EE662 Spring 2008 - Homework 2

April 15, 2008

1 Linear Discriminant Criterion Function

Here we investigate the use of the Fisher criterion function $J(w) = \frac{w^t S_B w}{w^t S_w w}$ compared to $J(w) = w^t S_B w$ when selecting a hyperplane to separate the data in two classes.

1.1 Theoretical Remarks

The two approaches for the choice of w can be formulated as two optimization problems:

Problem1:Consider the optimization problem to maximize $J(w) = \frac{w^t S_B w}{w^t S_w w}$, formulated as follows:

$$\begin{gathered} \text{maximize } w^t S_B w \\ \text{subject to } w^t S_w w = 1 \end{gathered}$$

The Lagrangian function is written as

$$L_w = w^t S_B w - \lambda w^t S_w w$$

The necessary condition for the maxima is given by

$$\begin{gathered} \frac{dL_w}{dw} = S_B w - \lambda S_w w = 0 \\ S_B w = \lambda S_w w \\ S_w^{-1} S_B w = \lambda w \end{gathered}$$

We use the fact that $S_B w \propto m_2 - m_1$ and have the final solution:

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

Problem 2: Consider the optimization problem to maximize $J(w) = w^t S_B w$, formulated as follows:

$$\begin{gathered} \text{maximize } w^t S_B w \\ \text{subject to } w^t w = 1 \end{gathered}$$

The Lagrangian function is written as

$$L_w = w^t S_B w - \lambda w^t w$$

The necessary condition for the maxima is given by

$$\frac{dL_w}{dw} = S_B w - \lambda w = 0$$
$$S_B w = \lambda w$$

We use the fact that $S_B w \propto m_2 - m_1$ and have the final solution:

$$w \propto (m_2 - m_1)$$

The solution for the **Problem 2** implies that the decision hyperplane between the two classes will be orthogonal to the line between the classes' means regardless of their *intra-class* covariances. The solution for the **Problem 1** implies a decision hyperplane between two classes that is *rotated* from the direction of the line between the means depending on the *intra-class* covariances.

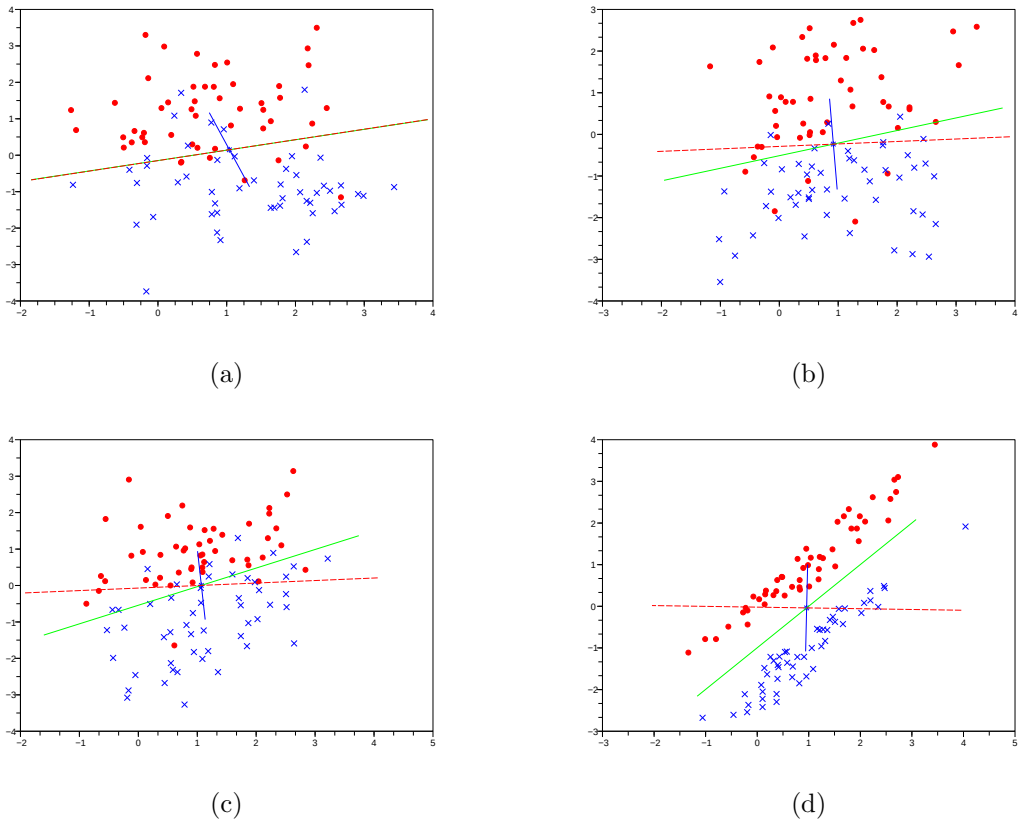
In the subsection below we verify experimentally the consequences of the different choices of $J(w)$.

1.2 Fisher Discriminant Experiments

For this experiment we generate 100 points for each of two classes of normally distributed data in the 2D space with $\mu_1 = (0\ 0)^T$,$\mu_2 = (1\ 1)^T$ and, $\Sigma_1 = \Sigma_2 = I + R$, where R is a matrix that ajusts the correlation between the features for each class by having the anti-diagonal elements equal to ρ and zeros in all the other positions.

Figure 1 shows the decision surface for both criterion functions separating the two normally distributed 2D data. As we can see in Figure 1a, for weakly correlated features ($\rho = 0.1$), there is little difference between using the Fisher Discriminant and the separation hyperplane that is orthogonal to the line between the means. As we can see in Figures 1b,1c and, 1d, the Fisher discriminant does a much better job separating the classes as the correlation between features increases ($\rho = \{0.2, 0.5, 0.95\}$) since it takes into consideration the intra-class correlation structure between the features. This happens because when the Fisher Criterion is used, the separation line originally orthogonal to the line between the means is "rotated" by the matrix S_w^{-1} in order to better separate the points between the classes.

A intuitive way to understand why the Fisher Criterion function $J(w) = \frac{w^t S_B w}{w^t S_w w}(1)$ is better than just using $J(w) = w^t S_B w(2)$ is that the w that maximizes (2) is always parallel to the line between the classes' means. Therefore, when (2) is used as a criterion function, the data will *always* be projected in the line between the means and a separation threshold is selected in order to best separate the points. This works well if the 2 features are not correlated or weakly correlated. The supperiority of the Fisher criterion (1) is observed when the data exhibits high correlation between the features. In that case, only projecting data over the line between the means is not enough to separate the classes. Instead, the Fisher Criterion will project the data over a line "rotated" from the line between the means. The rotation is done in order to maximize the separation between the projected points from both classes.



(a) (b) (c) (d)

Figure 1: Comparison of the decision lines using the Fisher criterion ($J(w) = \frac{w^t S_B w}{w^t S_w w}$) and $J(w) = w^t S_B w$ for bi-variated normally distributed data as we vary the correlation between the 2 features of each class. The correlation coefficients tested are: (a) $\rho = 0.1$, (b) $\rho = 0.2$, (c) $\rho = 0.5$ and, (d)$\rho = 0.95$

2 Support Vector Machines and Artificial Neural Network Classifiers

2.1 Diabetes Data Set

For the experiments with Support Vector Machines and Artificial Neural Network we use the Diabetes data set. Each sample point in this data set has 8 features and is classified as "1"(positive) or "0" (negative). All the features are scaled to the [0,1] range. The whole dataset contains 768 samples.

In the experiments below we divide the dataset among *training set*, *cross-validation set* and, *testing set*. The cross-validation set is used to evaluate the performance of the classifier as we vary its parameters. Finally, once we believe we have the best set of parameters, we test the classifier using the testing set and compute the final classifier accuracy.

2.2 Support Vector Machines

All the experiments with Support Vector Machines performed in this paper used the LIBSVM library (*http://www.csie.ntu.edu.tw/ cjlin/libsvm/*). The package also provides a Python script called `easy.py` that allows us to easily experiment with the SVM classifier. We provide the training set of 384 samples to the script, which splits into training and cross-validation sets several times. The test set with 384 samples is provided to finally evaluate the accuracy of the SVM classifier.

All the experiments used the RBF (Radial Basis Function) kernel defined as: $K(x, y) = e^{\gamma||x-y||^2}$.

The SVM classification also depends on the parameter C, which is the *Penalty Parameter of the Error Term*. For very large values of C, no error is allowed for the training samples, what can lead to overfitting. Conversely, if small values of C are chosen, the classifier imprecision will increase. We perform the following training strategy to select the combination of the parameters C and γ that gives us the best classification accuracy:

- Select values for C and γ and train the classifier using the training dataset;
- Evaluate the classifier performance using the cross-validation data;
- Repeat the procedure for other values of C and γ until best accuracy is found for the cross-validation test;
- Use the classifier with the values of C and γ that give the best accuracy over the cross-validation data to classify test samples.

