HW 3: Non-parametric Techniques

[Anonymous]

April 15, 2008

Contents

1 Introduction

The purpose of this homework is to study non-parametric classification techniques. The first question discusses a parametric technique, Fisher's discriminant. Then the remaining sections discuss primarily non-parametric techniques: neural networks, support vector machines, Parzen windows, and the K-nearest neighbor approaches.

In questions two and three, we shall use grid approaches to explore two-dimensional spaces. In question 2, this space is the parameter space used for training the classifier. In question 3, this space is the data space from which the samples are drawn.

2 Question 1: Fisher's Discriminant

Fisher's Discriminant provides a heuristic to describe the separation between two classes when projected onto a linear one-dimensional subspace,

$$J(w) = \frac{w^T S_B w}{w^T S_w w}$$

where $S_b = (m_2 - m_1)(m_2 - m_1)^T$ and $S_w = \sum_{c \in classes} \sum_{y \in c} (x - m_c)(x - m_c)^T$. Fisher's discriminant then projects the data so as to minimize this metric. This minimizing projection is given by

$$w_0 = S_w^{-1}(m_2 - m_1)$$

A question raised in class is why the within-class variance, S_w, is necessary. Can we maximize

$$J(w) = \frac{w^T S_B w}{w^T w} = w^T S_B w$$

subject to the constraint that $||w|| = 1$ instead? This results in the fairly intuitive solution

$$w_{\|} = (m_2 - m_1)$$

which simply projects data onto a line parallel to the vector joining the means. To consider the case where this discriminant might be useful, we construct an artificial data set similar to one in Bishop's "Pattern Recognition and Machine Learning." We construct two classes in two dimensions, each with a high correlation between the variables, as illustrated in Figure 1. The two classes are constructed such that they lie very close together, but are still possible to separate. Furthermore, the means of the classes are carefully arranged so that projection onto the line connecting the means causes the classes to overlap. However, in the Fisher line, the classes hardly overlap at all.

How often do data-sets like this occur in practice? It's difficult to say without testing on real data or test data that is less contrived.

3 Question 2: Support Vector Machines and Neural Networks

For this section, we tested neural networks and support vector machines using the "Diabetes" data set, which contains 765 data points in eight dimensions. All the dimensions have been scaled to lie in the range [0 1]. Each point is classified in one of two classes. For neural networks, we used FANN (Fast Artificial Neural Networks). For support vector machines, we used LibSVM. Both programs are commonly used and have freely available source code.

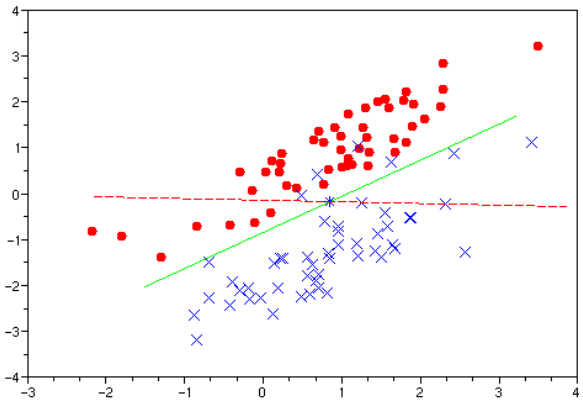


Figure 1: Example of Fisher's Linear Discriminant. The synthetic data is described in the text. The dashed line is the decision surface based on the vector between the means, The dotted line is the decision surface based on Fisher's Discriminant. The decision surfaces are perpendicular to the projection lines $w_{\|}^T x = 0$ and$w_0^T x = 0$, and both are defined to intersect the mid-point between the means.

3.1 Artificial Neural Networks

We use sigmoidal functions, which the default activation functions provided by FANN: . The output function was also a sigmoid function. We had two output nodes, representing the two possible classes. We trained the neural network to produce one for the output when that class was the correct label and zero for the output when that class was the incorrect layer. When testing the network, we selected the classification with the highest output to be the class chosen by the neural network. There are other ways to set up the output of the neural network, such as using a single output or a soft-max output. (The soft-max output forces the sum of the outputs to be one.) We believe our network has similar capabilities to these other output configurations.

3.1.1 Experiment 1

For our first experiment, we divided the data into a training set (384 points) and a testing set (384 points). We perform a validation search for the best parameters for the neural network using a 2D grid of parameters. There are two parameters which we attempted to vary: The number of hidden nodes and the MSE training error that we used to train the network. We set the maximum number of epochs high (30,000) so that the training error is always the limiting factor on training. We tried every combination of these two parameters, yielding the grid of test cases illustrated with red x's in Figure 2. Each point on the

grid represents a test using a specific number of hidden nodes and terminating parameters. For each test, we train the neural network with the specified number of hidden nodes, using as many epochs as are required to drive the error to the specified MSE error. We then validate the neural network with the testing data, predicting the class and comparing it with the true class from the test set. In this way, we compute the error rate for each combination in the grid. (Technically we should have used a separate cross-validation set, but it is too late to run the test again. It takes well over 8 hrs to complete.) The error rates are plotted using contours which represent equal error curves in Figure 2.

