Purdue University
School of Electrical and Computer Engineering
Graduate School

Name
Collaborators
ECE 662: Homework #2
Prof. Boutin
April 15, 2008

1. Parametric method evaluation

1.1. For this part of the homework we executed the following steps:

1.1.1. Generate synthetic Gaussian data
1.1.2. 1,000 training samples, N_T
1.1.3. 9,000 test samples, N
1.1.4. We generated two different classes, w_1 and w_2

1.2. Computed the sample mean for both classes, μ_1 and μ_2

$$\mu_i = \frac{1}{N_T}\sum_{x \epsilon w_i} x \text{, where } i = 1,2 \text{ and x are the training samples}$$

1.2.1. Computed the with-in class scatter matrix S_w

$$S_i = \sum_{x \epsilon w_i} (x-\mu_i)(x-\mu_i)^t$$
$$S_w = S_1 + S_2$$

1.2.2. Computed the projection vectors, v_1 and v_2

$$v_1 = {S_w}^{-1} * (\mu_1 - \mu_2)$$
$$v_2 = \mu_1 - \mu_2$$

1.2.3. Projected the training samples using v_1 and v_2

$$y_i = {v_1}^t * w_i$$
$$z_i = {v_2}^t * w_i$$

1.2.4. Computed the histogram of the projected data

The histogram gives the distribution of the projected data. We used this distribution to compute the cut-off classification threshold.

1.2.5. Computed the cut-off threshold

First, We computed the difference of the histograms for each projected samples; $y_1 - y_2$ and $z_1 - z_2$. Then, we used changes in sign to determine where to place the cut-off threshold. We assumed a maximum of two changes in sign and at least one change in sign. With the computed threshold, we obtain a range of values, in which the samples that inside that range will be classified in class w_1, otherwise the samples will be classified in class w_2.

1.2.6. Classified the test samples and computed the error rate

As explained before, we have 9,000 test samples for each class. We classified the samples and verified which samples were misclassified.

$$error\ rate_{w_i} = \frac{\#\ of\ misclassified\ samples}{N}$$

We computed the error rate for each class, using two different classifiers. For classifier C_{S_w} we compute the with-in class scatter matrix S_w as shown in section 1.2.1. The other classifier is C_I; for this classifier we set $S_w = I$. When we talk about $S_w \neq I$, we are saying that S_w where computed as shown in section 1.2.1. We repeated steps 1.1.1 to 1.2.6 for 8 different data sets,

1.3. Results

At each data set subsection we give a brief analysis of the results, and then at the end of the results section we discuss the error rates and compare all experiments.

1.3.1. Data set #1

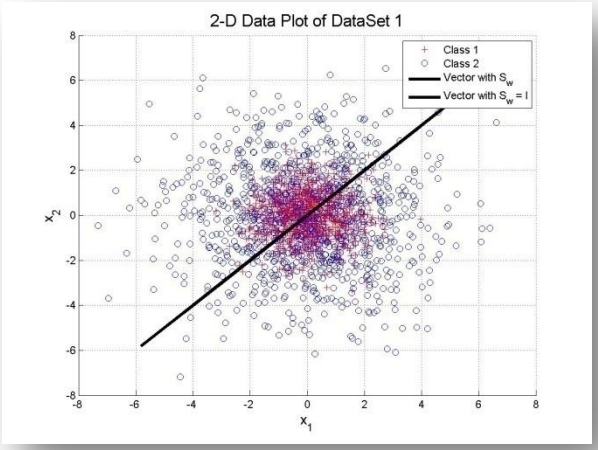

Figure 1

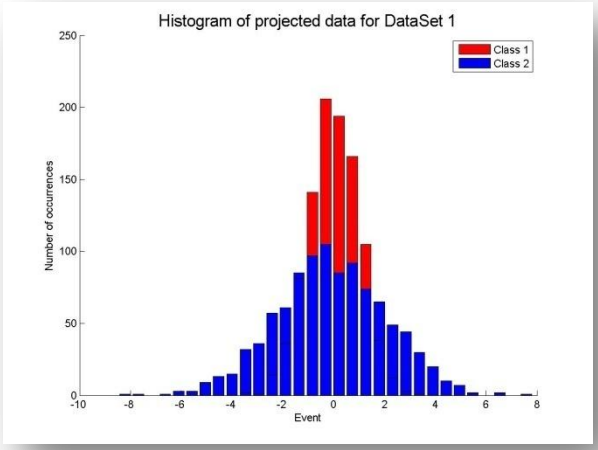

Figure 2

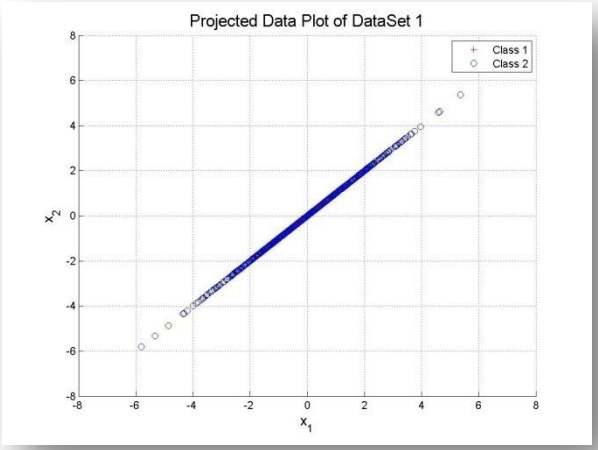

Figure 3

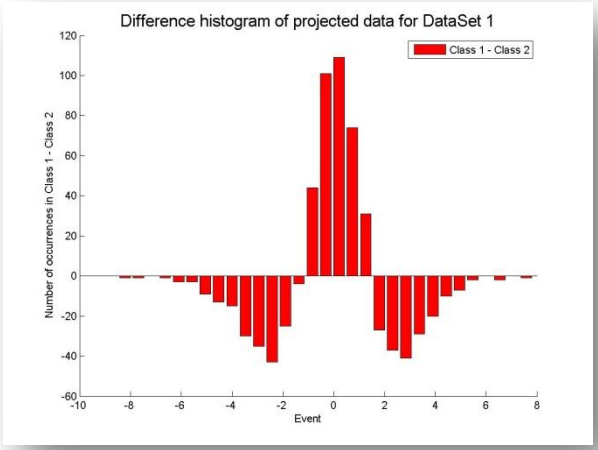

Figure 4

The first data set was used to show the behavior of the parametric method with highly correlated data. You can observe, see Figure 1 and Figure 2, that class 1 is really dense in the mean. On the other hand class 2 has a higher variance. In this

case, we can discriminate in the data using two thresholds, see Figure 4. However, you can observe in Figure 3 that the projection did not result in a separation of the classes. Moreover, there were no significant difference between the projections lines for C_{S_w} and C_I, see Figure 1. We just concluded that the parametric method is not good for very highly correlated data.

1.3.2. Data set #2

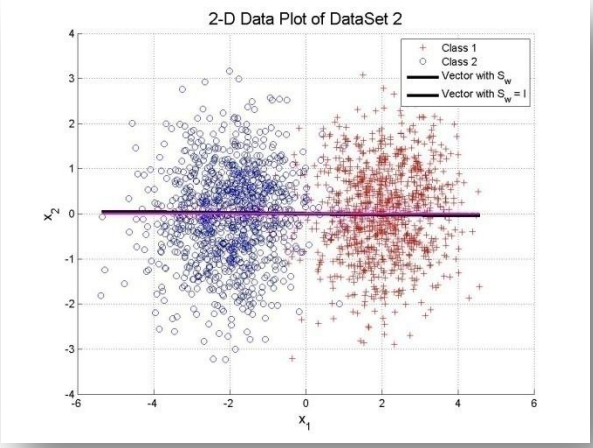

Figure 5

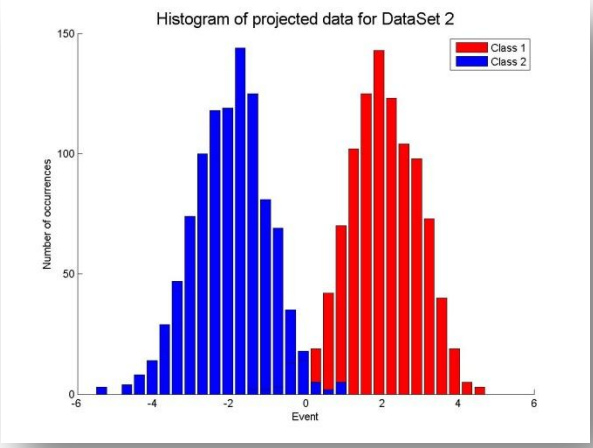

Figure 6

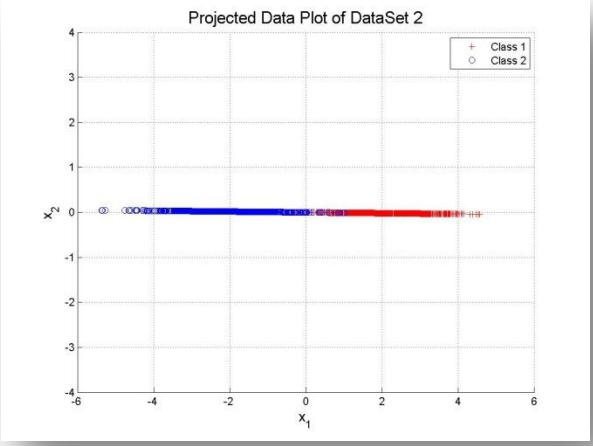

Figure 7

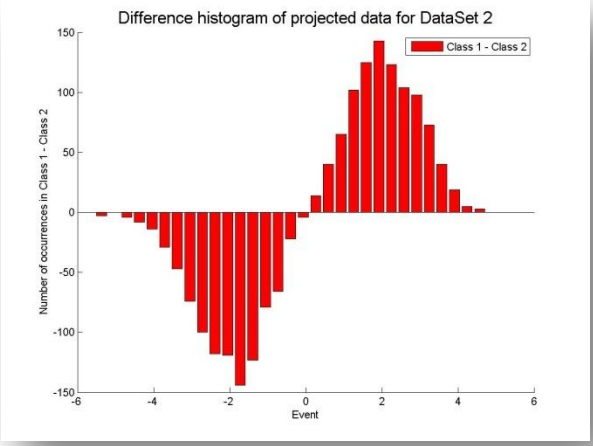

Figure 8

For this data set we decreased the correlation between classes, see Figure 5 and Figure 6. Now you can see well separated projected samples in Figure 7. However, there were no difference between the projections lines for C_{S_w} and C_I, see Figure 5.