Figure 2 shows the accuracy for the classifier for the parameters C and γ validaded over the cross-validation data. The best parameters selected were: C=2048 and, γ=0.000488. We used these parameters to test the testing set of the data and the final accuracy verified is 80.5% (the classifier classifies correctly 309 out of 384 samples from the testing set).

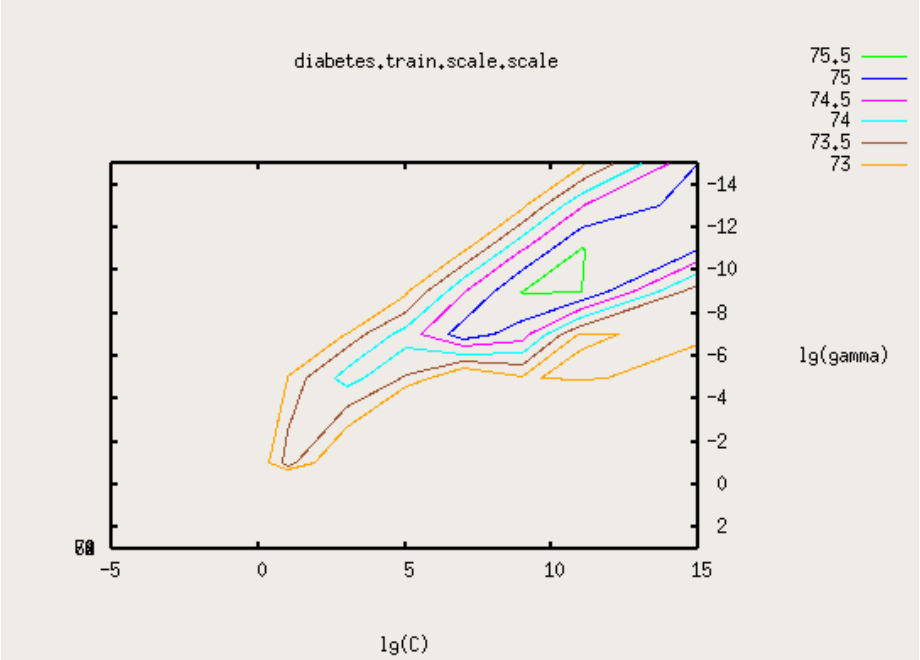


Figure 2: SVM Classifier accuracy as C and γ are varied. The best accuracy over the cross-validation data is achieved when C=2048 and γ=0.000488.

2.3 Artificial Neural Network

All the experiments with Artificial Neural Network performed in this paper used the FANN library(*http://leenissen.dk/fann/*). This library provide the basic software infra-structure to build and train neural networks by specifying the networks parameters and training protocols. We augmented the library to perform experiments automatically as we especify them in a configuration file.

2.4 Artificial Neural Network Experiments

The main goal of the experiments is to determine the accuracy of the ANN classification as we change some of the network parameters.

During all the experiments, the architecture of the network used is described by the following parameters that remain fixed during the experiments:

- **Number of layers**: 3
- **Number of output units**: 2
- **Number of input units**: 8
- **Hidden/Output unit activation functions**: symmetric sigmoidal
- **Training protocol**: Batch backpropagation
- **Maximum number of epochs**: 30,000

A set of experiments is performed to test changes in the following parameters:

- Size of the training set: number of training samples
- Number of hidden units
- Maximum MSE Training Error accepted

2.4.1 Experiment 1 - Number of Samples vs. Number of Hidden Units

We divided the dataset among *training*(384 samples), *cross-validation*(150 samples) and, *test*(234 samples) data sets. In Figure 3 we show the accuracy of classification as we vary the number of training samples and the number of hidden units, verifying the accuracy on the cross-validation data. The maximum MSE error over the training data is set to be 0.01.

As we can see in Figure 3, for the diabetes dataset experimented by us, the accuracy improves as the number of training samples grows lage. However, no accuracy degradation trend is seen as we increase the number of hidden neurons for a fixed size of the training set, what contradicts the "rule of thumb" proposed in DHS textbook that the number of hidden neurons should approximately the size of the training sample set divided by 10.

The best accuracy (83%) is achieved when we choose the number of training samples to be 120 and the number of hidden units to be 70. However the similar accuracy is seem for 115 hidden units and 180 training data points.

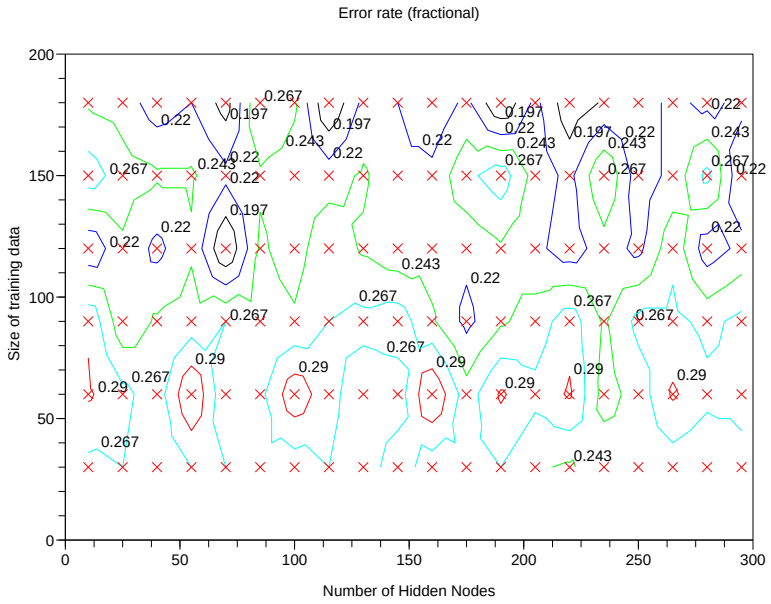


Figure 3: Contour plot of the error rate of the Artificial Neural Network classifier for different number of neurons in the hidden layer and the number of training samples. Each of the points represent one experiment and the contour curves represent the error rate level.

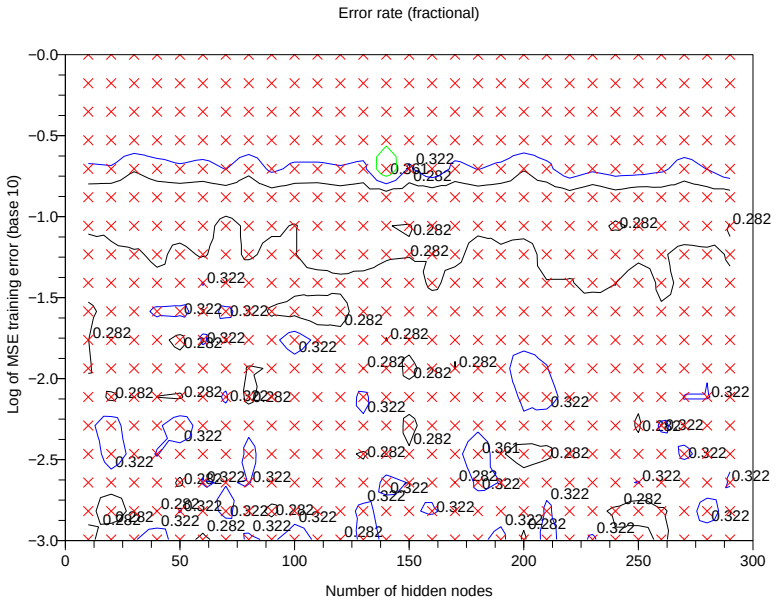


Figure 4: Contour plot of the error rate of the Artificial Neural Network classifier for different values of acceptable MSE training error and number of hidden units. Each of the points represent one experiment and the contour curves represent the error rate level.

2.4.2 Experiment 2 - Number of Hidden Units vs. Maximum MSE for the Training Set

We verify the accuracy of the Artificial Neural Network classifier as we vary the maximum MSE training error accepted during the training phase and testing the classifier with the testing set. The training set with 384 samples and a testing of the same size were used in each trial of this experiment. The results are shown in Figure 4. A large acceptable MSE training error results in a imprecise classifier while a very small error rate leads to overfitting. According to the plot in Figure 4 the accuracy of the classifier doesn't change much as we vary the number of hidden units. The best combination of parameters gives a classification accuracy of 76%, when the number of hidden units is either 30 or 230 and the maximum MSE error is set to 0.13.

Finally we pick the best combination of the best parameters and test over a test data set. The selected parameters are: 100 hidden units (arbitrarily chosen since it doesn't seem to affect our results), MSE error over training data equals to 0.13. The number of samples is set to 384. The accuracy of the classification obtained is 74% (284 samples classified correctly out of 384 samples in total).