For our final test, we chose to use an MSE termination criterion of 0.13, which the MSE value of the points in the optimal band in Figure 2. We arbitrarily chose to use 100 hidden nodes since this did not seem to affect the result. When we trained the neural network on the training data with these parameters, it classified 284 of the testing points correctly and, 100 of the testing points incorrectly. Therefore, the overall error the neural network for this data is 73.9%

3.1.2 Experiment 2

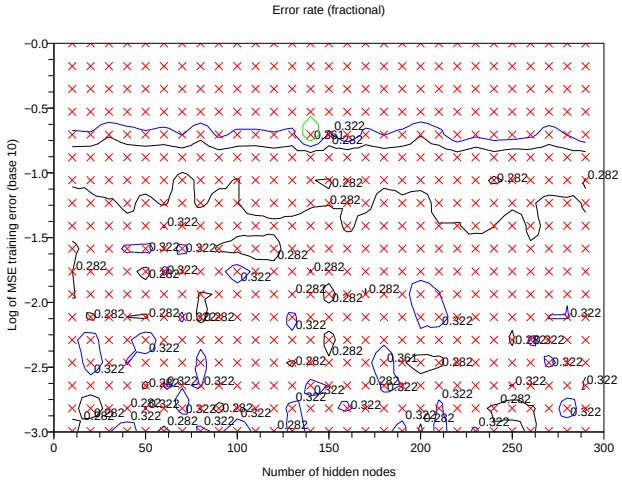
For our second experiment, we wished to find how the size of the training data and the number of hidden nodes affected the performance of the neural network. To accomplish this, we split the data into three parts, a training set (384 points), a cross-validation set (150 points), and a testing set (234 points). The cross-validation set is used to determine the best parameters for the neural network. Once these parameters are selected, the testing set may be used to give a final estimate of the error of the neural networks method.[1]

We again tested the network on a grid of data points. This time, each red x in Figure 3 represents an experiment performed with a given number of hidden units and training data points. For each point, we trained on the training set and validated the result on the validation set. The error rate on the validation is again shown using the contours in Figure 3. It is not too surprising that the error decreases as we increase the amount of training data. However, we were surprised to discover that changing the number of hidden nodes in the data set does not seem to have any predictable effect on the performance.

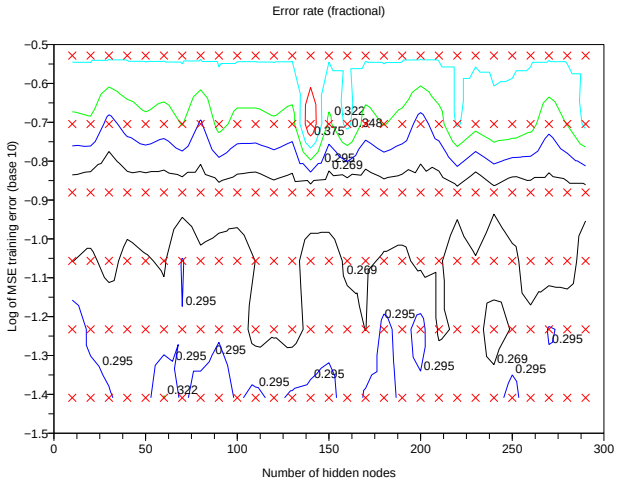
3.2 Support Vector Machines

The LibSVM package allows us to do grid experiments quite easily, with a python script included with the package called "easy.py" We highly recommend following the technique used by this script, even if the script itself is not used. Again, the data is divided into training, validation, and testing sets. Together, the training and validation sets have 384 points, and the testing set has the remaining 384 points. The experiment is run on a grid of possible values for

[1]In the end we did not use the testing set at all for Experiment 2, because Experiment 1 provides us with the final test for Neural Networks, so there is no need for a further test of the neural networks.



(a)



(b)

Figure 2: Variation of Error rate with training parameters. There is a sweet region near the MSE training error of 0.1 (that is, $\text{Log}_{10}MSE = -1$) where the error rate is minimum. It is visible in this graph as the region between the two lines marked 0.282. With higher error rates, the training does not go on long enough. With lower error rates, the neural network overfits the data. Curiously, the number of hidden nodes appears to have no effect on the validation error rate. (a) Full test (b) Close-up of best MSE values

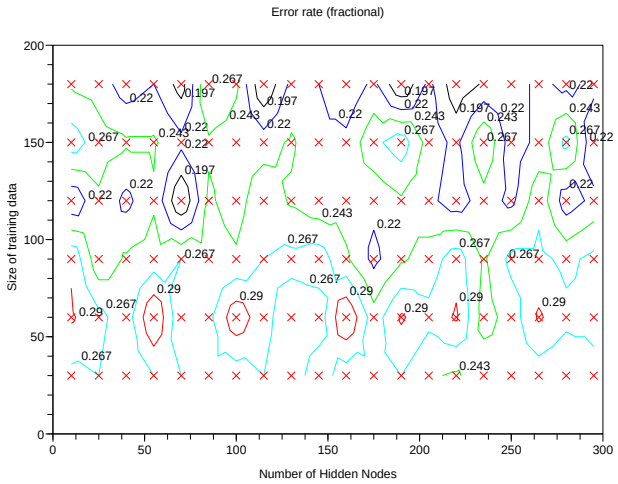


Figure 3: Lack of correlation between the size of the training set and the number of hidden nodes.

two of the parameters. These two parameters are C (The penalty parameter for misclassified points) and γ (the scale factor for the RBF kernel). The training data is actually split into training and validation data several times by the script, in a process known as cross-validation. An example grid with contour errors is given in Figure 4.

