     plot(l*w(1)-10,l*w(2), 'o');
% end
%
% for i=1:size(class2_data)
%     l = (class2_data(i,:)*w)/(w'*w);
%     plot(l*w(1)-10,l*w(2), 'ro');
% end

N1=size(class1_data,1);
N2=size(class2_data,1);
N=N1+N2;

% m is the overall mean of the training data
m=(N1*m1+N2*m2)/N;

% This choice of w0 is obtained by posing the Fisher's Linear discriminant
% as a least squares problem.
w0=-w'*m';

% plotting the discriminating hyperplane
```

```
x1=-10;y1=-(w(1)*x1+w0)/w(2);
x2=110;y2=-(w(1)*x2+w0)/w(2);
plot([x1;x2],[y1;y2],'m-','LineWidth',2);

xlabel('x_1');
ylabel('x_2');
legend('Class 1','Class 2', 'Weight vector', 'Separating hyperplane');
grid on;

% testing the accuracy of the separating hyperplane on test data
testing_data = [[test_data{1,1} test_data{1,8}];
                [test_data{4,1} test_data{4,8}]];

N1_test = length(test_data{1,1});
N2_test = length(test_data{4,1});
true_class_label = [ones(N1_test,1);
                    2*ones(N2_test,1)];
estimated_class_label = zeros(N1_test+N2_test,1);

for i=1:N1_test+N2_test
    x = testing_data(i,:);
    if (w'*x'+w0) > 0
        estimated_class_label(i) = 1;
    else
        estimated_class_label(i) = 2;
    end
end

num_misclassifications = sum(true_class_label ~= estimated_class_label);
classification_accuracy = (1-(num_misclassifications/(N1_test+N2_test)))*100;

disp('Classification using FLD weight vector.');
disp(['# test vectors: ' num2str(N1_test+N2_test)]);
disp(['# misclassifications: ' num2str(num_misclassifications)]);
disp(['# classification accuracy: ' num2str(classification_accuracy)]);

% Starting from here, we investigate the effect of setting S_W to identity matrix
figure;
hold
on;plot(class1_data(:,1),class1_data(:,2),'.');plot(class2_data(:,1),class2_data(:
,2),'r.');
plot([m1(1);m2(1)],[m1(2);m2(2)],'k-','LineWidth',2);
axis([-20 120 -20 120]);

S_W = eye(2);

% determining optimal weight vector
w = inv(S_W)*(m1-m2)';

% This choice of w0 is obtained by posing the Fisher's Linear discriminant
% as a least squares problem.
w0=-w'*m';

% plotting the discriminating hyperplane
x1=-10;y1=-(w(1)*x1+w0)/w(2);
x2=110;y2=-(w(1)*x2+w0)/w(2);
plot([x1;x2],[y1;y2],'m-','LineWidth',2);
```

```
xlabel('x_1');
ylabel('x_2');
legend('Class 1','Class 2', 'Weight vector', 'Separating hyperplane');
grid on;

% testing the accuracy of the separating hyperplane on test data
testing_data = [[test_data{1,1} test_data{1,8}];
                [test_data{4,1} test_data{4,8}]];

N1_test = length(test_data{1,1});
N2_test = length(test_data{4,1});
true_class_label = [ones(N1_test,1);
                    2*ones(N2_test,1)];
estimated_class_label = zeros(N1_test+N2_test,1);

for i=1:N1_test+N2_test
    x = testing_data(i,:);
    if (w'*x'+w0) > 0
        estimated_class_label(i) = 1;
    else
        estimated_class_label(i) = 2;
    end
end

num_misclassifications = sum(true_class_label ~= estimated_class_label);
classification_accuracy = (1-(num_misclassifications/(N1_test+N2_test)))*100;

disp('Classification using difference-of-means weight vector.');
disp(['# test vectors: ' num2str(N1_test+N2_test)]);
disp(['# misclassifications: ' num2str(num_misclassifications)]);
disp(['# classification accuracy: ' num2str(classification_accuracy)]);
```

```
% ECE 662 HW2 Problem 1 : Linear discriminant classification for 3-d dataset
close all;

global training_data test_data

% Fisher's Linear Discriminant in 3D
class1_data = [training_data{1,1} training_data{1,8} training_data{1,15}];
class2_data = [training_data{4,1} training_data{4,8} training_data{4,15}];

figure;hold
on;plot3(class1_data(:,1),class1_data(:,2),class1_data(:,3),'.');plot3(class2_data
(:,1),class2_data(:,2),class2_data(:,3),'r.');

m1=mean(class1_data);
m2=mean(class2_data);

S_W = zeros(3,3);

for i=1:size(class1_data)
    S_W = S_W + (class1_data(i,:)-m1)'*(class1_data(i,:)-m1);
end

for i=1:size(class2_data)
    S_W = S_W + (class2_data(i,:)-m2)'*(class2_data(i,:)-m2);
end
```

```
% determining optimal weight vector
w = inv(S_W)*((m1-m2))';