2.4.3 Experiment Conclusions

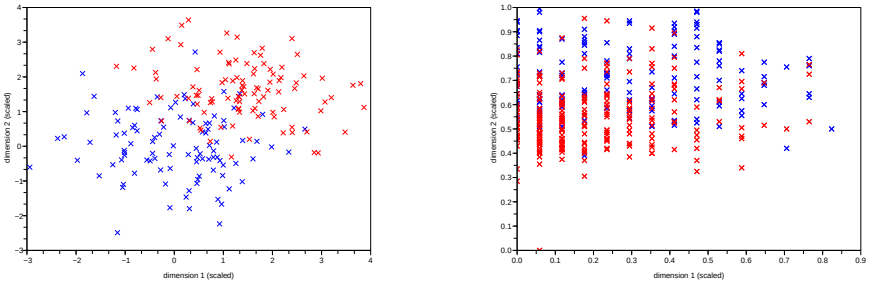
The experiments clearly show that a optimal choice for the MSE error over the training data, as well as a minimum number of samples is desirable for a good classification performance. However, according to our experiments, the selection of the number of hidden units is arbitrary since there is no observable trend in terms of classification accuracy as the number of hidden units changes in either of the experiments.

2.5 Comparison of Artificial Neural Network and Support Vector Machines classifiers

In our experiments the Support Vector Machines showed supperior accuracy when compared to the Artificial Neural Network . However, it's hard to make a final conclusion of what approach is better in the general case.

The SVM classifier directly searchs for a separation surface between the two sets outputting the result of the classification. Conversely, the ANN aims to learn a function based on the inputs and outputs real values with no direct meaning and has to be thresholded, as opposed to the classification decision itself given by the SVM.

Conceptually, SVMs are much simpler to understand than ANNs. When working with SVMs, the user has to pick two parameters C and γ that are intuitive and appear to be related to how mixed the two classes are in the data. ANNs, instead, deal with complicated training protocols and the choice of the parameters that determine the network architecture, such as the number of hidden layers or, yet, the number of hidden units in a 3-layer network. the choice of such parameters is not obvious.



(a) (b)

Figure 5: Layout of the two datasets used during the experiments: (a) synthetic gaussian dataset and, (b) 2D diabetes dataset

3 Experiments with Parzen-Window, K-Nearest Neighbor and, Nearest Neighbor Classifiers

This section presents the results of tests with the Parzen-Window, K-Nearest Neighbor and Nearest Neighbor classifiers. We have implemented all the methods using the Scilab environment.

3.1 Data Sets

For the experiments in this section, the following datasets were used:

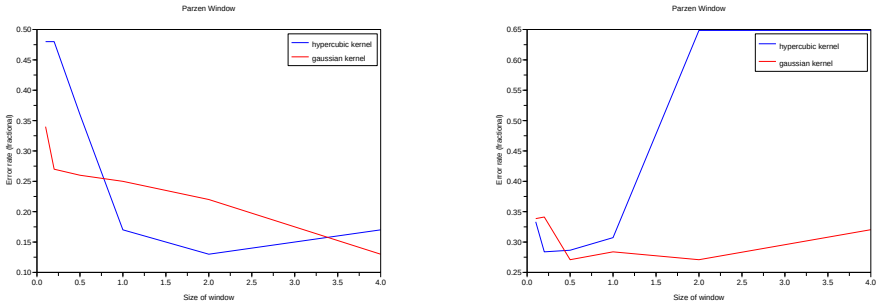
- **Gaussian**: artificially generated gaussian 2D dataset, contains 2 classes with identical covariance matrices, separated by 1 standard deviation.The layout of this dataset is shown in Figure 5a.
- **Diabetes**: the same dataset used in the previous section with data projected in the 2D space. The layout of this dataset is shown in Figure 5b.

3.2 Parzen Window Classifier

The parzen-window classifier was implemented with two kernels: *hypercubic* and, *gaussian*.

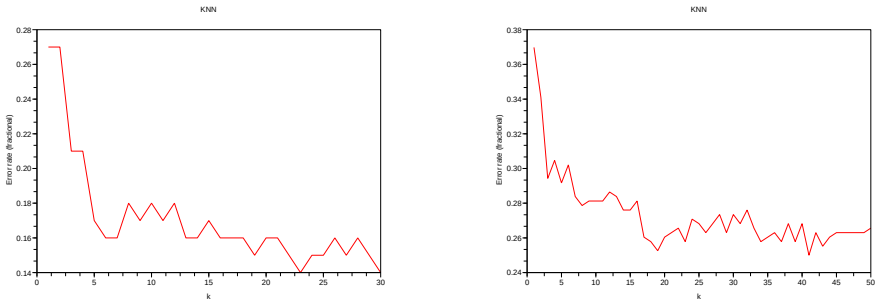
Using these kernels we estimate the probability densities of the two classes in the neighborhood of a sample point $\mathbf{x}$. The classification is performed choosing the class with highest probability in that region.

The performance comparison between the gaussian and the hypercubic kernels is shown in Figure 6. The overall accuracy of the gaussian kernel is better since there is no hard cut in the window as with the hypercubic kernel, resulting in a better estimation of the probability density function in the neighborhood



(a) (b)

Figure 6: Performane of the Parzen Window classifier as the window size increases for: (a)the guassian data and, (b) the 2D diabetes data.



(a) (b)

Figure 7: Performane of the K-Nearest Neighbor classifier as the value of k increases for: (a)the guassian data and, (b) the 2D diabetes data.

of a point. Since the hypercubic window implies in sampling data with fixed hard cuts uniformly in the window region, large window sizes equally consider a broad set of points in the estimation of the probability density decreasing the classification accuracy. In experiments with both datasets the gaussian kernel appears to be more robust, keeping its accuracy relatively higher than the cubic kernel, even for very small or very large window sizes.

3.3 K-Nearest Neighbor Classifier

Here we verify the performance of the K-nearest neighbor classifier as the parameter K (number of neighbors considered) grows. Figure 7 shows the misclassification rate as the value of K increases. As we can see from the experiments, a reasonable value of K is desirable in order to have a good classification ac-

curacy. However, if K grows too large, the classifier considers points that are too far, having its accuracy diminished. The best values of K found from our experiments inspecting Figure 7 are K=23 for the gaussian synthetic data (error rate of 14%) and K=18 to the diabetes data (error rate of 25%).

3.4 Nearest Neighbor Classifier

This classifier is a particular case of the K-Nearest Neighbor when k=1. The separation surfaces induced by this classifier in the 2D Gaussian data is presented in Figure 8b. The surface for the diabetes data is presented in Figure 9b. We make the observation that it could be the case that the application may require speed while tolerating a higher percentage of misclassification. In that case, using the Nearest Neighbor classifier can be a reasonable option since it runs very fast. During our experiments, the error rate for the Nearest Neighbor is 28% for the gaussian data (compared to 14% for the kNN when K=23) and 37% for the diabetes data (compared to 25% for the kNN when K=18).

3.5 Comparison of the Three Techniques

In order to compare all the three classification techniques we select the the parameters that made each of them perform best in the previous experiments. Figure 10 shows the misclassification rate as the number of training samples is varied for both datasets. As we can see from that figure, for both datasets the classification accuracy increases (error decreases) as the number of training samples increases. However, the relative accuracy of each of the classifiers remained the same: K-Nearest Neighbor is the most accurate, followed by the Parzen-Window and finally the Nearest Neighbor.

In figure 8 we show the decision surfaces induced by the classifiers studied in this section for the gaussian data. The same results for the diabetes data are shown in Figure 9.

In order to draw the decision surfaces we make a grid over the space considered sampling points at intervals of 0.025 for de diabetes data and 0.2 for the synthetic gaussian data and use a standard contour plot function to plot the separation surfaces. As we can see in the surfaces of figures 8 and 9, the Nearest Neighbor method leads to obvious overfitting. Moreover, we programmed the Parzen-Window classifiers to choose arbitrarily the class 1 if no point falls in the window. Therefore, regions with no points are mapped to class 1. In the k nearest neighbor method, the arbitrary choice of a default class never happens since the size of the window is always determined by the closest neighbor.