1.3.3. Data set #3

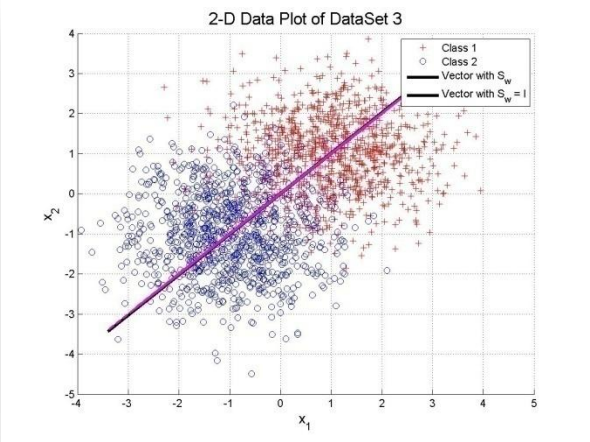

Figure 9

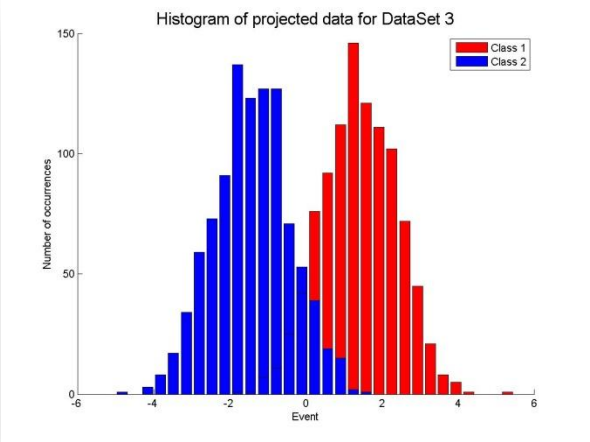

Figure 10

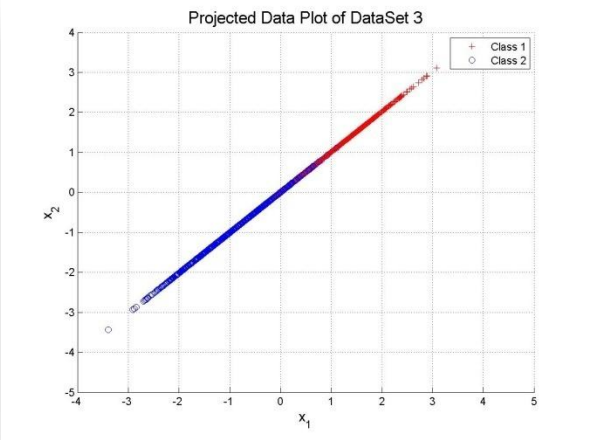

Figure 11

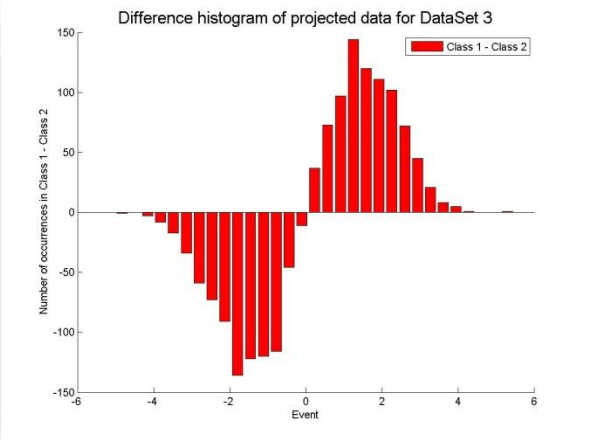

Figure 12

For data set #3 we just rotated the data 45 degrees. The correlation was similar to that in data set #2. We can observe that there is no significant difference between the projections lines for C_{S_w} and C_I, see Figure 9.

1.3.4. Data set #4

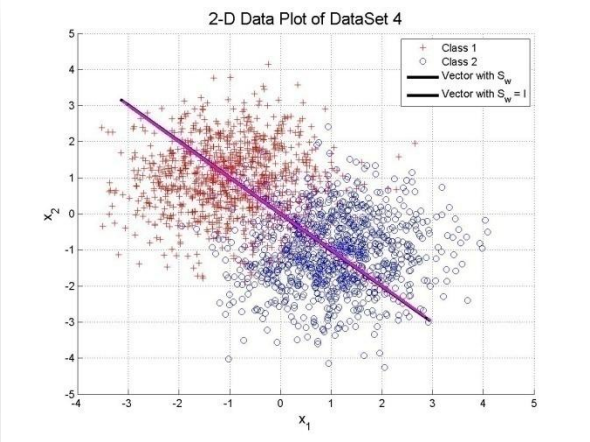

Figure 13

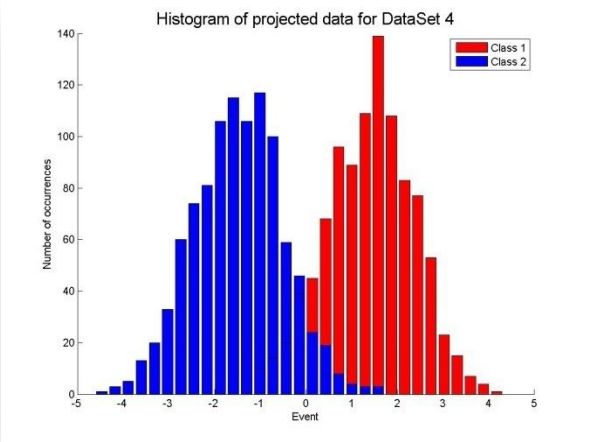

Figure 14

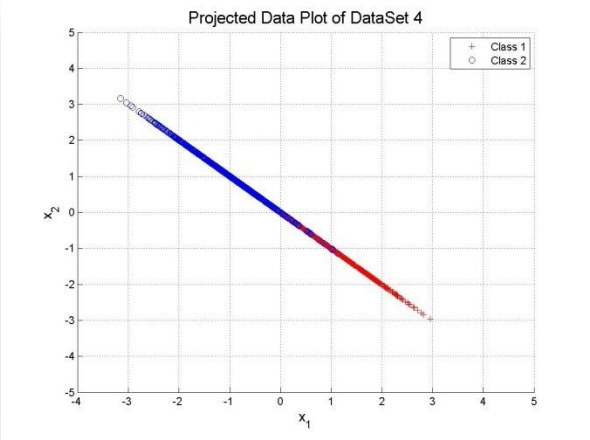

Figure 15

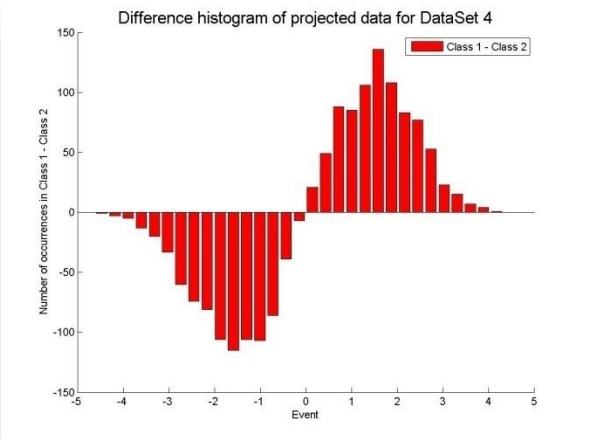

Figure 16

For data set #4, we rotated the data -45 degrees. The statistical properties of the data were similar to the previous two data sets. The outcome of the experiment was very similar to our previous results. The separation of the projected data was acceptable and there is no significant difference between the projections lines for C_{S_w} and C_I, see Figure 13.

For the next data sets we used our intuition to try different data sets that should create a difference between the projections lines for C_{S_w} and C_I.

1.3.5. Data set #5

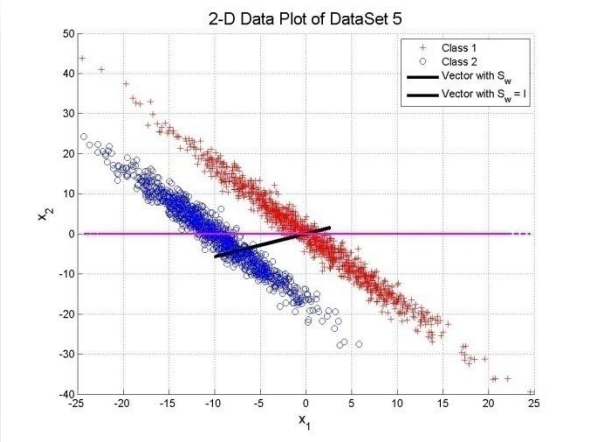

Figure 17

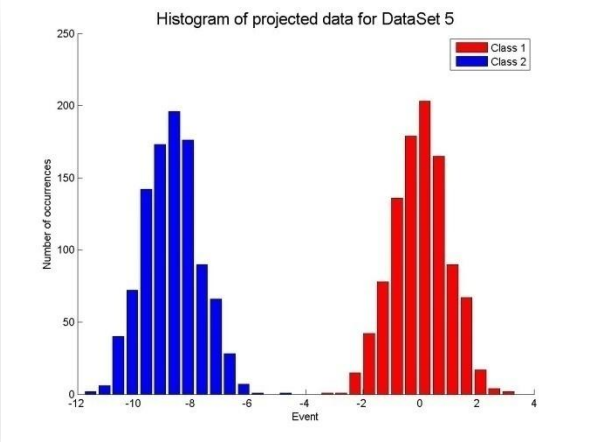

Figure 18

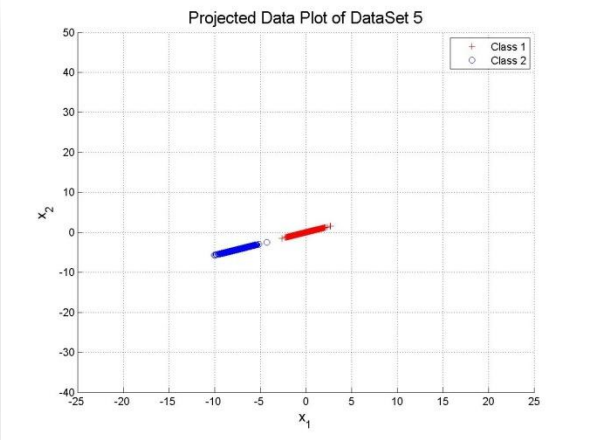

Figure 19

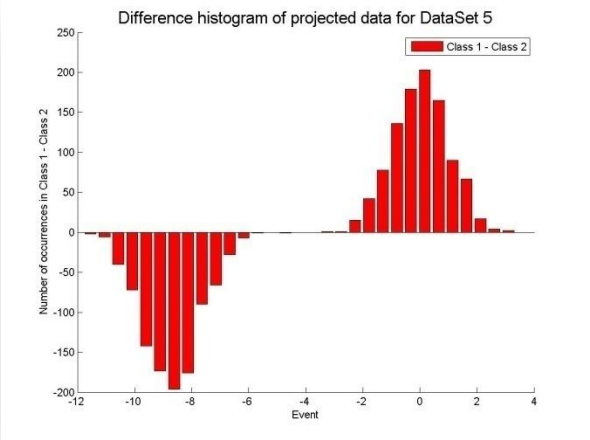

Figure 20

From the computed data we know that the between-class scatter matrix S_B measures the distance between the classes' mean. On the other hand, the within-class scatter matrix S_w takes in consideration the distribution of the data. Therefore, when we set $S_w = I$ we are only taking in account the distance between the means and when we compute the projection axis for $S_w \neq I$ we take in consideration the distance of the means and the variance of the data.