% plotting the weight vector
point1 = (5e4)*w;
point2 = (-20e4)*w;
plot3([point1(1);point2(1)],[point1(2);point2(2)],[point1(3);point2(3)],'k-','Line
Width',2);grid on;

N1=size(class1_data,1);
N2=size(class2_data,1);
N=N1+N2;

% m is the overall mean of the training data
m=(N1*m1+N2*m2)/N;

% This choice of w0 is obtained by posing the Fisher's Linear discriminant
% as a least squares problem.
w0=-w'*m';

% plotting the discriminating hyperplane
[X,Y]=meshgrid([-5:60]);
Z=-(w(1)*X+w(2)*Y+w0)/w(3);
surf(X,Y,Z);
axis equal;

xlabel('x_1');
ylabel('x_2');
zlabel('x_3');
legend('Class 1','Class 2', 'Weight vector');
grid on;

% testing the accuracy of the separating hyperplane on test data
testing_data = [[test_data{1,1} test_data{1,8} test_data{1,15}];
                [test_data{4,1} test_data{4,8} test_data{4,15}]];

N1_test = length(test_data{1,1});
N2_test = length(test_data{4,1});
true_class_label = [ones(N1_test,1);
                    2*ones(N2_test,1)];
FLD_class_label = zeros(N1_test+N2_test,1);

for i=1:N1_test+N2_test
    x = testing_data(i,:);
    if (w'*x'+w0) > 0
        FLD_class_label(i) = 1;
    else
        FLD_class_label(i) = 2;
    end
end

num_misclassifications = sum(true_class_label ~= FLD_class_label);
classification_accuracy = (1-(num_misclassifications/(N1_test+N2_test)))*100;

disp('Classification using FLD weight vector.');
disp(['# test vectors: ' num2str(N1_test+N2_test)]);
disp(['# misclassifications: ' num2str(num_misclassifications)]);
```

```
disp(['# classification accuracy: ' num2str(classification_accuracy)]);

% Starting from here, we investigate the effect of setting S_W to identity matrix
figure;
hold
on;plot3(class1_data(:,1),class1_data(:,2),class1_data(:,3),'.');plot3(class2_data
(:,1),class2_data(:,2),class2_data(:,3),'r.');
plot3([m1(1);m2(1)],[m1(2);m2(2)],[m1(3);m2(3)],'k-','LineWidth',2);

S_W = eye(3);

% determining optimal weight vector
w = inv(S_W)*(m1-m2)';

% This choice of w0 is obtained by posing the Fisher's Linear discriminant
% as a least squares problem.
w0=-w'*m';

% plotting the discriminating hyperplane
[X,Y]=meshgrid([-5:60]);
Z=-(w(1)*X+w(2)*Y+w0)/w(3);
surf(X,Y,Z);
axis equal;

xlabel('x_1');
ylabel('x_2');
legend('Class 1','Class 2', 'Weight vector');
grid on;

% testing the accuracy of the separating hyperplane on test data
testing_data = [[test_data{1,1} test_data{1,8} test_data{1,15}];
                [test_data{4,1} test_data{4,8} test_data{4,15}]];

N1_test = length(test_data{1,1});
N2_test = length(test_data{4,1});
true_class_label = [ones(N1_test,1);
                    2*ones(N2_test,1)];
FLD_class_label = zeros(N1_test+N2_test,1);

for i=1:N1_test+N2_test
    x = testing_data(i,:);
    if (w'*x'+w0) > 0
        FLD_class_label(i) = 1;
    else
        FLD_class_label(i) = 2;
    end
end

num_misclassifications = sum(true_class_label ~= FLD_class_label);
classification_accuracy = (1-(num_misclassifications/(N1_test+N2_test)))*100;

disp('Classification using difference-of-means weight vector.');
disp(['# test vectors: ' num2str(N1_test+N2_test)]);
disp(['# misclassifications: ' num2str(num_misclassifications)]);
disp(['# classification accuracy: ' num2str(classification_accuracy)]);
```

```
% ECE 662 HW 2 Problem 2 : Neural network classification
```

```
class1 = 1;
class2 = 4;

N1_train = length(training_data{class1,1});
N2_train = length(training_data{class2,1});

N1_test = length(test_data{class1,1});
N2_test = length(test_data{class2,1});

neural_train_labels = [ones(N1_train,1);
                       4*ones(N2_train,1)];

neural_test_labels = [ones(N1_test,1);
                      4*ones(N2_test,1)];

fp=fopen('nn_results.txt','w');

for attribute=1:16

    neural_train_data=[];
    neural_test_data=[];

    % preparing training and test data
    for j=1:attribute
        neural_train_data = [neural_train_data
[training_data{class1,j};training_data{class2,j}]];
        neural_test_data = [neural_test_data
[test_data{class1,j};test_data{class2,j}]];
    end

    % creating, training and testing the network.
    net = newff(neural_train_data', neural_train_labels', 5);
    net = train(net, neural_train_data', neural_train_labels');
    net_output = sim(net, neural_test_data');