In general K-Nearest Neighbors presented better accuracy when compared to Parzen-Window, being elected the best classifier among all the three studied in this section. The reason is that, as discussed in DHS, the size of the "best" window in the kNN is a function of the training data surrounding the testing sample, rather than some arbitrary function of the overall samples as in the case of the Parzen-Window. A reasonable value of K is desirable for the accuracy of the classifier as verified in our experiments. However, a small value of K will

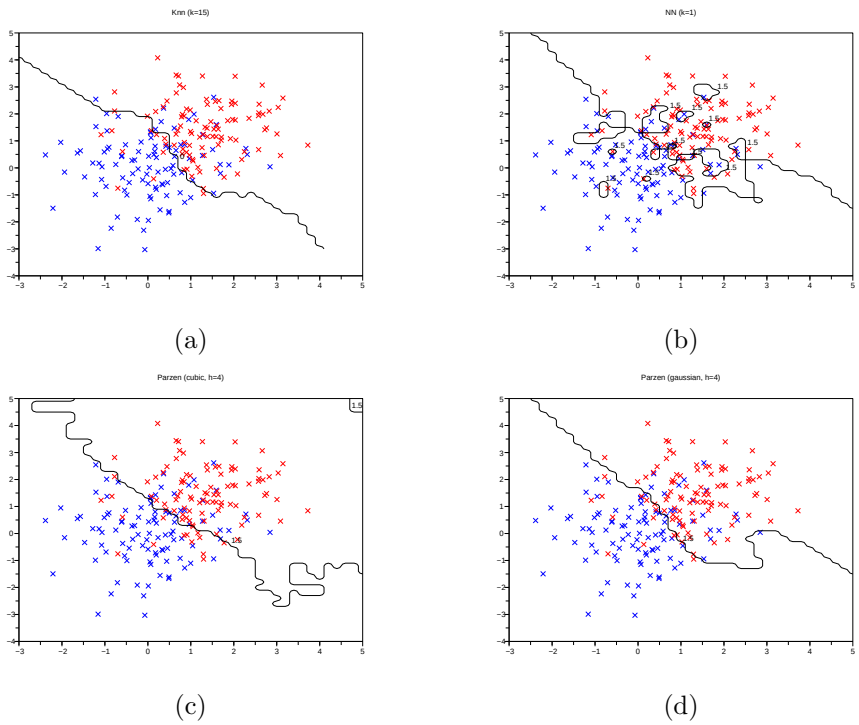
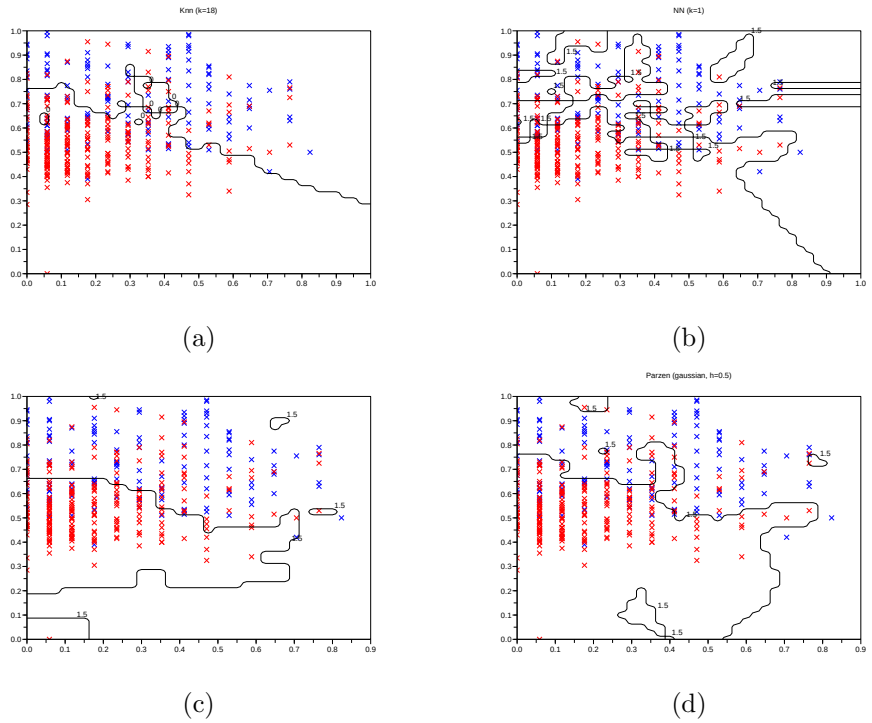


Figure 8: Decision surfaces for the classification methods using the synthetic gaussian dataset: (a) K-Nearest Neighbor, (b) Nearest Neighbor, (c) Parzen-Window with Hypercubic kernel and, (d) Parzen-Window with Gaussian kernel

result in better computational performance. This last fact *per se* would make a case for the Nearest Neighbor, since it's very efficient and can be effective if the points of the two classes are not mixed too much.

4 Conclusions

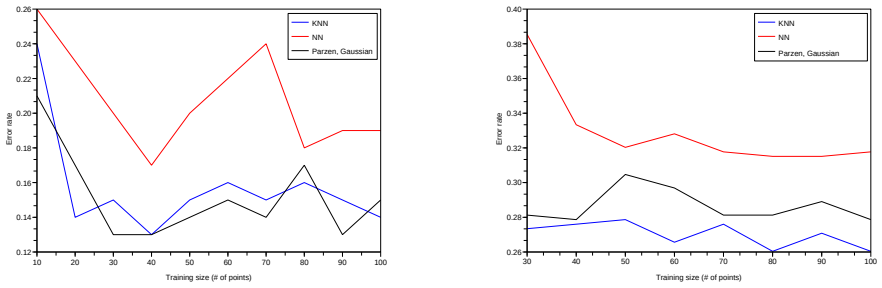
Throughout this report we made use of grid sampling and countor plots to better understand the decision surfaces of several classifiers as well as to select the best combination of parameters for the Support Vector Machines and Artificial Neural Network classifiers. As a general conclusion, the usage of a classifier is sensitive not only to the data layout but to the combination of the classifiers parameters used. A good choice of those parameters must be made before the classification of test data. Very complicated methods such as Artificial Neural Network classifiers don't seem to perform significantly better (in fact, they performed worse in our experiments) than simple methods such as the K-Nearest Neighbor classifiers. The engineer deploying classification methods in real-world problems should give preference to simple methods with intelligible



(a) (b)

(c) (d)

Figure 9: Decision surfaces for the classification methods using the diabetes dataset: (a) K-Nearest Neighbor, (b) Nearest Neighbor, (c) Parzen-Window with Hypercubic kernel and, (d) Parzen-Window with Gaussian kernel



(a) (b)

Figure 10: Performance comparison of the three classification methods considered on question 3 as the number of training samples for each class increases for (a) the synthetic gaussian dataset and, (b) the diabetes dataset