Let define two variables which are different distances between the classes. ξ_1 is the distance between classes w_1 and w_2 parallel to the projection axis when $S_w \neq I$. ξ_2 is the distance between classes means μ_1 and μ_2, which are parallel to the projection axis when $S_w = I$. Consequently, we expected that a data set with $\xi_1 \ll \xi_2$ should result in

different projection vectors. We can create such data, if the distribution of the samples has a high variance orthogonal to the projection axis when $S_w \neq I$, see Figure 17. As expected, we got a significant difference between the projection vectors. As Figure 19 shows, the optimal projection vector provides well separated classes. Figure 18 shows the projected data, and Figure 19 and Figure 20 sow the histogram of the projected data. We can observe that easily we can select a threshold that should effectively differentiate between classes.

1.3.6. Data set #6

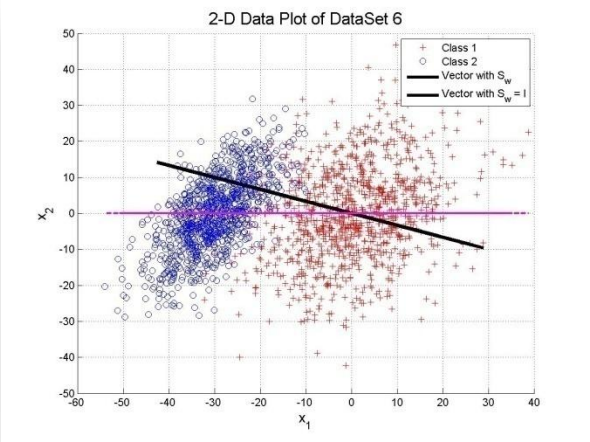

Figure 21

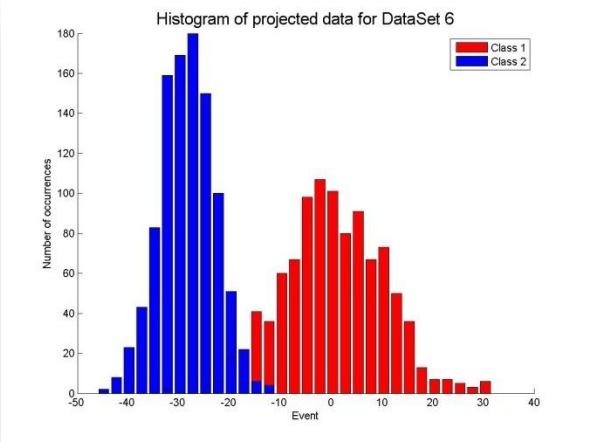

Figure 22

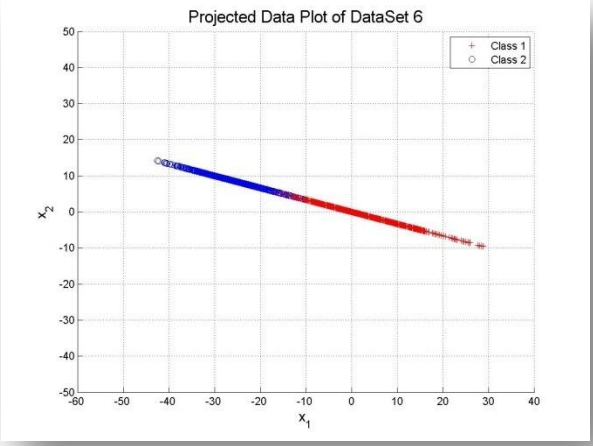

Figure 23

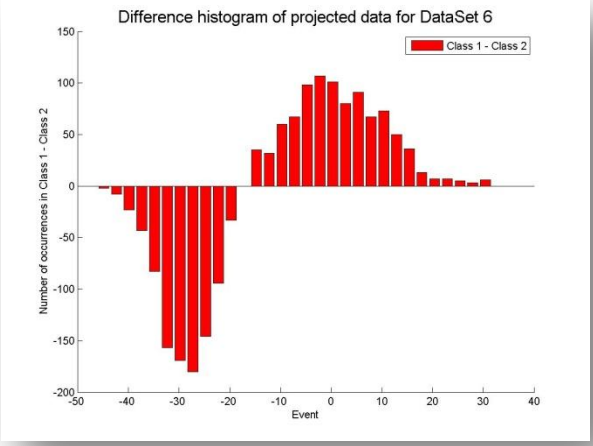

Figure 24

For this data set we changed the distribution of the data. The most significant changes are the increment in the correlation between the classes and the difference between the classes variance. As with data set #5, we got a significant difference between the projected vectors. However, because the data were more correlated we get some

overlapping of samples in the projection. Nevertheless, the results, are better than we set up $S_w = I$.

1.3.7. Data set #7

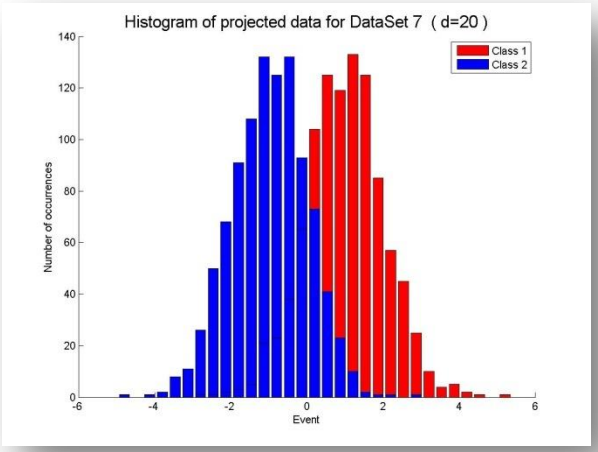

Figure 25

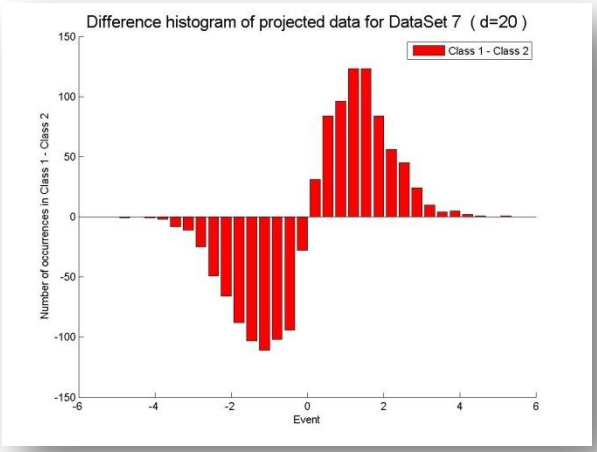

Figure 26

1.3.8. Data set #8

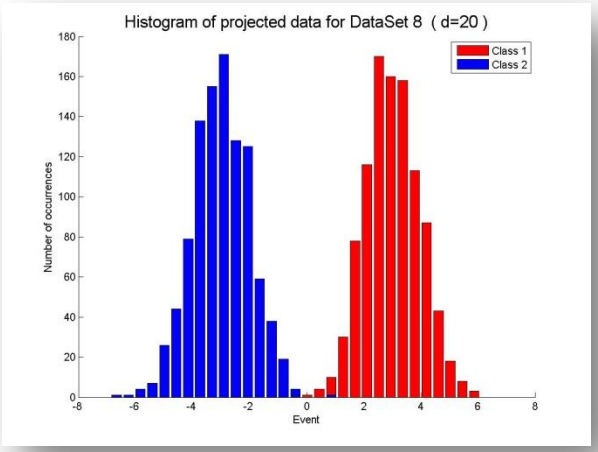

Figure 27

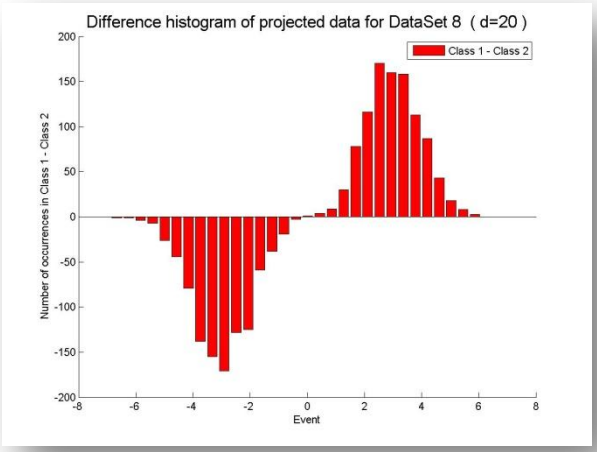

Figure 28

For data sets #7 and #8 we went to higher feature dimensions in order to see how the system will behave. Our results were very similar to our previous experiments.

1.3.9. Error rate results

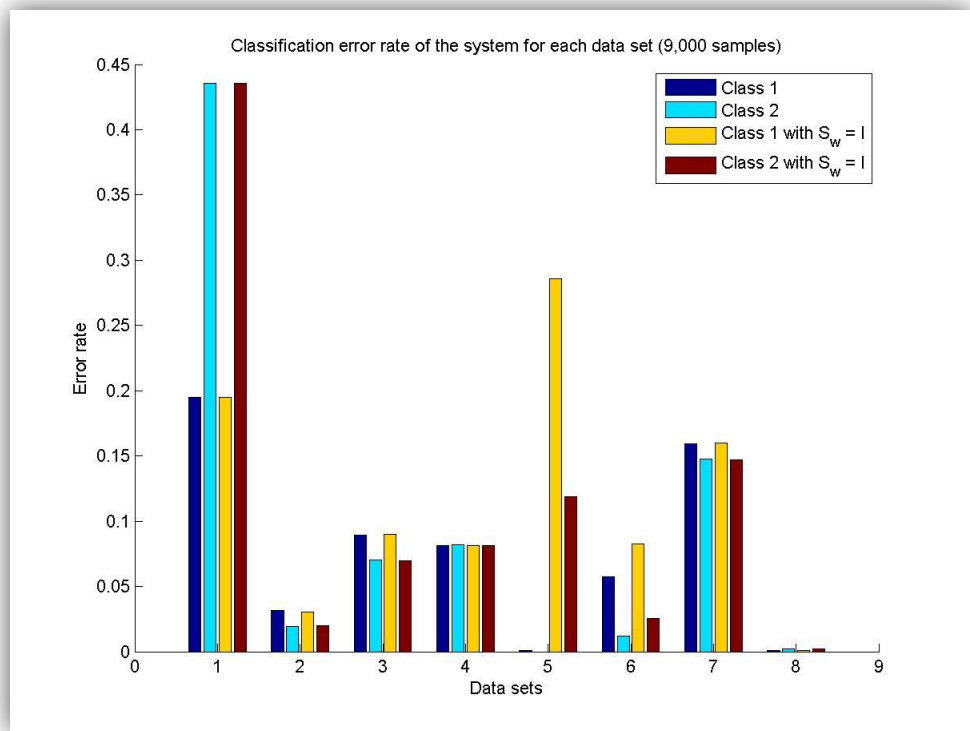

Figure 29

1.4. Analysis and discussion

As many other pattern recognition strategies, the results of the parametric method if $S_w \neq I$ or $S_w = I$ depend on the properties of the data we are categorizing. From our experiments results, the error rates help us to understand the effects of making $S_w = I$. As explained in section 1.3.5, the projections vector will be different when $\xi_1 \ll \xi_2$. As we can see in Figure 29, for data sets 1, 2, 3, 4, 6, and 7 $\xi_1 \approx \xi_2$. Therefore, the error rates are the same no matter what projection vector we use. However, when we made $\xi_1 \ll \xi_2$, in data sets 5 and 6 we got different results for each projection vector. In addition we can see that we got better results for the classifier that has $S_w \neq I$.