    % The network output needs to be threholded to give a 2-class
    % classification.
    index = find(net_output<=2.5);
    net_output(index) = 1;

    index = find(net_output>2.5);
    net_output(index) = 4;

    num_misclassifications = sum(neural_test_labels' ~= net_output);
    classification_accuracy = (1-(num_misclassifications/(N1_test+N2_test)))*100;
    disp(classification_accuracy);

    fprintf(fp,'%4.2f\n',classification_accuracy);

end

fclose(fp);
```

```
% ECE 662 HW 2 Problem 3 : Parzen Window Density Estimation

% We use Gaussian Parzen window.
```

```
ATTRIBUTES=16;

N1_train = length(training_data{1,1});
N2_train = length(training_data{4,1});
N1_test = length(test_data{1,1});
N2_test = length(test_data{4,1});

h=1200;
h_N1=h/sqrt(N1_train);
h_N2=h/sqrt(N2_train);

parzen_train_labels = [ones(N1_train,1);
                       4*ones(N2_train,1)];

parzen_test_labels = [ones(N1_test,1);
                      4*ones(N2_test,1)];

for attribute=1:ATTRIBUTES

    parzen_train_data=[];
    parzen_test_data=[];

    for j=1:attribute
        parzen_train_data = [parzen_train_data
[training_data{1,j};training_data{4,j}]];
        parzen_test_data =  [parzen_test_data [test_data{1,j};test_data{4,j}]];
    end

    predicted_test_labels=[];

    for i=1:N1_test+N2_test
        test_point = parzen_test_data(i,:);
        test_vector = ones(N1_train+N2_train,1) * test_point;

        diff_vector = parzen_train_data - test_vector;

        diff_vector(1:N1_train,:) = diff_vector(1:N1_train,:)/h_N1;
        diff_vector(N1_train+1:N1_train+N2_train,:) =
diff_vector(N1_train+1:N1_train+N2_train,:)/h_N2;

        if attribute == 1
            gaussian_pdf_values = normpdf(diff_vector);
        else
            gaussian_pdf_values = mvnpdf(diff_vector);
        end

        class1_probability = sum(gaussian_pdf_values(1:N1_train))/N1_train;
        class2_probability =
sum(gaussian_pdf_values(N1_train+1:N1_train+N2_train))/N2_train;

        if class1_probability >= class2_probability
            predicted_test_labels=[predicted_test_labels;1];
        else
            predicted_test_labels=[predicted_test_labels;4];
        end
    end
```

```
    num_misclassifications = sum(parzen_test_labels ~= predicted_test_labels);
    classification_accuracy = 100*(1 - num_misclassifications/(N1_test+N2_test));

    disp(['attributes=' num2str(attribute) ' <--> Classification accuracy='
num2str(classification_accuracy) '%']);
end
```

```
% ECE 662 HW 2 Problem 3 : Nearest neighbour and K-nearest neighbour
% algorithm implementation.

K=[1 3 5 7 9];

ATTRIBUTES=16;

N1_train = length(training_data{1,1});
N2_train = length(training_data{4,1});
N1_test = length(test_data{1,1});
N2_test = length(test_data{4,1});

knn_train_labels = [ones(N1_train,1);
                    4*ones(N2_train,1)];

knn_test_labels = [ones(N1_test,1);
                   4*ones(N2_test,1)];

fp = fopen('knn_results.txt','w');

for k=K

    fprintf(fp, 'k=%d\n\n',k);

    for attribute=1:ATTRIBUTES

        disp(['k=' num2str(k) '   attributes=' num2str(attribute)]);

        knn_train_data=[];
        knn_test_data=[];

        for j=1:attribute
            knn_train_data = [knn_train_data
[training_data{1,j};training_data{4,j}]];
            knn_test_data = [knn_test_data [test_data{1,j};test_data{4,j}]];
        end

        predicted_test_labels=[];

        for i=1:N1_test+N2_test
            test_point = knn_test_data(i,:);
            test_vector = ones(N1_train+N2_train,1) * test_point;

            diff_vector = knn_train_data - test_vector;
            euclidean_distance = sqrt(sum((diff_vector .* diff_vector),2));

            [sorted_distance, index]=sort(euclidean_distance);

            class1_score=0;
            class2_score=0;
```

```
            for m=1:k
                if knn_train_labels(index(m)) == 1
                    class1_score=class1_score+1;
                else
                    class2_score=class2_score+1;
                end
            end

            if class1_score > class2_score
                predicted_test_labels=[predicted_test_labels;1];
            else
                predicted_test_labels=[predicted_test_labels;4];
            end
        end

        num_misclassifications = sum(knn_test_labels ~= predicted_test_labels);
        classification_accuracy = 100*(1 -
num_misclassifications/(N1_test+N2_test));
        fprintf(fp, '%4.2f\n', classification_accuracy);
    end
    fprintf(fp, '\n\n');
end

fclose(fp);
```