theoretical properties such as Support Vector Machines and kNN as opposed to the naive use of Artificial Neural Network classifiers.

```
function output = multigauss(mu,sig,alpha,n)
// generate n datatpoints from a Mixture of Gaussians with the means being columns
// in the matrix mu and the sigams being diagonal matrices with their diagonals
// in the colums of sig.  Alpha is the likelihood of each sub-cluster.
//
// mu - d x k
// sig - d x k
// alpha - 1 x k matrix

if(mtlb_any(size(mu)~=size(sig)))
  error('mu and sig are not the same size!')
end

if(size(mu,2)~=size(alpha,2))
  error('alpha must be a row vector with the same number of columns as mu')
end

if(abs(sum(alpha)-1)>0.0001)
  error('alpha should sum to one')
end

c = [cumsum(alpha)];
c(size(c,2))=1;

output = zeros(size(mu,1),n);
for i=1:n
  r = rand(1);

  ind = find(c>r);
  if(isempty(ind))
    error('alpha should sum to one. This should never happen');
  end
  ind = ind(1);
  if(ind>size(mu,2))
    error('index out of bounds.  This should never happen');
  end


  tmp = grand(1,'mn',mu(:,ind),diag(sig(:,ind)));
  output(:,i) = tmp;
end

endfunction

function output =  multiclassgauss(mu,sig,alpha,n)
// Generate n datatpoints from c randomly-selected clases, where each class is a Mix
ture of Gaussians with the means being columns
// in the matrix mu and the sigams being diagonal matrices with their diagonals
// in the colums of sig.  Alpha is the likelihood of each sub-cluster.
//
// Same as multigauss, except that mu,sig,and alpha are three-dimensional, with the
third
// dimensions representing which class is being represented.  Each class must
// have the same number of clusters, but some clusters may have probability 0 if
// desired.
//
// Additionally, the true class is returned as the "last" dimension of each datapoin
t.
// E.g. for
//
//  [ 2.1  3.9  2.2
//  [ 3.3  4.1  2.9
//  [  1    2
//
// c = size(mu,3) = # of classes
//
// mu - d x k x c
// sig - d x k x c
// alpha - 1 x k x c    mixture priors
if(mtlb_any(size(mu)~=size(sig)))
  error('mu and sig are not the same size!')
end
```

```
if(size(mu,2)~=size(alpha,2))
  error('alpha must be a deep (3d) row vector with the same number of columns as mu'
)
end

if(mtlb_any(abs(sum(alpha,2)-1)>0.0001))
  error('alpha should sum to one')
end

if(ndims(mu)<3)
  error('must use 3rd dimension')
end

nc = size(mu,3);
output = zeros(size(mu,1)+1,n);
for i=1:n
  r = rand(1);
  c = ceil(nc*r);
  tmp = multigauss(mu(:,:,c),sig(:,:,c),alpha(:,:,c),1);
  output(:,i) =  [tmp;c];
end

endfunction


function [data,parameters] = generatetestsets(d,k,c,m,n,scale)
// Generate m datasets of multiple-class, multiple-dimension,
// multiple-gaussian data.
//
// The means are generated uniformly within "scale"
//
// d - number of dimensions
// k - number of clusters in each class
// c - number of classes
// m - number of datasets
// n - number of samples from each dataset.
// scale - range of means
//
// data - (d+1) x n x m  -- the datapoints generated
//   the last row is the true clas of the generated point.
// parameters - [p x k x c x m] -- the parameters used to generate the data
//   where p is the number of parameters needed to specify a single cluster and is g
iven by
//      p = 2*d+1 (mean (d), sig (d), alpha (1))
//

  p = 2*d+1;
  data = zeros(d+1,n,m);
  parameters = zeros(p,k,c,m);
  for i=1:m
    mu = scale*rand(d,k,c);
    sig = ones(d,k,c);
    alpha = ones(1,k,c)/k;

    data(:,:,i) = multiclassgauss(mu,sig,alpha,n);
    parameters(:,:,:,i) = cat(1,mu,sig,alpha);
  end
endfunction

function testfisher()
// Run using Ctl-y to get results in general workspace.
  data = generatefisherdata()
  omega = fisherline(data);
  plotfishervssimple(data,omega)
endfunction

function data= generatefisherdata()
  d1 = grand(50,'mn',[1;1],[1 .9;.9 1]);
  d2 = grand(50, 'mn',[1;-1],[1 .8;.8 1]);
  data = [d1 d2;ones(1,50) 2*ones(1,50)];
endfunction

function omega = fisherline(data)
```

```
  // data - (d+1) x n  -- the labeled datapoints
  // omega - the fischer decision line.
  //   omega*x = 0 is the equation of the line.

  // omega = S_w^{-1}
  // x_0 = w^T (m_1+m_2)
  // S_w = \sum_{y \in class} (y-m_i)(y_i-m_i)^T

  // Assume 2 classes.
  d1 = data(1:2,data(3,:)==1);
  d2 = data(1:2,data(3,:)==2);
  m1 = mean(d1,2);
  m2 = mean(d2,2);
  n1=size(d1,2);
  n2=size(d2,2);
  diff1=(d1-m1*ones(1,n1));
  diff2=(d2-m2*ones(1,n2));
  S1 = diff1*diff1';
  S2 = diff2*diff2';
  Sw = S1+S2;
  omega = Sw\(m2-m1);
  omega = omega/sqrt(omega'*omega)
endfunction

function plotfishervssimple(data,omega)
// was function plotprettystuff(data,omega)
// Given the fisher projection line omega,
// plot the decision surface acording to omega
// and according to (m2-m1)

  // Assume 2 classes.
  d1 = data(1:2,data(3,:)==1);
  d2 = data(1:2,data(3,:)==2);
  m1 = mean(d1,2);
  m2 = mean(d2,2);

  clf
  plot(d1(1,:),d1(2,:),'r.')
  plot(d2(1,:),d2(2,:),'bx')
//  l1 = [m1 m2]; // line between means
//  plot(l1(1,:),l1(2,:),':')
  m0 = (m1+m2)/2; // midpoint
  plot(m0(1,:),m0(2,:),'*')
  //omega = whatever is input
  v2 = [omega(2);-omega(1)]
  p1 = m0-3*v2;
  p2 = m0+3*v2;
  l2 = [p1 p2];
  plot(l2(1,:),l2(2,:),'g-')

  omega2 = [m2-m1];
  omega2 = omega2/sqrt(omega2'*omega2);
  v2 = [omega2(2);-omega2(1)]
  p1 = m0-3*v2;
  p2 = m0+3*v2;
  l2 = [p1 p2];
  plot(l2(1,:),l2(2,:),'r--')
  prefix = "/home/yoderj";
  xs2eps(0,prefix+"/Desktop/tmp.eps",1);
endfunction

function fishererrorrate()
endfunction
```

```c
/*
        Fast Artificial Neural Network Library (fann)
        Copyright (C) 2003 Steffen Nissen (lukesky@diku.dk)

        This library is free software; you can redistribute it and/or
        modify it under the terms of the GNU Lesser General Public
        License as published by the Free Software Foundation; either
        version 2.1 of the License, or (at your option) any later version.

        This library is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
        Lesser General Public License for more details.

        You should have received a copy of the GNU Lesser General Public
        License along with this library; if not, write to the Free Software
        Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
        */

#include <stdio.h>

#include "fann.h"

#define isone(a) (fabs(a-1.0)<0.001)
typedef struct{
        int num_neurons_hidden;
        float desired_error;
        int max_epochs;
        int epochs_between_reports;
        int activation_function_hidden;
        int activation_function_output;
        char *trainfile;
        char *testfile;
} experiment;

void print_experiment_line(experiment config);
experiment *get_experiment_sequence(char *filename, int *num);
void read_experiment_line(experiment *config, FILE *file);
int get_num_experiments(FILE *file);

experiment *get_experiment_sequence(char *filename, int *num)
{
        FILE *fileptr;
        int i;
        experiment *ptr;
        if ((fileptr=fopen (filename,"r")) == 0)
        {
                *num =0;
                return 0;
        }
        else
        {
                printf ("get_num_experiments\n");
                *num = get_num_experiments(fileptr);
                printf ("allocate mem ( exp = %d)\n",*num);
                ptr = (experiment *) malloc ((*num)*sizeof(experiment));
                for (i=0; i< *num;i++)
                {
                        printf ("reading %d\n",i);
                        read_experiment_line (&ptr[i], fileptr);
                }
                fclose (fileptr);
                return ptr;
        }
}

int get_num_experiments(FILE *file)
{
        int number;
        fscanf(file,"%d\n",&number);
        return number;
}
```

```
void read_experiment_line(experiment *config, FILE *file)
{
        config->trainfile = (char *)malloc (20*sizeof(char));
        config->testfile = (char *) malloc (20*sizeof(char));
        fscanf (file,"%d %f %d %d %d %d %s %s\n",
                        &config->num_neurons_hidden,
                        &config->desired_error,
                        &config->max_epochs,
                        &config->epochs_between_reports,
                        &config->activation_function_hidden,
                        &config->activation_function_output,
                        config->trainfile,
                        config->testfile);
        return;
}

void print_experiment_line(experiment config)