Our conclusion is that a parametric method with a classifier C_I will be less effective than a parametric method with classifier C_{S_w}, if the vector that connects the data means is not parallel to the optimal projection vector. We can pre-process the data to decide which classifier to use. However, a classifier with $S_w \neq I$ is the safe way to go.

1.5. Matlab's code

1.5.1. Code for classes with a feature size of two

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECE662 - Prof. Boutin        %
% Homework #2 - Problem 1      %
% March 23, 2008               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%clear all variables and close all windows
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Run properties
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%data filename
filename = 'DataSet';
fnum = 1;
%dimensions
d = 2;
%training samples
Nt = 10^3;
%number of bins for histograms
bins = 30;
%number of data sets
ND = 6;
%variable to store error rate of system for each data set
error = zeros(4, ND);


%loop for each data set
for dataCount = 1:ND

fnum = dataCount;

display '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
sprintf('%s%s%d%s', 'Running ', filename, fnum, '.mat')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data = load(sprintf('%s%s%d%s', 'data/', filename, fnum, '.mat'));
class1 = data.Class1(:, 1:Nt);
class2 = data.Class2(:, 1:Nt);
mu = data.m1 - data.m2;
Sw = data.Sw;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Project samples to straigth line for vector 1 with Sw
```

```
w1 = inv(Sw)*mu;
if(mu == 0)
   w1 = [0.001; 0.001];
end
w1 = w1/norm(w1);
y1 = w1'*class1;
y2 = w1'*class2;
minY = min([y1, y2]);
maxY = max([y1, y2]);
stepY = (maxY-minY)/(bins);
angleY = atan(w1(2)/w1(1));
wx1 = cos(angleY).*[min(y1, y2); max(y1, y2)];
wy1 = sin(angleY).*[min(y1, y2); max(y1, y2)];
%histograms for Sw
x1 = minY:stepY:maxY;
ny1 = hist(y1, x1);
ny2 = hist(y2, x1);
%plot histogram for projected data with Sw
f = figure;
hold on
bar(x1, ny1, 'r');
bar(x1, ny2, 'b');
legend('Class 1', 'Class 2');
xlabel('Event');
ylabel('Number of occurrences');
title(sprintf('%s%s %d', 'Histogram of projected data for ', filename, fnum), 'FontSize', 16);
hold off
saveas(f, sprintf('%s%s%d%s', 'img/', 'histo_', fnum, '.jpg'), 'jpg');
close(f);
%plot histogram difference for projected data with Sw
f=figure;
hold on
bar(x1, ny1-ny2, 'r');
legend('Class 1 - Class 2');
xlabel('Event');
ylabel('Number of occurrences in Class 1 - Class 2');
title(sprintf('%s%s %d', 'Difference histogram of projected data for ', filename, fnum), 'FontSize', 16);
hold off
saveas(f, sprintf('%s%s%d%s', 'img/', 'histoDiff_', fnum, '.jpg'), 'jpg');
close(f);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%% Projections with Sw = I
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
w2 = mu/norm(mu);
if(mu == 0)
   w2 = [0.001; 0.001];
end
z1 = w2'*class1;
z2 = w2'*class2;
```

```
minZ = min([z1, z2]);
maxZ = max([z1, z2]);
stepZ = (maxZ-minZ)/(bins);
angleZ = atan(w2(2)/w2(1));
wx2 = cos(angleZ).*[min(z1, z2); max(z1, z2)];
wy2 = sin(angleZ).*[min(z1, z2); max(z1, z2)];
%histograms for Sw = I
x2 = minZ:stepZ:maxZ;
nz1 = hist(z1, x2);
nz2 = hist(z2, x2);
%plot histogram for projected data with Sw = I
figure
hold on
bar(x2, nz1, 'r');
bar(x2, nz2, 'b');
legend('Class 1', 'Class 2');
xlabel('Event');
ylabel('Number of occurrences');
title(sprintf('%s%s %d', 'Histogram of projected data (S_w = I) for ',...
   filename, fnum), 'FontSize', 16);
hold off
saveas(f, sprintf('%s%s%d%s', 'img/', 'HistoProjSwI_', fnum, '.jpg'), 'jpg');
close(f);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% Plot 2D data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
f = figure('Position', [1000, 300, 600, 600]);
hold on
plot(class1(1, :), class1(2, :), '+r');
plot(class2(1, :), class2(2, :), 'ob');
plot(wx1, wy1, '-k', 'LineWidth', 3);
plot(wx2, wy2, '--m', 'LineWidth', 2);
legend('Class 1', 'Class 2', 'Vector with S_w', 'Vector with S_w = I');
xlabel('x_1', 'FontSize', 14);
ylabel('x_2', 'FontSize', 14);
title(sprintf('%s%s %d', '2-D Data Plot of ', filename, fnum), ...
   'FontSize', 16);
xLimit = xlim;
yLimit = ylim;
grid on;
hold off
saveas(f, sprintf('%s%s%d%s', 'img/', 'data_', fnum, '.jpg'), 'jpg');
close(f);
%plot projected samples
f=figure('Position', [1000, 300, 600, 600]);
hold on
plot(cos(angleY).*y1, sin(angleY).*y1, '+r');
plot(cos(angleY).*y2, sin(angleY).*y2, 'ob');
xlabel('x_1', 'FontSize', 14);
```

```
ylabel('x_2', 'FontSize', 14);
legend('Class 1', 'Class 2');
title(sprintf('%s%s %d', 'Projected Data Plot of ', filename, fnum), 'FontSize', 16);
xlim(xLimit);
ylim(yLimit);
grid on;
hold off;
saveas(f, sprintf('%s%s%d%s', 'img/', 'ProjData_', fnum, '.jpg'), 'jpg');
close(f);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%% Compute thresholds for y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%compute discrimination threshold
len = length(ny1);
th = len.*ones(1, 2);
count = 0;
%check polarity of first bin in difference histogram
polar = 0;
if( (ny1(1) - ny2(1)) <= 0 )
    polar = 0;
else
    polar = 1;
    th(1) = 1;
    count = 1;
end
%loop to get thresholds
for i=2:len
  if( ((ny1(i) - ny2(i)) < 0) && (polar == 1) )
     count = count + 1;
     if ( count > 2 )
        display 'More than two thresholds'
        break;
     end
     th(count) = i - 1;
     polar = 0;
  elseif( ((ny1(i) - ny2(i)) > 0) && (polar == 0) )
     count = count + 1;
     if ( count > 2 )
        display 'More than two thresholds'
        break;
     end
     th(count) = i;
     polar = 1;
  end
end

%compute error rate of classification algorithm
expC1 = w1'*data.Class1(:, Nt+1:end);
expC2 = w1'*data.Class2(:, Nt+1:end);
```

```
len = length(expC1);
errorY1 = 0;
errorY2 = 0;
LB = x1(th(1)) - (x1(2)-x1(1))/2;
UB = x1(th(2)) + (x1(2)-x1(1))/2;
for i=1:len
   if( not( ( (LB <= expC1(i)) && (expC1(i) <= UB) ) ) )
      errorY1 = errorY1 + 1;
   end
   if( (LB <= expC2(i)) && (expC2(i) <= UB) )
      errorY2 = errorY2 + 1;
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Threshold for projected data with Sw = I     %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%compute discrimination threshold
len = length(nz1);
th = len.*ones(1, 2);
count = 0;
%check polarity of first bin in difference histogram
polar = 0;
if( (nz1(1) - nz2(1)) <= 0 )
   polar = 0;
else
   polar = 1;
   th(1) = 1;
   count = 1;
end
%loop to get thresholds
for i=2:len
  if( ((nz1(i) - nz2(i)) < 0) && (polar == 1) )
     count = count + 1;
     if ( count > 2 )
        display 'More than two thresholds'
        break;
     end
     th(count) = i - 1;
     polar = 0;
  elseif( ((nz1(i) - nz2(i)) > 0) && (polar == 0) )
     count = count + 1;
     if ( count > 2 )
        display 'More than two thresholds'
        break;
     end
     th(count) = i;
     polar = 1;
  end
end

%compute error rate of classification algorithm
```

```
expC1 = w2'*data.Class1(1:d, Nt+1:end);
expC2 = w2'*data.Class2(1:d, Nt+1:end);
len = length(expC1);
errorZ1 = 0;
errorZ2 = 0;
LB = x2(th(1)) - (x2(2)-x2(1))/2;
UB = x2(th(2)) + (x2(2)-x2(1))/2;
for i=1:len
   if( not( ( (LB <= expC1(i)) && (expC1(i) <= UB) ) ) )
      errorZ1 = errorZ1 + 1;
   end
   if( (LB <= expC2(i)) && (expC2(i) <= UB) )
      errorZ2 = errorZ2 + 1;
   end
end

%compute error rate
error(:, dataCount) = [ errorY1;
                  errorY2;
                  errorZ1;
                  errorZ2]/len;

end

%plot error rate of each data set
f=figure;
hold on
bar(1:ND, error')
legend('Class 1', 'Class 2', 'Class 1 with S_w = I', 'Class 2 with S_w = I');
title('Classification error rate of the system for each data set (9,000 samples)');
xlabel('Data sets');
ylabel('Error rate');
hold off;
saveas(f, sprintf('%s%s%s', 'img/', 'error', '.jpg'), 'jpg');
```

1.1.1. For the classes with features vector greater than two we used a similar code. The only thing we removed was the plot for 2-D data. We found that was a waste of space and paper to add the code here. If you need this code, let me know.

1.1.2. Data generator code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECE 662 Homework #1         %
% Generates Normal Data       %
% Professor Boutin          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%clear all data
clear all;
```

```
close all;

%Procedure to generate random data

%data size, number of samples
N = 10^4;
%size of feature vector
d = 2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%mean of each feature
mu1 = zeros(d, 1); mu1(1) = 0; mu1(2) = 0;
mu2 = -mu1; mu2(1) = -30; mu2(2) = 0;
%variance of each mean
sd1 = 1 * ones(1, d); sd1(1) = 100; sd1(2) = 200;
sd2 = 1 * sd1;  sd2(1) = 25; sd2(2) = 150;
%generate data
DataGen(N, d, mu1, mu2, sd1, sd2, 'DataSet6', -30*pi/180, -30*pi/180);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECE 662 Homework #1          %
% Generates Normal Data        %
% Professor Boutin             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%N - number of samples
%d - size of feature vector
%m1 - mean vector
%sd - variance vector
%fileName - data is stored in a file with this name
function DataGen(N, d, m1, m2, sd1, sd2, fileName, z1, z2)

%reserve memory for variable
Class1 = zeros(d, N);
Class2 = zeros(d, N);
%generate normalize data
for i = 1:d
   for j=1:N
      Class1(i, j) = m1(i) + sqrt(sd1(i))*randn(1);
      Class2(i, j) = m2(i) + sqrt(sd2(i))*randn(1);
   end
end

%Rotate classes data
if( (m2(1) ~= 0) || (m2(2) ~= 0) )
   Class2 = Class2 - m2*ones(1,N);
end
Class1 = [cos(z1), -sin(z1); sin(z1), cos(z1)]*Class1;
```

```
Class2 = [cos(z2), -sin(z2); sin(z2), cos(z2)]*Class2;
if( (m2(1) ~= 0) || (m2(2) ~= 0) )
   Class2 = Class2 + m2*ones(1,N);
end

%Compute scatter matrices
S1 =(Class1-m1*ones(1,N))*(Class1-m1*ones(1,N))';
S2 =(Class2-m2*ones(1,N))*(Class2-m2*ones(1,N))';
Sw = S1 + S2;
Sb = (m1 - m2)*(m1- m2)';

%save
save(sprintf('%s%s%s', 'data/', fileName, '.mat'), 'Class1', 'Class2', 'm1', 'm2', 'S1', 'S2', 'Sw', 'Sb');
sprintf('%s%s%s', 'Data ', fileName, ' generated.')
```

2. Neural networks and Support Vector Machines

2.1. Experiment description

2.1.1. We used 40 black and white image samples shown in Figure 45. The image samples consisted of the letter A and R typed in different computer typefaces. With the exception of two four image samples that were handwritten.

2.1.2. We repeated each experiment for different training set sizes (6, 10, and 14 samples for each letter). The remaining samples were used for testing.

2.2. Procedure

2.2.1. The first step is to pre-process the input. Each image was cropped as much as possible. Then, we resized all images to the same width and height (50 x 50 pixels).

2.2.2. After the pre-process, we used the raw image input for one of the Neural Network experiments. For the other experiments we extracted a set of features explained in the section below.

2.3. Feature extraction

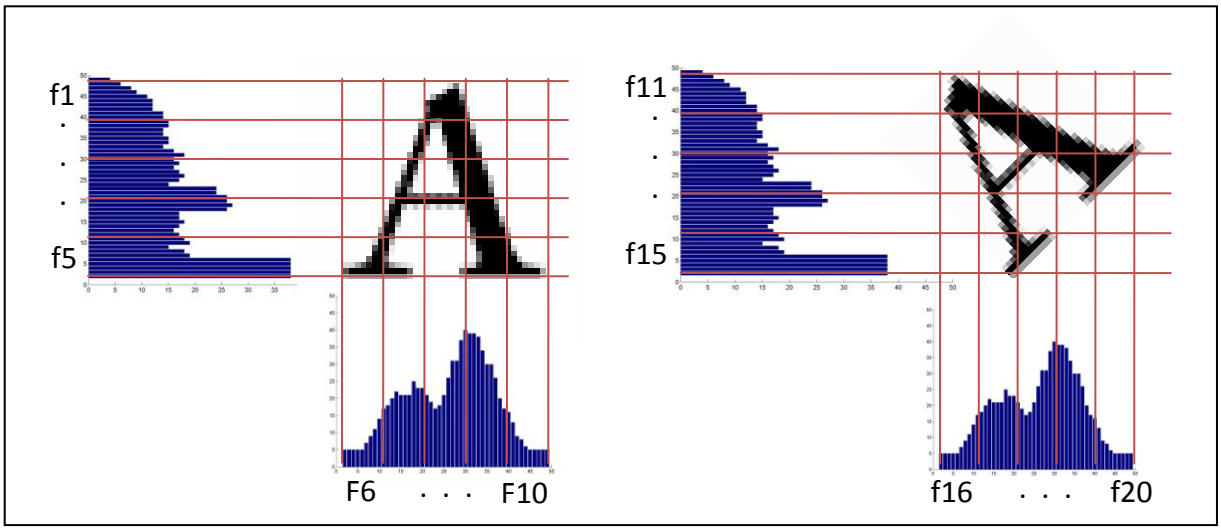

Figure 30

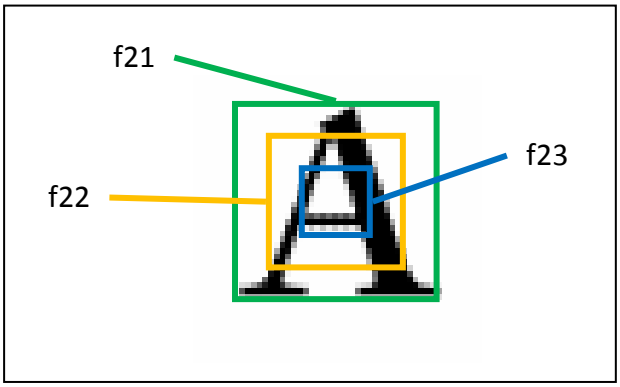

Figure 31

2.3.1. Figure 30 and Figure 31 show how we computed the features. For the first ten features we computed histogram of the number of pixels with a zero (black pixels) for each row and columns. Then, we partition the each histogram in five equally spaced regions and each feature contained the area under the curve of the histogram for its respective region. For features 11 to 20, we rotated the image 45 degrees counter-clockwise, and then followed the same procedure discussed above. The last three features, Figure 31, are the number of pixels with zero inside each frame respectively.

$f21 = balck\ pixels\ in\ blue\ square$
$f22 = black\ pixels\ in\ yellow\ square - f21$
$f23 = black\ pixels\ in\ \ green\ square - f22 - f21$

2.4. The Neural Networks experiments

2.4.1. For the NN experiments we made three different systems. One received the image raw data as input (each pixel in the image is an input) and with a five digit binary output. The second experiment was using our 23 features set and five digits binary output. The third experiment used the features, but the output was a single integer. All systems had two hidden layers. We used Matlab's NN algorithms to create and train the networks.

2.5. Support vector machine

2.5.1. For the SVM experiments we used Matlab's SVM functions to create and train the systems. The SVM were based on three different kernel functions— liner, quadratic, polynomial, and Gaussian Radial Basis.

2.6. Results and analysis

2.6.1. NN training examples

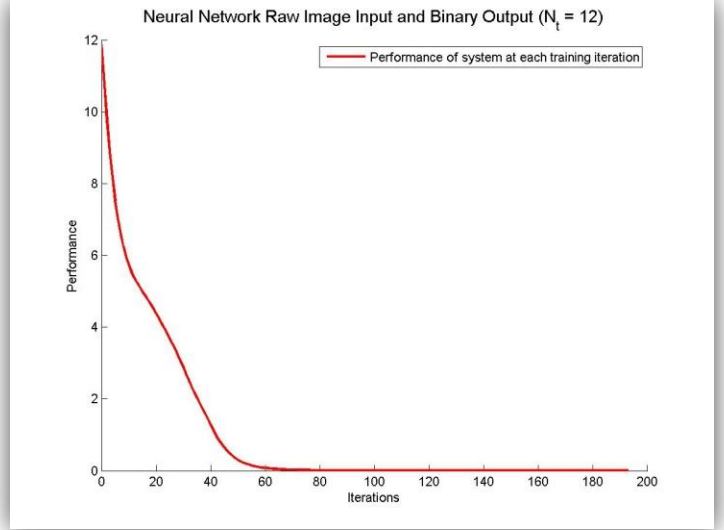

Figure 32

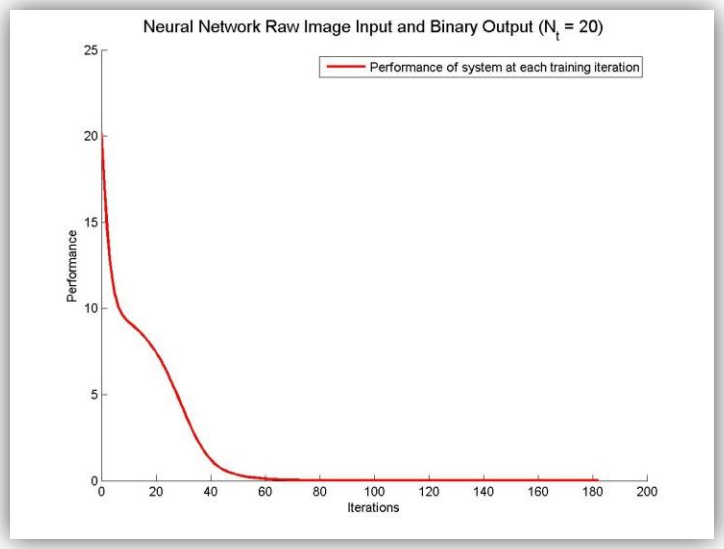

Figure 33

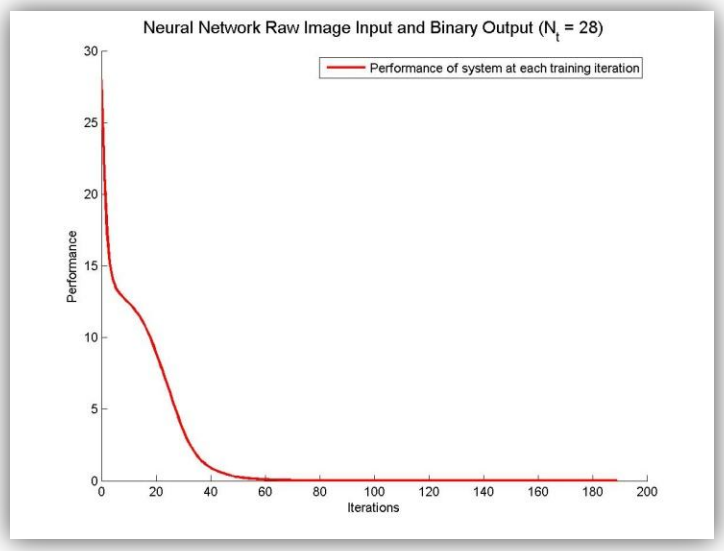

Figure 34

Notice that the number of training samples N_t is 12, 20, and 28. This is because is the sum of A and R letters and we have 6, 10, and 14 training samples for each letter. From these results I would like to highlight the similarities between Figure 33 and Figure 34. With NN sometimes adding more training samples is not necessary. It can lead to overfitting the network. For these systems, we get to our goal of 0.001 error rate in less than 60 interations.