{
        return;
}

int ann_experiment_driver (int num_layers,
                int num_neurons_hidden,
                float desired_error,
                int max_epochs,
                int epochs_between_reports,
                int activation_function_hidden,
                int activation_function_output,
                char *trainfile,
                char *testfile,
                FILE *outputptr)
{
        //const unsigned int num_layers = 3;
        //const unsigned int num_neurons_hidden = 32;
        //const float desired_error = (const float) 0.0001;
        //const unsigned int max_epochs = 300;
        //const unsigned int epochs_between_reports = 10;
        //
        struct fann *ann;
        struct fann_train_data *train_data, *test_data;

        long ncorrect=0, nincorrect=0;
        fann_type *expected_output, *actual_output;
        unsigned int i = 0;

        fprintf(outputptr, "Creating network.\n");

        train_data = fann_read_train_from_file(trainfile);

        ann = fann_create_standard(num_layers,
                        train_data->num_input, num_neurons_hidden, train_data->num_o
utput);

        fprintf(outputptr, "Training network.\n");

        fann_set_activation_function_hidden(ann, FANN_SIGMOID_SYMMETRIC_STEPWISE);
        fann_set_activation_function_output(ann, FANN_SIGMOID_STEPWISE);

        //fann_set_training_algorithm(ann, FANN_TRAIN_INCREMENTAL);

        fann_train_on_data(ann, train_data, max_epochs, epochs_between_reports, desi
red_error);

        fprintf(outputptr, "Testing network.\n");

        test_data = fann_read_train_from_file(testfile);

        fann_reset_MSE(ann);
        for(i = 0; i < fann_length_train_data(test_data); i++)
        {
                fann_test(ann, test_data->input[i], test_data->output[i]);

                actual_output = fann_run(ann, test_data->input[i]);
```

```
                expected_output = test_data->output[i];


                printf ("actual_output[0]=%f, expected_output[0]=%f, isone=%d\n", ac
tual_output[0], expected_output[0], isone(expected_output[0]));
                if (((actual_output[0] - actual_output[1]) > 0 && isone(expected_out
put[0]))||
                                                        ((actual_output[0] - actual
_output[1]) < 0) && !isone(expected_output[0]))
                {
                                printf ("correct!\n");
                                ncorrect++;
                }
                else
                {
                                printf("incorrect!\n");
                                nincorrect++;
                }

        }

        fprintf(outputptr, "MSE error on test data: %f\n", fann_get_MSE(ann));

        fprintf(outputptr, "Correct outputs: %d\n", ncorrect);
        fprintf(outputptr, "Incorrect outputs: %d\n", nincorrect);

        fprintf(outputptr, "Saving network.\n");


        fprintf(outputptr, "Cleaning up.\n");
        fann_destroy_train(train_data);
        fann_destroy_train(test_data);
        fann_destroy(ann);

        return 0;
}

int main (int *argc, char **argv)
{
        experiment *exp_vector;
        int num_experiments;
        int i;
        FILE *outputptr;
        outputptr = fopen (argv[2],"w");
        exp_vector = get_experiment_sequence (argv[1], &num_experiments);

        fprintf (outputptr, "The num of experiments is %d\n", num_experiments);

        for (i=0; i< num_experiments; i++)
        {
                fprintf (outputptr,"Experiment %d :", i);

                fprintf (outputptr,"%d %f %d %d %d %d %s %s\n",
                                exp_vector[i].num_neurons_hidden,
                                exp_vector[i].desired_error,
                                exp_vector[i].max_epochs,
                                exp_vector[i].epochs_between_reports,
                                exp_vector[i].activation_function_hidden,
                                exp_vector[i].activation_function_output,
                                exp_vector[i].trainfile,
                                exp_vector[i].testfile);

                ann_experiment_driver (3,
                                exp_vector[i].num_neurons_hidden,
                                exp_vector[i].desired_error,
                                exp_vector[i].max_epochs,
                                exp_vector[i].epochs_between_reports,
                                exp_vector[i].activation_function_hidden,
                                exp_vector[i].activation_function_output,
                                exp_vector[i].trainfile,
                                exp_vector[i].testfile,
                                outputptr);
        }
```

```
        fclose (outputptr);
        return 0;
}
```

```
// compare parzen window and kNN methods

function [data_train, data_test, label_train, label_test] = create_gaussian_data_set
 (mean1, mean2, cov1, cov2, ntrain, ntest)
  class1=grand(ntrain,"mn",mean1, cov1);
  class2=grand(ntrain,"mn",mean2, cov2);
  label_train = [ones(1,ntrain), ones(1,ntrain)*2];
  test1=grand(ntest,"mn",mean1, cov1);
  test2=grand(ntest,"mn",mean2, cov2);
  label_test = [ones(1,ntest), ones(1,ntest)*2];
  [data_train] = merge_from_two_classes (class1, class2, label_train);
  [data_test] = merge_from_two_classes (test1, test2, label_test);
endfunction

function [class1, class2] = convert_from_labeled_dat (data, label)
  class1 = data(:, label == 1);
  class2 = data(:, label == 2);
endfunction

function plot_classes ( data, label)
  [class1, class2] = convert_from_labeled_dat (data, label)
  plot(class1(1,:), class1(2,:), 'bx');
  plot(class2(1,:), class2(2,:), 'rx');
endfunction

function y = load_diabetes_file(filename)
  prefix='/home/yoderj/Desktop/'
  path = prefix+filename
 [fp,err] = mopen(path,'r');
 if(err)
   printf('error! Could not open '+path+'\n');
   disp(error);
   return;
 end

 y = mfscanf(-1,fp,'%f %f %f %f');
 mclose(fp);
endfunction

function [data_train, data_test, label_train, label_test] = load_diabetes(filename)

raw_train = load_diabetes_file('diabetes2.train');
raw_test = load_diabetes_file('diabetes2.test');

data_train = (raw_train(:,1:2))';
label_train = (raw_train(:,4)+1)';
data_test = (raw_test(:,1:2))';
label_test = (raw_test(:,4)+1)';
endfunction


function test_parzen_and_knn()
 //create 2D the datasets
//cov = eye (2,2);
//mean1=[0 0]'
//mean2=[1.5 1.5]'
//[data_train, data_test, label_train, label_test] = create_gaussian_data_set(mean1,
 mean2, cov, cov, 100, 50);

[data_train, data_test, label_train, label_test] = load_diabetes();

//plot data
//figure;
clf
plot_classes(data_train, label_train);
xlabel('dimension 1 (scaled)')
ylabel('dimension 2 (scaled)')
//xs2eps(0,"anon/xy_synth.eps",1);
xs2eps(0,"anon/xy_diabetes.eps",1);


//
//test parzen window for several values of h
```

```
// (for both kernels)
//
hs=[0.1 0.2 0.5 1 2 4];

//%% Plot 1
error_rates_cubic = ones (hs);
for i=1:length(hs)
  [predicted_labels] = test_parzen_window_class(hs(i), 'cubic', data_train, data_tes
t, label_train, label_test)
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_rates_cubic(i) = errorRate;
end
//figure
clf
plot(hs, error_rates_cubic,'b');
title("Parzen Window")
xlabel("Size of window")
ylabel("Error rate (fractional)")
xs2eps(0,"anon/parzen_diabetes.eps",1);

error_rates_gaussian = ones (hs);
for i=1:length(hs)
  [predicted_labels] = test_parzen_window_class(hs(i), 'gaussian', data_train, data_
test, label_train, label_test)
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_rates_gaussian(i) = errorRate;
end
plot (hs, error_rates_gaussian,'r');

legend ('hypercubic kernel', 'gaussian kernel');

//%% Plot 2
//
//test kNN for several values of k
// (for both kernels)
//
k=1:50;
error_rates_knn = ones (k);
for i=1:length(k)
  [predicted_labels] = knn(k(i),data_train,label_train,data_test);
   [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_rates_knn(i) = errorRate;
end
clf
plot (k, error_rates_knn,'r');

title ('KNN');
xlabel("k")
ylabel("Error rate (fractional)")
xs2eps(0,"anon/knn_diabetes.eps",1);

//
// From here we select the best k for the knn
// and the best h and kernel f for the parzen window
//
//[data_train, data_test, label_train, label_test] = load_diabetes();
k=19;
h=0.5;
parzen_kernel_type="gaussian";
sizes=[30:10:100];

error_size_parzen = ones (sizes);
error_size_knn = ones (sizes);
error_size_nn = ones (sizes);