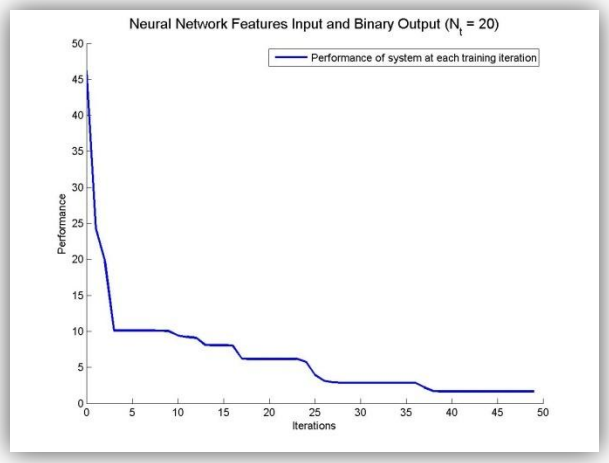

Figure 35

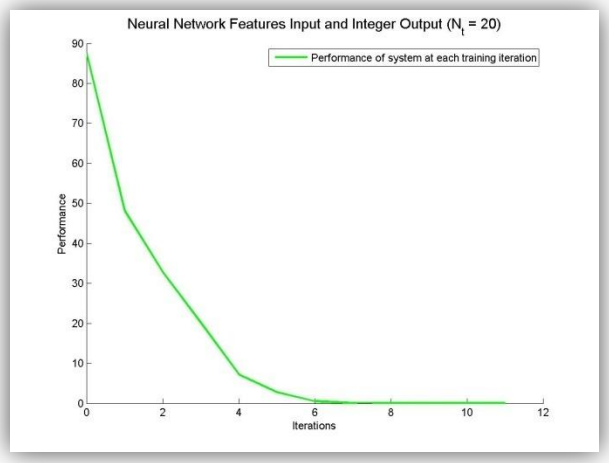

Figure 36

Figure 35 and Figure 36 are two examples of the training of two different NN systems. For the system in Figure 35, we used the 23 extracted features as the network input and the out was 5 digits binary output. The system never achieved our performance goal of 0.001. On the other hand, when we used the features and the output was a single integer (1 or 2), the system reached its goal in less than 6 iterations.

2.6.2. Error rate

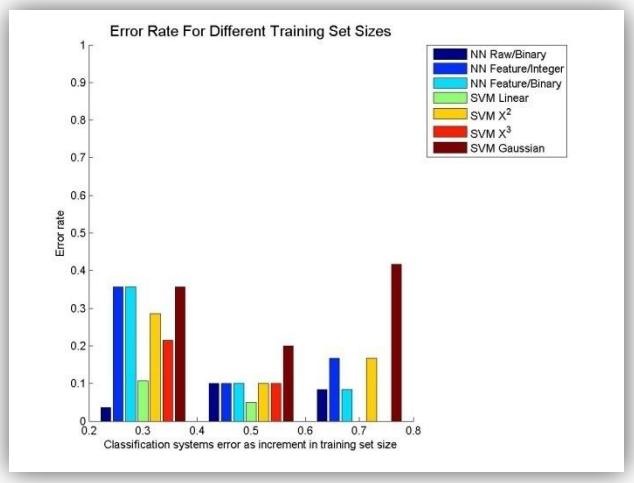

Figure 37

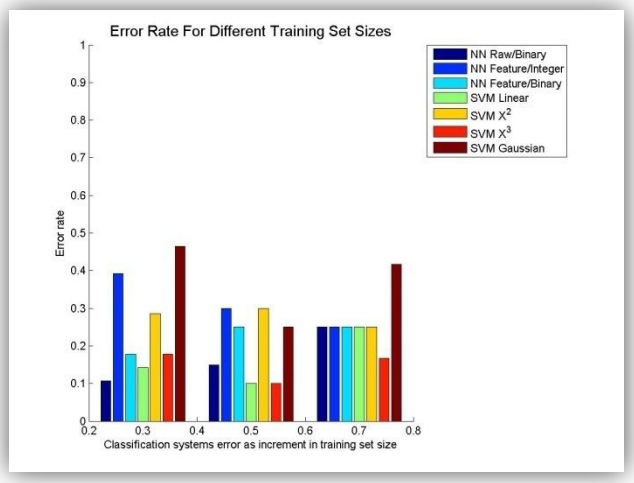

Figure 38

As you can see in Figure 37 and Figure 38, the systems error rates varies by the size of the training sample. Particularly interesting was to see that for a NN with the image raw input the best performance was obtained when we have the fewer training samples. The increment of training data led to the over fitting of the NN. Overall we observed that too many training samples (over 14 samples) led to over fitting. However, we decided to show the error rate from two different runs to point out the effect of the selected training data. Our algorithms selected the training samples randomly. We just tell the system how many

samples to choose. You can see a difference in performance in the figures above. Therefore, we can conclude that we need to analyze the data used for training. Selecting outliners for training can lead to misclassifications. Finally when designing an NN or SVM we need to study our data to determine what training samples are good and how many of them we are going to use.

When comparing NN with SVM we can say that we observe more consistent results in the side of SVM. NN systems are very sensitive to input noise. On the other hand, for SVM we need to preprocess all data in order to extract features. Although we did the same preprocessing of the data for NN, SVM sees to handle features better than NN. However, my intuition led me to think that this is true given the characteristics of my problem—identifying two letters. Therefore, NN could outperform SVM in other applications.

We would like to clarify that there is a simple concept that applies to these experiments; trash in, trash out. We tried to select meaningful features from the characters to detect. Given that we were only dealing with two classes of characters I found enough the 23 features mentioned above. However, we need to admit that a comprehensive study of this problem should be done in order to decide which features are good for classification. A new set of features may lead to better and more consistent results.

2.7. Matlab's code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ECE 662: Homework #2        %%
%% Problem 2                   %%
%% Prof. Boutin                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%clear all data and variables
clear all
%close all windows
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load data                    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%load data
letters = ['A', 'R']; %letter to categorize
numC = size(letters, 2); %number of letters or classes to categorize
numSL = 20;   %Number of training samples per letter
letBinSize = 4; %binary size of letters
%Next we load each letter image. All images
%should have the same dimensions
[m, n] = size(imread('char/A1.tif', 'tif'));
%create array that is going to contain training data
nnData = zeros( 50*50, numSL * numC );
svmData = zeros( 23, numSL * numC );
outDataB = zeros( letBinSize, numSL * numC );
```

```
outDataL = zeros( 1, numSL * numC );

%get training data
for i = 1 : numC %loops for each letter
   for j = 1 : numSL %loops for each letter sample

      %load image
      temp = imread(sprintf('%s%s%d%s', 'char/', letters(i),...
               j, '.tif'), 'tif');

      %give label
      outDataB(:, (i-1) * numSL + j ) = de2bi(i, letBinSize);
      outDataL(:, (i-1) * numSL + j ) = i;

      %preprocess image for NN
      nnData(:, (i-1) * numSL + j) = reshape((1-preprocess(temp)), 1, []);
      %preprocess image and get features for NN and SVM
      svmData(:, (i-1) * numSL + j) = getFeatures(preprocess(temp));
   end
end

%loop to increment training set size
error = zeros(3, 7);
count = 1;
for i = 0.3:0.2:0.7

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% separate data for training, evaluation, and testing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
index = 1:numSL * numC;
[trainD1, evalD1, testD1] = dividevec(index(1:20), index(1:20), 0, 1-i);
[trainD2, evalD2, testD2] = dividevec(index(21:40), index(21:40), 0, 1-i);
%merge data for raw NN approoach
trainSetNN =    [nnData(:, trainD1.P) nnData(:, trainD2.P)];
outTrainNN =    [outDataB(:, trainD1.P) outDataB(:, trainD2.P)];
testSetNN =     [nnData(:, testD1.P) nnData(:, testD2.P)];
%outTestNN =     [outDataB(:, testD1.P) outDataB(:, testD2.P)];
%merge data for feature based NN and SVM approach
trainSetSVM =   [svmData(:, trainD1.P) svmData(:, trainD2.P)];
outTrainSVM =   [outDataL(:, trainD1.P) outDataL(:, trainD2.P)];
testSetSVM =    [svmData(:, testD1.P) svmData(:, testD2.P)];
outTestSVM =    [outDataL(:, testD1.P) outDataL(:, testD2.P)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Train NN systems
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%create NN network with raw image input and binary output
numInternalNeurons = [20, size(outTrainNN, 1)];
netB = newff(minmax(trainSetNN), numInternalNeurons, ...
         {'logsig', 'logsig'}, 'traingdx');
```

```
%train network
netB.LW{2,1} = netB.LW{2,1}*0.01;
netB.b{2} = netB.b{2}*0.01;
netB.performFcn = 'sse';
netB.trainParam.goal = 0.0001;
netB.trainParam.show = 20;
netB.trainParam.epochs = 5000;
netB.trainParam.mc = 0.95;
[netB,trB]=train(netB, trainSetNN, outTrainNN);
%plot and save training performance
f = figure;
hold on
plot(trB.epoch, trB.perf, '-r', 'LineWidth', 2)
title(sprintf('%s%d%s', ...
      'Neural Network Raw Image Input and Binary Output (N_t = ', ...
      size(trainSetNN, 2), ')'), 'FontSize', 14);
xlabel('Iterations');
ylabel('Performance');
legend('Performance of system at each training iteration');
hold on
%saveas(f, sprintf('%s%d%s', 'img\NNRaw', ...
%         size(trainSetNN, 2), '.jpg'), 'jpg');
close(f);

%create NN network with features as inputs and single integer output
numInternalNeurons = [23, 1];
netS = newff(minmax(trainSetSVM), numInternalNeurons, ...
        {'logsig', 'purelin'}, 'trainlm');
%train network
netS.performFcn = 'sse';
netS.trainParam.goal = 0.0001;
netS.trainParam.show = 20;
netS.trainParam.epochs = 5000;
[netS,trS]=train(netS, trainSetSVM, outTrainSVM);
%plot and save training performance
f = figure;
hold on
plot(trS.epoch, trS.perf, '-g', 'LineWidth', 2)
title(sprintf('%s%d%s', ...
      'Neural Network Features Input and Integer Output (N_t = ', ...
      size(trainSetNN, 2), ')'), 'FontSize', 14);
xlabel('Iterations');
ylabel('Performance');
legend('Performance of system at each training iteration');
hold on
%saveas(f, sprintf('%s%d%s', 'img\NNFS', ...
%         size(trainSetNN, 2), '.jpg'), 'jpg');
close(f);

%create NN network with features as inputs and binary output
numInternalNeurons = [23, size(outTrainNN, 1)];
netFB = newff(minmax(trainSetSVM), numInternalNeurons, ...
```

```
        {'logsig', 'logsig'}, 'trainlm');
%train network
netFB.performFcn = 'sse';
netFB.trainParam.goal = 0.0001;
netFB.trainParam.show = 20;
netFB.trainParam.epochs = 5000;
[netFB,trFB]=train(netFB, trainSetSVM, outTrainNN);
%plot and save training performance
f = figure;
hold on
plot(trFB.epoch, trFB.perf, '-b', 'LineWidth', 2)
title(sprintf('%s%d%s', ...
      'Neural Network Features Input and Binary Output (N_t = ', ...
      size(trainSetNN, 2), ')'), 'FontSize', 14);
xlabel('Iterations');
ylabel('Performance');
legend('Performance of system at each training iteration');
hold on
%saveas(f, sprintf('%s%d%s', 'img\NNFB', ...
%          size(trainSetNN, 2), '.jpg'), 'jpg');
close(f);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Test NN systems
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%size of test output vector
sVO =  size(outTestSVM, 2);

%create NN network with raw image input and binary output
outSim = sim(netB, testSetNN);
outSim = bi2de(uint32(round(outSim))');
res = outSim == outTestSVM';
error(count, 1) = 1 - sum( res ) / sVO;

%create NN network with features as inputs and single integer output
outSim = sim(netS, testSetSVM);
outSim = uint32(round(outSim))';
res = outSim == outTestSVM';
error(count, 2) = 1 - sum( res ) / sVO;

%create NN network with features as inputs and binary output
outSim = sim(netFB, testSetSVM);
outSim = bi2de(uint32(round(outSim))');
res = outSim == outTestSVM';
error(count, 3) = 1 - sum( res ) / sVO;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Train SVM systems
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%linear kernel
svm1 = svmtrain(trainSetSVM', outTrainSVM', 'Kernel_Function', 'linear');

%quadratic  kernel
svm2 = svmtrain(trainSetSVM', outTrainSVM', 'Kernel_Function', 'quadratic');

%polynomial kernel
svm3 = svmtrain(trainSetSVM', outTrainSVM', 'Kernel_Function', 'polynomial');

%Gaussian Radial Basis Function kernel (rbf)
svm4 = svmtrain(trainSetSVM', outTrainSVM', 'Kernel_Function', 'rbf');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Test SVM systems
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%linear kernel
outSim = svmclassify(svm1, testSetSVM');
res = outSim == outTestSVM';
error(count, 4) = 1 - sum( res ) / sVO;

%quadratic  kernel
outSim = svmclassify(svm2, testSetSVM');
res = outSim == outTestSVM';
error(count, 5) = 1 - sum( res ) / sVO;

%polynomial kernel
outSim = svmclassify(svm3, testSetSVM');
res = outSim == outTestSVM';
error(count, 6) = 1 - sum( res ) / sVO;

%Gaussian Radial Basis Function kernel (rbf)
outSim = svmclassify(svm4, testSetSVM');
res = outSim == outTestSVM';
error(count, 7) = 1 - sum( res ) / sVO;

count = count + 1;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f = figure('Position', [200, 200, 800, 400]);
hold on
bar(0.3:0.2:0.7, error)
xlabel('Classification systems error as increment in training set size');
ylabel('Error rate');
title('Error Rate For Different Training Set Sizes', 'FontSize', 14);
```

```
legend('NN Raw/Binary', 'NN Feature/Integer', 'NN Feature/Binary', ...
   'SVM Linear', 'SVM X^2', 'SVM X^3', 'SVM Gaussian', ...
   'Orientation', 'vertical', 'Location', 'NorthEastOutside');
xlim([0.2 0.8]);
ylim([0 1]);
hold off
saveas(f, 'img\errorHisto2.jpg', 'jpg');
```

3. Parzen windows, nearest neighborhood, and k- nearest neighborhood

3.1. Experiment description

3.1.1. For these experiments we used the same data as Section 2

3.1.2. We used Matlab's principal component analysis (PCA) algorithm to project the data.

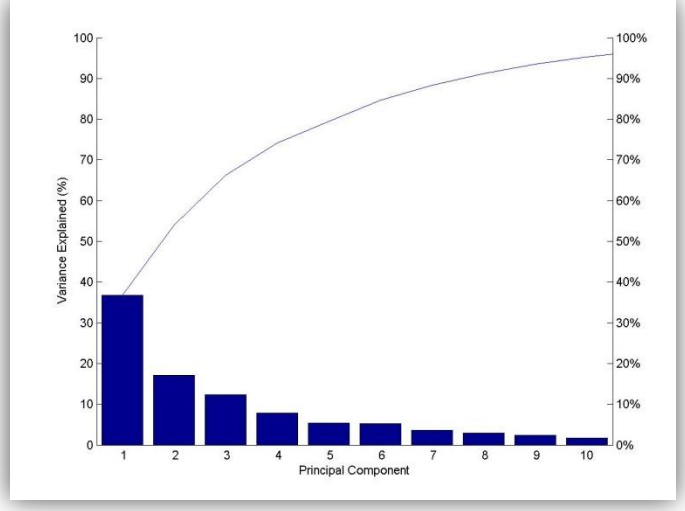

Figure 39

The figure above shows how much the components describe the data. We have that with the first two components we can describe almost 60% of the data. Below you can observe how our data was projected. In Figure 40, you can see that each class has a well defined cluster, with the exception of a few outliners.

3.1.3. We projected the data to a 2-D plane.

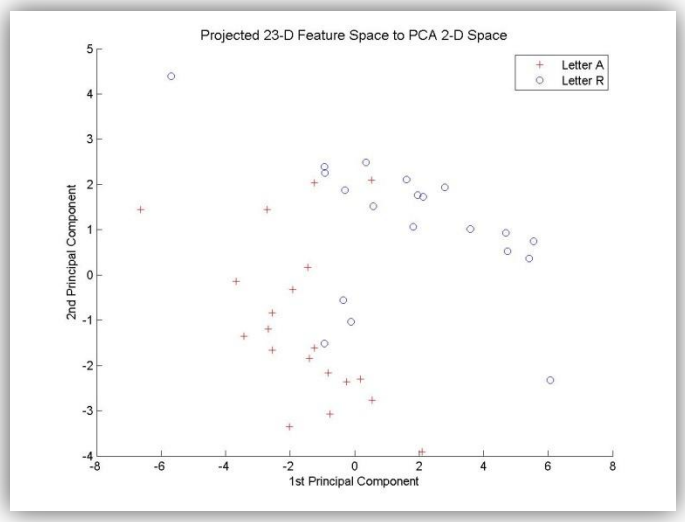

Figure 40

3.1.4. Parzen Windows

For the Parzen Windows we used a square window. We tested different sizes and showed the results for a window width of 1.5.

3.1.5. K-Nearest Neighbor (K-NN)

We created an algorithm for K-NN, and the set k to one in order to implement nearest neighbor approach. The distance between samples was Euclidean distance.

$$distance = \sqrt{(x_{train} - x_{test})^2}$$

3.2. Results and analysis

We got interesting results from our experiments. First, the Parzen Window strategy gave a noticeable decrease in error rate as we incremented the training set size. We also saw the same behavior with the K-NN systems, but not as consistent as with the Parzen Window. Also, we can notice that incrementing the number of closest neighbors in the nearest neighbor approach increase the error rate. This may be due to our sparse data. You can notice a separation between the classes, see Figure 40. The problem is that the classes are really close at the boundary of the clusters. Therefore, as we increase the number of neighbors the probability of mismatch increases. This means that given our data, a parametric method, as the one we used in section 1, should be a better strategy to classify our classes. You can see examples of outliers, when using a Pazen Window in Figure 43 and Figure 44.

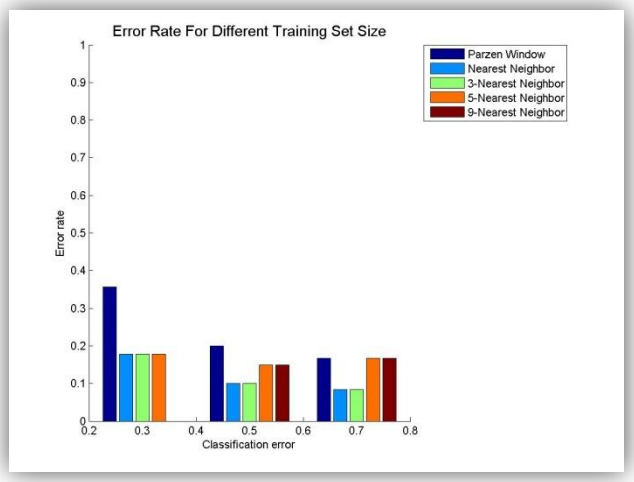

Figure 41

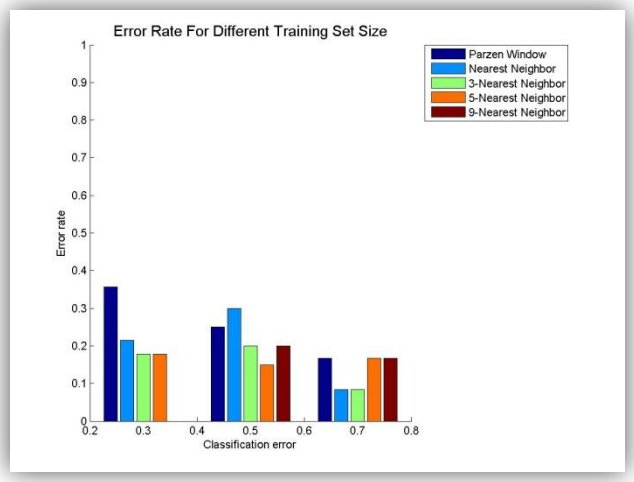

Figure 42

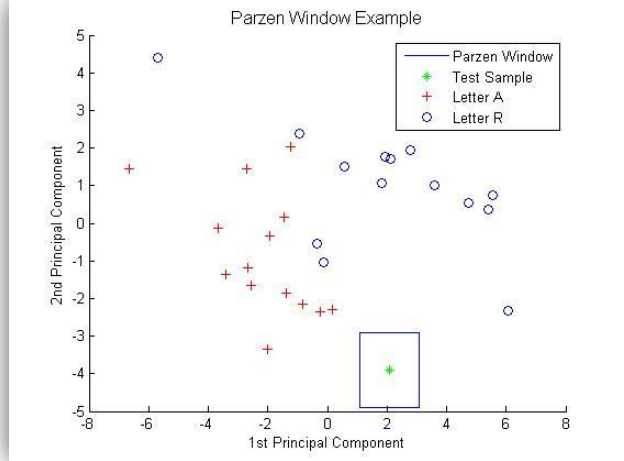

Figure 43

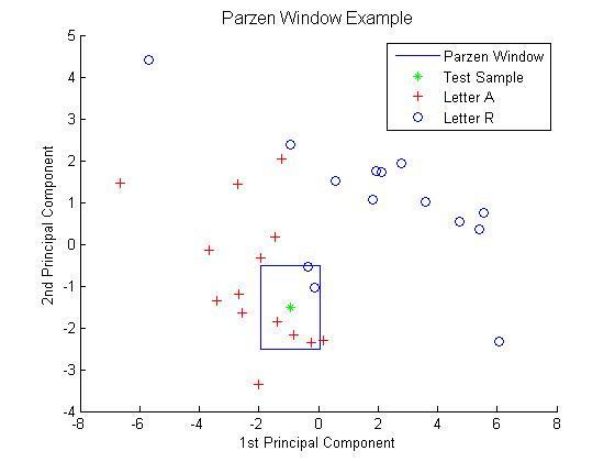

Figure 44

3.3. Matlab's code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ECE 662: Homework #2          %%
%% Problem 2                     %%
%% Prof. Boutin                  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%clear all data and variables
clear all
%close all windows
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load data                     %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%load data
letters = ['A', 'R']; %letter to categorize
numC = size(letters, 2); %number of letters or classes to categorize
numSL = 20;   %Number of training samples per letter
%Next we load each letter image. All images
%should have the same dimensions
[m, n] = size(imread('char/A1.tif', 'tif'));
%create array that is going to contain training data
data = zeros( numSL * numC, 23 );
outData = zeros( numSL * numC, 1 );

%get training data
for i = 1 : numC %loops for each letter
    for j = 1 : numSL %loops for each letter sample

        %load image
        temp = imread(sprintf('%s%s%d%s', 'char/', letters(i),...
```

```
                        j, '.tif'), 'tif');

            %give label
            outData( (i-1) * numSL + j, : ) = i;

            %preprocess image and get features for NN and SVM
            data( (i-1) * numSL + j, :) = getFeatures(preprocess(temp));
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Project from 23-d to 2-d
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %project data for class A
    stdData = std(data);
    stdData = data ./ repmat(stdData, size(data, 1), 1);
    [pca, pData, vars, t] = princomp(stdData);

    %plot a pareto percent of variability explained
    percent_explained = 100*vars/sum(vars);
    f = figure;
    hold on
    pareto(percent_explained);
    xlabel('Principal Component')
    ylabel('Variance Explained (%)')
    hold off
    %saveas(f, 'img\PCAVar.jpg', 'jpg');
    close(f);

    %plot projected data
    f = figure;
    hold on
    plot(pData(1:20, 1), pData(1:20, 2), '+r');
    plot(pData(21:40, 1), pData(21:40, 2), 'ob');
    xlabel('1st Principal Component');
    ylabel('2nd Principal Component');
    legend('Letter A', 'Letter R')
    title('Projected 23-D Feature Space to PCA 2-D Space', 'FontSize', 12)
    hold off
    %saveas(f, 'img\2DProjection.jpg', 'jpg');
    close(f);

    %plot projected data
    f = figure;
    hold on
    plot3(pData(1:20, 1), pData(1:20, 2), pData(1:20, 3), '+r');
    plot3(pData(21:40, 1), pData(21:40, 2), pData(21:40, 3), 'ob');
    xlabel('1st Principal Component');
    ylabel('2nd Principal Component');
    zlabel('3nd Principal Component');
    legend('Letter A', 'Letter R')
    title('Projected 23-D Feature Space to PCA 3-D Space', 'FontSize', 12)
    grid on
    view(3)
```

```
hold off
%saveas(f, 'img\3DProjection.jpg', 'jpg');
close(f);

%loop to increment training set size
error = zeros(3, 5);
count = 1;
for i = 0.3:0.2:0.7

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Separate data for training, evaluation, and testing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
index = 1:numSL * numC;
d = 3; %dimensions of projected data
[trainD1, evalD1, testD1] = dividevec(index(1:20), index(1:20), 0, 1-
i);
[trainD2, evalD2, testD2] = dividevec(index(21:40), index(21:40), 0, 1-
i);
%merge data for raw NN approoach
trainSetA =   pData(trainD1.P, 1:d);
%outTrainA =   outData(trainD1.P, 1:d);
trainSetR =   pData(trainD2.P, 1:d);
%outTrainR =   outData(trainD2.P, 1:d);
testSet =     [pData(testD1.P, 1:d); pData(testD2.P, 1:d)];
outTest =     [outData(testD1.P, 1); outData(testD2.P, 1)];
%size of test data
sVO = size(outTest, 1);
%size of train data
nT = size(trainSetA, 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Parzen Window Classification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%train and test system
simOut = ParzenWindowSQR2(trainSetA, trainSetR, testSet, 3, d);
%Compute error
res = simOut == outTest;
pwE = 1 - sum( res ) / sVO;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% nearest neighbor Classification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%train and test system
simOut = KNN(trainSetA, trainSetR, testSet, 1, d);
%Compute error
res = simOut == outTest;
nnE = 1 - sum( res ) / sVO;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% k-nearest neighbor Classification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%train and test system
```

```
simOut = KNN(trainSetA, trainSetR, testSet, 3, d);
%Compute error
res = simOut == outTest;
knnE(1, 1) = 1 - sum( res ) / sVO;

%train and test system
simOut = KNN(trainSetA, trainSetR, testSet, 5, d);
%Compute error
res = simOut == outTest;
knnE(1, 2) = 1 - sum( res ) / sVO;

%train and test system
if( nT < 9) simOut = outTest;
else
simOut = KNN(trainSetA, trainSetR, testSet, 9, d);
end
%Compute error
res = simOut == outTest;
knnE(1, 3) = 1 - sum( res ) / sVO;

%save error
error(count, :) = [pwE, nnE, knnE];

count = count + 1;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f = figure('Position', [200, 200, 800, 400]);
hold on
bar(0.3:0.2:0.7, error)
xlabel('Classification error');
ylabel('Error rate');
title('Error Rate For Different Training Set Size', 'FontSize', 14);
legend('Parzen Window', 'Nearest Neighbor', '3-Nearest Neighbor', ...
    '5-Nearest Neighbor', '9-Nearest Neighbor', 'Location',
'NorthEastOutside');
xlim([0.2 0.8]);
ylim([0 1]);
hold off
%saveas(f, 'img\errorHisto3D1.jpg', 'jpg');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function simOut = ParzenWindowSQR2(trainSet1, trainSet2, testSet, h, d)

%width of window and volume of window
V = h^d;
%Number of training samples
n = size(trainSet1, 1);
```

```
%Number of test samples
sVO = size(testSet, 1);
%vector for classified test samples
simOut = zeros(sVO, 1);

%compute probability of each test sample and classify
for k = 1 : sVO

    %window coordinates
    lB = testSet(k, :) - (h/2)*ones(1, d);
    uB = testSet(k, :) + (h/2)*ones(1, d);

    %find elements in A and R that their
    %features are inside the range of the window
    %samples from trainSet1 that are in window
    winS = 0;
    for z=1:n
         if(lB <= trainSet1(z, :))
             if(uB >= trainSet1(z, :))
                 winS = winS + 1;
             end
         end
         if(lB <= trainSet2(z, :))
             if(uB >= trainSet2(z, :))
                 winS = winS - 1;
             end
         end
    end
    %if wins >= 0 the test sampe is an A, R otherwise
    if(winS > 0)
        simOut(k) = 1;
    elseif(winS < 0)
        simOut(k) = 2;
    else
        simOut(k) = 0;
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function simOut = KNN(trainSetA, trainSetR, testSet, k, d)

%size of vectors
Ntest = size(testSet, 1);
N = size(trainSetA, 1);

%output vector
simOut = zeros(Ntest, 1);

%distance vectors
```

```
distA = zeros(N, 1);
distR = zeros(N, 1);

%loop to go trough all test samples
for i=1:Ntest
    %loop to go through all train samples
    for j=1:N
        %init
        distA(j) = 0;
        %loop to go through features
        for z=1:d
            %distance from A
            distA(j) = distA(j) + (trainSetA(j, z) - testSet(i, z))^2;
            distR(j) = distR(j) + (trainSetR(j, z) - testSet(i, z))^2;
        end
        distA(j) = sqrt(distA(j));
        distR(j) = sqrt(distR(j));
    end

    %sort distances in acending order
    distA = sort(distA, 'ascend');
    distR = sort(distR, 'ascend');

    %loop to take the closest samples
    counter = 0;
    acount = 1;
    rcount = 1;
    for j=1:k
        if(distA(acount) <= distR(rcount))
            counter = counter + 1;
            acount = acount + 1;
        else
            counter = counter - 1;
            rcount = rcount + 1;
        end
    end
    %decide
    if(counter >= 0)
        simOut(i) = 1;
    else
        simOut(i) = 2;
    end
end
```