for i=1:length(sizes)
//  [data_train, garbage1, label_train, garbage2] = create_gaussian_data_set(mean1,
mean2, cov, cov, sizes(i), 50);
  [data_train, garbage1, label_train, garbage2] = load_diabetes();
  data_train = data_train(:,1:sizes(i));
  label_train = label_train(:,1:sizes(i));
  //create_gaussian_data_set(mean1, mean2, cov, cov, sizes(i), 50);

  [predicted_labels] = knn(k,data_train,label_train,data_test);
```

```
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_size_knn(i) = errorRate;

  [predicted_labels] = knn(1,data_train,label_train,data_test);
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_size_nn(i) = errorRate;

  [predicted_labels] = test_parzen_window_class(h, parzen_kernel_type, data_train, d
ata_test, label_train, label_test)
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_size_parzen(i) = errorRate;
end

clf
plot (sizes, error_size_knn, 'b-');
plot (sizes, error_size_nn, 'r-');
plot (sizes, error_size_parzen, 'k-');
legend("KNN","NN","Parzen, Gaussian")
xlabel("Training size (# of points)");
ylabel("Error rate");
xs2eps(0,"anon/train_size_diabetes.eps",1);

endfunction

function compute_decision_surf()
//cov = eye (2,2);
//mean1=[0 0]'
//mean2=[1.5 1.5]'
//[data_train, data_test, label_train, label_test] = create_gaussian_data_set(mean1,
 mean2, cov, cov, 100, 50);
[data_train, data_test, label_train, label_test] = load_diabetes();


//[xx,yy] = meshgrid(-3:.2:5);
[xx,yy] = meshgrid(0:.025:1);
grid_points = [xx(:)';yy(:)'];

h=0.2;
  [grid_labels] = test_parzen_window_class(h, 'cubic', data_train, grid_points, labe
l_train, 2*ones(grid_points(1,:)));

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('Parzen (cubic, h=0.2)'+)
xs2eps(0,"anon/parzen_0_2_cubic_surface_diabetes.eps",1);


h=0.5;

  [grid_labels] = test_parzen_window_class(h, 'gaussian', data_train, grid_points, l
abel_train, 2*ones(grid_points(1,:)));

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('Parzen (gaussian, h=0.5)')
xs2eps(0,"anon/parzen_0_5_surface_diabetes.eps",1);

h=2;

  [grid_labels] = test_parzen_window_class(h, 'gaussian', data_train, grid_points, l
abel_train, 2*ones(grid_points(1,:)));

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('Parzen (gaussian, h=2)')
xs2eps(0,"anon/Desktop/parzen_2_surface_diabetes.eps",1);
```

```
k=18;

  [grid_labels] = knn(k,data_train,label_train,grid_points);

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2'-1.5,[0 10]);
title('Knn (k=18)')
xs2eps(0,"anon/knn_surface_diabetes.eps",1);


  [grid_labels] = knn(1,data_train,label_train,grid_points);

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('NN (k=1)')
xs2eps(0,"anon/nn_surface_diabetes.eps",1);

endfunction
```

```
function [y] = hypercubic_kernel(u)
  [rows, cols]=size(u);
  y=double(and(abs(u) < 1/2,'r'));
  //y=double(sum(abs(u) < 1/2,'r') == rows)
endfunction

function [y] = gaussian_kernel(u)
  [d,n] = size(u);
  y=zeros(1,n);
  for i=1:n
    y(i) = gaussian(u(:,i));
  end
endfunction

function [y] = gaussian(u)
  u=u(:);
  d = length(u);
  y = exp(-(u'*u)/2)/((2*%pi)^(d/2));
endfunction

function [pn] = gauss_parzen_window_dens(h, u, v)
  [d, n] = size (u);
  hn=h/sqrt(n);
  phi = gaussian_kernel((u - v*ones(1,n))/hn)
  pn = sum(phi)/(hn);
endfunction

function [pn] = cubic_parzen_window_dens(h, u, v)
    [d, n] = size(u);
    V = h^d;
    phi = hypercubic_kernel((u - v*ones(1,n))/h);
    pn = sum(phi)/(n*V);
endfunction

function [pn] = parzen_window_estimate(h, u, v, kernel_type)
  if (kernel_type == 'gaussian')
    [pn] = gauss_parzen_window_dens (h, u, v);
  else
    if (kernel_type == 'cubic')
      [pn] = cubic_parzen_window_dens (h, u ,v);
    end
  end
endfunction

function [class, p1, p2]=run1()
  cov=eye(2,2);
  mean1=[2 2]';
  mean2=[0 0]';
  class1=grand (500,'mn',mean1, cov);
  class2=grand (500,'mn',mean2, cov);
  [class, p1, p2] = clickable_experiment (class1, class2, 'cubic');
endfunction

function [class, p1, p2] = clickable_experiment (class1, class2, kernel_type)
  clf
  plot (class1(1,:), class1(2,:), 'bx');
  plot (class2(1,:), class2(2,:), 'rx');
  legend ('class1','class2');
  v = xclick();
  v = v(2:3)';
  plot (v(1),v(2),'k*');
  [class, p1, p2] = parzen_window_classifier (1,class1, class2, v, kernel_type);
endfunction

function [class, p1, p2]=parzen_window_classifier (h, class1, class2, v, kernel_type
)
  p1 = parzen_window_estimate (h, class1, v, kernel_type);
  p2 = parzen_window_estimate (h, class2, v, kernel_type);
  if (p1 >= p2)
    class = 1;
  else
    class = 2;
  end
```

```
endfunction

function [class1, class2] = convert_from_labeled_dat (data, label)
  class1 = data (:, label == 1);
  class2 = data (:, label == 2);
endfunction

function [data] = merge_from_two_classes (class1, class2, label)
  [d1, n1]= size (class1);
  [d2, n2]= size (class2);

  data = zeros (d1, n1+n2);

  data(:,label ==1) = class1;
  data(:,label ==2) = class2;
endfunction

function [predicted_labels] = test_parzen_window_class(h, kernel_type, data_train, d
ata_test, label_train, label_test)
  [class1_train, class2_train] = convert_from_labeled_dat (data_train, label_train);
  [d, n] = size (label_test);
  predicted_labels = zeros (1,n);
  for i=1:n
    [class, p1, p2] = parzen_window_classifier (h, class1_train, class2_train, data_
test(:,i), kernel_type);
    predicted_labels(:,i) = class(:);
  end
endfunction
```

```
function [PredictedLabels] = knn(k,TrainPattern,TrainLabel,TestPattern)
// Inspired by the MIT version.
// Ouput variables initialisation (not found in input variables)
//
// Input:
// k - 1 x 1 - number of neighbors to consider
// TrainPattern - d x N - Training vectors
// TrainLabel - 1 x N - Labels of the classes. Values 1 and 2
// TestPattern - d x numTests - Testing vectors
//
// ...where:
// d - the number of dimensions
//
// Output:
// PredictedLabels - 1 x numTests
//
// Please note the differences from the MIT version of the script:
//   * This is for scilab, not matlab
//     + I didn't translate the movie code
//     + mtlb_XXX functions were automatically translated by scilab
//   * I use transposed inputs when compared with MIT.
//     e.g. MyTrainPattern = MITTrainPattern'
//     this makes vector operations more natural for me
//   * Some of the code is a bit more vectorized now.
//   * The best class is computed by voting within the k nearest neighbors.
//     In a case of a tie, I assume class1.

PredictedLabels=zeros(1,size(TestPattern,2));

// Display mode
mode(0);

// Display warning for floating point exception
ieee(1);

//K-Nearest-Neighbor-Classifier MatLab Code


//k-nearest neighbor classifier

//Determines distances of all TrainPattern points from TestPattern points

//Outputs TrainLabel associated with nearest TrainPattern point

[numDims,numTests] = size(mtlb_double(TestPattern));

K = 40;

if(size(TrainPattern,2)<k)
  error('Must have at least as many training points as k, points='+size(TrainPattern
,2)+' k='+k);
end

// !! L.19: Matlab function moviein not yet converted, original calling sequence use
d
//M = moviein(K);

for numTest = 1:numTests(1,1)

  S = size(mtlb_double(TrainPattern));

  N = S(1,2);

  d = zeros(1,N);
  //creates specified space for distance column vector

//  set(gca(),"auto_clear","on");
//  //releases previous plot

//  plot(TestPattern(1,numTest),TestPattern(2,numTest),"k*");
//  //begins and holds new plot

//  title("Train Pattern Scatter Plot")
```

```
//  set(gca(),"auto_clear","off")

  for i = 1:N //creates distance column vector with N rows
   delta = TestPattern(:,numTest)-TrainPattern(:,i);
   d(1,i) = sqrt(delta'*delta);
  end;

//  // Select the two classes for ploting.
//  Train1 = TrainPattern(:,TrainLabel(1,:)<.5);
//  Train2 = TrainPattern(:,~(TrainLabel(1,:)<.5));

//  set(gca(),"auto_clear","off");
//  plot(Train1(1,:),Train1(2,:),"bx");
//  plot(Train2(1,:),Train2(2,:),"rx");
//  legend("test point","Class 0","Class 1");

  [cldvalues,clIndx] = mtlb_sort(d);// decreasing-order sort
 //clIndx = mtlb_fliplr(clIndx);
  // original translation: gsort
  // original matlab command: sortrows
  //determines closest distances and their indices

  CLTrainLabels=zeros(1,N);
//  CLTrainLabels(1,1:N)=TrainLabel(clIndx);

  Closest_Train_Labels = TrainLabel(clIndx(1:k));//CLTrainLabels(1,1:(k(1,1)));
  //displays """"kth"""" closest labels
//  disp (Closest_Train_Labels)
//  PredictedLabels = cell();