4. Appendix

4.1. Image letter samples

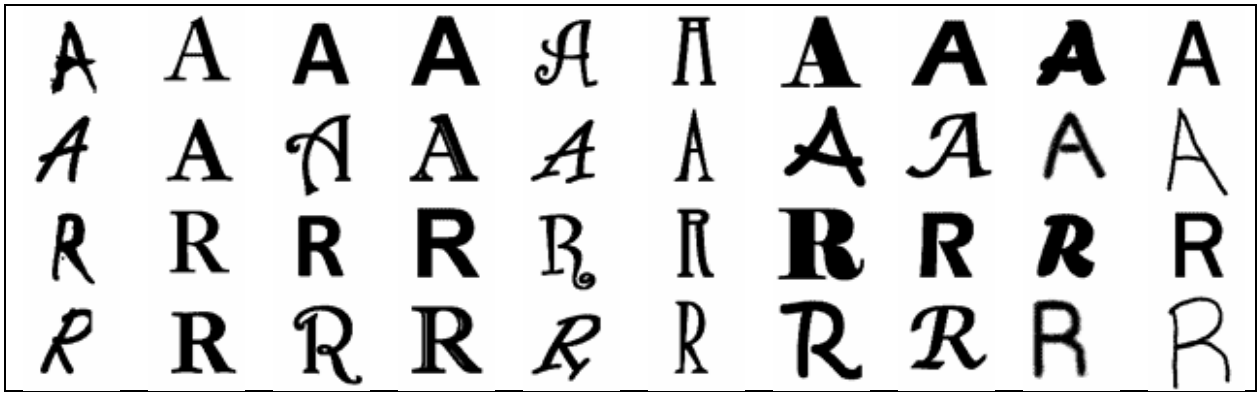

Figure 45

4.2. Matlab's code used by most problems

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ECE 662: Homework #2        %%
%% Problem 2                   %%
%% Prof. Boutin                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%I_in is a binary matrix representing a black and white
%image. This function finds the boundary closest to the character.
%Then it cropped and resize the image.
%The function returns a 7x7 matrix with the preprocessed image.
function I = preprocess(I_in)

%find edge of image
I = edge(uint8(I_in));

%fill inside of character
se = strel('square',2);
I = imdilate(I, se);
I= imfill(I,'holes');

%find tigth boundary for character
I = bwlabel(I);
%disp(num);         %% If you want to check how many thing were found
I = regionprops(I, 'BoundingBox');
bounds = floor([I.BoundingBox]);

%check bounds are inside image boundaries
if(bounds(1) < 1)
   bounds(1) = 1;
end
if(bounds(2) < 1)
   bounds(2) = 1;
```

```
end
if(bounds(2) + bounds(4) > size(I_in, 1))
   bounds(4) = size(I_in, 1) - bounds(2);
end
if(bounds(1) + bounds(3) > size(I_in, 2))
   bounds(3) = size(I_in, 2) - bounds(1);
end

%crop image
I = I_in(bounds(2):bounds(2) + bounds(4), ...
         bounds(1):bounds(1) + bounds(3));




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ECE 662: Homework #2        %%
%% Problem 2                   %%
%% Prof. Boutin                %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%This funtions rotates the image to the given
%degree and compute the sum of each column and row
%in the rotated image
function F = getFeatures(img)

%reserve space for feature vector
F = zeros(1, 23);

%preprocess image and compute dist
I0 = 1-img;
%compute horizontal pdf
xpdf = 50-sum(I0, 2);
%compute vertical pdf
ypdf = 50-sum(I0, 1)';

%rotate image
I45 = 1-preprocess(imrotate(img, 45));
%compute horizontal pdf
xpdf45 = 50-sum(I45, 2);
%compute vertical pdf
ypdf45 = 50-sum(I45, 1)';

%get square frame weights
fw = zeros(1, 3);
fw(1) = sum(sum(I0(20:30, 20:30)));
fw(2) = sum(sum(I0(10:40, 10:40))) - fw(1);
fw(3) = sum(sum(I0)) - fw(2) - fw(1);
```

```
%set features
for i=1:5
   F(i) = sum(xpdf(10*(i-1)+1:10*(i-1)+10));
   F(i+5) = sum(ypdf(10*(i-1)+1:10*(i-1)+10));
   F(i+10) = sum(xpdf45(10*(i-1)+1:10*(i-1)+10));
   F(i+15) = sum(ypdf45(10*(i-1)+1:10*(i-1)+10));
end
F(21:23) = fw;
```