  //NCLTL = k;
  if(length(find(Closest_Train_Labels==1))>=k/2)
    PredictedLabels(1,numTest) = 1;
  else
    PredictedLabels(1,numTest) = 2;
  end

//  halt
//  pause
end;

//  visualizeResults(TrainPattern,TrainLabel,TestPattern,TestLabel,PredictedLabels);

disp("Done!")
endfunction

function visualizeResults(TrainPattern,TrainLabel,TestPattern,TestLabel,PredLabel)
// Plot Training data and Testing data with both true & predicted Classifications
// Computer number of correct and incorrect classifications.

  // Select the two classes for ploting.
  Train1 = TrainPattern(:,TrainLabel(1,:)<1.5);
  Train2 = TrainPattern(:,~(TrainLabel(1,:)<1.5));
  Test1 = TestPattern(:,PredLabel(1,:)<1.5);
  Test2 = TestPattern(:,~(PredLabel(1,:)<1.5));

  clf
  set(gca(),"auto_clear","off");
  plot(Train1(1,:),Train1(2,:),"bx");
  plot(Train2(1,:),Train2(2,:),"rx");
  plot(Test1(1,:),Test1(2,:),"bo");
  plot(Test2(1,:),Test2(2,:),"ro");

  Test1 = TestPattern(:,TestLabel(1,:)<1.5);
  Test2 = TestPattern(:,~(TestLabel(1,:)<1.5));
  set(gca(),"auto_clear","off");
  plot(Test1(1,:),Test1(2,:),"b*");
  plot(Test2(1,:),Test2(2,:),"r*");
  legend("Train Class 0","Train Class 1","Pred Class 0","Pred Class 1","True Class 0
","True Class 1");
  title('Training and Predicted Classifications with KNN")
  //legend("Train Class 0","Train Class 1","Test Class 0","Test Class 1");
```

```
   testResults(TestLabel,PredLabel)

endfunction

function [errorRate,correctClass,wrongClass] = testResults(TestLabel,PredLabel)
  // Count errors
  if or(size(TestLabel)~=size(PredLabel))
    error("Test and Pred Label Must be the same size!")
  end


  correctClass=length(find(TestLabel==PredLabel));
  wrongClass=length(find(TestLabel~=PredLabel));
  errorRate=wrongClass/length(TestLabel);
//  disp("Correct Class")
//  disp(correctClass)
//  disp("Wrong Class")
//  disp(wrongClass)
//  disp("Error Rate")
//  disp(errorRate)
endfunction
```

```
// compare parzen window and kNN methods

function [data_train, data_test, label_train, label_test] = create_gaussian_data_set
 (mean1, mean2, cov1, cov2, ntrain, ntest)
  class1=grand(ntrain,"mn",mean1, cov1);
  class2=grand(ntrain,"mn",mean2, cov2);
  label_train = [ones(1,ntrain), ones(1,ntrain)*2];
  test1=grand(ntest,"mn",mean1, cov1);
  test2=grand(ntest,"mn",mean2, cov2);
  label_test = [ones(1,ntest), ones(1,ntest)*2];
  [data_train] = merge_from_two_classes (class1, class2, label_train);
  [data_test] = merge_from_two_classes (test1, test2, label_test);
endfunction

function [class1, class2] = convert_from_labeled_dat (data, label)
  class1 = data(:, label == 1);
  class2 = data(:, label == 2);
endfunction

function plot_classes ( data, label)
  [class1, class2] = convert_from_labeled_dat (data, label)
  plot(class1(1,:), class1(2,:), 'bx');
  plot(class2(1,:), class2(2,:), 'rx');
endfunction

function y = load_diabetes_file(filename)
  prefix='/home/yoderj/Desktop/'
  path = prefix+filename
 [fp,err] = mopen(path,'r');
 if(err)
   printf('error! Could not open '+path+'\n');
   disp(error);
   return;
 end

 y = mfscanf(-1,fp,'%f %f %f %f');
 mclose(fp);
endfunction

function [data_train, data_test, label_train, label_test] = load_diabetes(filename)

raw_train = load_diabetes_file('diabetes2.train');
raw_test = load_diabetes_file('diabetes2.test');

data_train = (raw_train(:,1:2))';
label_train = (raw_train(:,4)+1)';
data_test = (raw_test(:,1:2))';
label_test = (raw_test(:,4)+1)';
endfunction


function test_parzen_and_knn()
 //create 2D the datasets
//cov = eye (2,2);
//mean1=[0 0]'
//mean2=[1.5 1.5]'
//[data_train, data_test, label_train, label_test] = create_gaussian_data_set(mean1,
 mean2, cov, cov, 100, 50);

[data_train, data_test, label_train, label_test] = load_diabetes();

//plot data
//figure;
clf
plot_classes(data_train, label_train);
xlabel('dimension 1 (scaled)')
ylabel('dimension 2 (scaled)')
//xs2eps(0,"anon/xy_synth.eps",1);
xs2eps(0,"anon/xy_diabetes.eps",1);


//
//test parzen window for several values of h
```

```
// (for both kernels)
//
hs=[0.1 0.2 0.5 1 2 4];

//%% Plot 1
error_rates_cubic = ones (hs);
for i=1:length(hs)
  [predicted_labels] = test_parzen_window_class(hs(i), 'cubic', data_train, data_tes
t, label_train, label_test)
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_rates_cubic(i) = errorRate;
end
//figure
clf
plot(hs, error_rates_cubic,'b');
title("Parzen Window")
xlabel("Size of window")
ylabel("Error rate (fractional)")
xs2eps(0,"anon/parzen_diabetes.eps",1);

error_rates_gaussian = ones (hs);
for i=1:length(hs)
  [predicted_labels] = test_parzen_window_class(hs(i), 'gaussian', data_train, data_
test, label_train, label_test)
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_rates_gaussian(i) = errorRate;
end
plot (hs, error_rates_gaussian,'r');

legend ('hypercubic kernel', 'gaussian kernel');

//%% Plot 2
//
//test kNN for several values of k
// (for both kernels)
//
k=1:50;
error_rates_knn = ones (k);
for i=1:length(k)
  [predicted_labels] = knn(k(i),data_train,label_train,data_test);
   [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_rates_knn(i) = errorRate;
end
clf
plot (k, error_rates_knn,'r');

title ('KNN');
xlabel("k")
ylabel("Error rate (fractional)")
xs2eps(0,"anon/knn_diabetes.eps",1);

//
// From here we select the best k for the knn
// and the best h and kernel f for the parzen window
//
//[data_train, data_test, label_train, label_test] = load_diabetes();
k=19;
h=0.5;
parzen_kernel_type="gaussian";
sizes=[30:10:100];

error_size_parzen = ones (sizes);
error_size_knn = ones (sizes);
error_size_nn = ones (sizes);
for i=1:length(sizes)
//  [data_train, garbage1, label_train, garbage2] = create_gaussian_data_set(mean1,
mean2, cov, cov, sizes(i), 50);
  [data_train, garbage1, label_train, garbage2] = load_diabetes();
  data_train = data_train(:,1:sizes(i));
  label_train = label_train(:,1:sizes(i));
  //create_gaussian_data_set(mean1, mean2, cov, cov, sizes(i), 50);

  [predicted_labels] = knn(k,data_train,label_train,data_test);
```

```
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_size_knn(i) = errorRate;

  [predicted_labels] = knn(1,data_train,label_train,data_test);
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_size_nn(i) = errorRate;

  [predicted_labels] = test_parzen_window_class(h, parzen_kernel_type, data_train, d
ata_test, label_train, label_test)
  [errorRate,correctClass,wrongClass] = testResults(label_test,predicted_labels);
  error_size_parzen(i) = errorRate;
end

clf
plot (sizes, error_size_knn, 'b-');
plot (sizes, error_size_nn, 'r-');
plot (sizes, error_size_parzen, 'k-');
legend("KNN","NN","Parzen, Gaussian")
xlabel("Training size (# of points)");
ylabel("Error rate");
xs2eps(0,"anon/train_size_diabetes.eps",1);

endfunction

function compute_decision_surf()
//cov = eye (2,2);
//mean1=[0 0]'
//mean2=[1.5 1.5]'
//[data_train, data_test, label_train, label_test] = create_gaussian_data_set(mean1,
 mean2, cov, cov, 100, 50);
[data_train, data_test, label_train, label_test] = load_diabetes();


//[xx,yy] = meshgrid(-3:.2:5);
[xx,yy] = meshgrid(0:.025:1);
grid_points = [xx(:)';yy(:)'];

h=0.2;
  [grid_labels] = test_parzen_window_class(h, 'cubic', data_train, grid_points, labe
l_train, 2*ones(grid_points(1,:)));

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('Parzen (cubic, h=0.2)'+)
xs2eps(0,"anon/parzen_0_2_cubic_surface_diabetes.eps",1);


h=0.5;

  [grid_labels] = test_parzen_window_class(h, 'gaussian', data_train, grid_points, l
abel_train, 2*ones(grid_points(1,:)));

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('Parzen (gaussian, h=0.5)')
xs2eps(0,"anon/parzen_0_5_surface_diabetes.eps",1);

h=2;

  [grid_labels] = test_parzen_window_class(h, 'gaussian', data_train, grid_points, l
abel_train, 2*ones(grid_points(1,:)));

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('Parzen (gaussian, h=2)')
xs2eps(0,"anon/Desktop/parzen_2_surface_diabetes.eps",1);
```

```
k=18;

  [grid_labels] = knn(k,data_train,label_train,grid_points);

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2'-1.5,[0 10]);
title('Knn (k=18)')
xs2eps(0,"anon/knn_surface_diabetes.eps",1);


  [grid_labels] = knn(1,data_train,label_train,grid_points);

clf
plot_classes(data_train, label_train);
grid_labels2 = matrix(grid_labels,size(xx));
contour(xx(1,:),yy(:,1),grid_labels2',[1.5 10]);
title('NN (k=1)')
xs2eps(0,"anon/nn_surface_diabetes.eps",1);

endfunction
```