Based on these cross-validation runs, the best error rate on cross-validation occurs when $C = 2048$, and $\gamma = 0.000488 = \frac{1}{2048}$. When the SVM is trained with these parameters, it achieves a success rate of 80.5%.

3.3 Comparison of Artificial Neural Networks and Support Vector Machines

Naturally, given the differences between neural networks and support vector machines, it is impossible to say that one is always better than the other. However, if a choice had to be made based on our experience in this homework assignment, I would choose support vector machines. There are two reasons that I would like to do this.

First, support vector machines directly search for the decision surface which separates the data. A neural network is trained to produce a real-number output which must be then thresholded to give the classification. Seeking the decision surface directly, as the support vector machine does, gives an answer that leaves no question of interpretation.

Second, the parameters used by the support vector machine are easier to understand intuitively. It is difficult to understand the effect that an individual hidden node has on the output of a neural network. Indeed, we did not see any

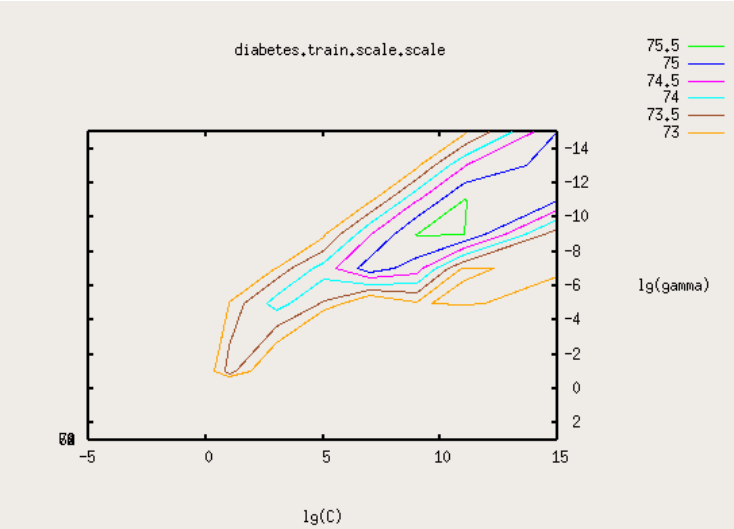


Figure 4: Training Grid for the Support Vector Machine. The contours represent the success rate in percent (higher success for the central contour).

change in performance in either of our experiments as we varied the number of hidden nodes. By contrast, both C and γ appear to be related to how mixed the two classes are, and it appears that the ideal values often satisfy $C = \frac{1}{\gamma}$.

So in the future, I will try a support vector machine first. However, it is important to note that support vector machines are designed for two-class problems. I'm not sure how well they extend to multi-class problems.

4 Question 3: Parzen Windows and KNN

In this question, we again use the diabetes data-set. We "project" the data to two dimensions by using the first two dimensions in the diabetes data-set. This allows us to visualize the results more easily.

4.1 Parzen Windows

We implemented Parzen windows using two windows functions, the cubic and Gaussian. We vary the size of the windows to test the effect on the data. The classification error is shown in Figure 5. In this figure, we see over-fitting for very small windows and under-fitting for larger windows.

Based on this original training error, we select the Gaussian window with a size of 0.5, and the cubic window with a size of 0.2 for future experiments. When selecting a window shape, we shall use the Gaussian, because it has an error rate that is consistently lower than the cubic window. This is intellectually

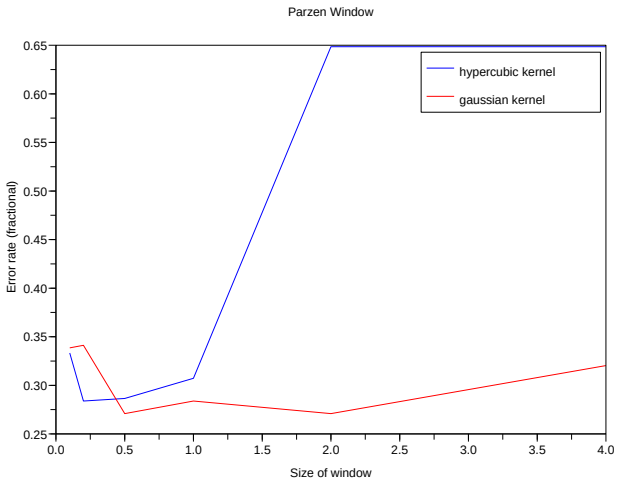


Figure 5: Classification error with Parzen windows as window size varies.

satisfying because the Gaussian window gives higher weight to those points which are closer and which we expect to relate more to the current point.

4.2 K-Nearest Neighbors (KNN)

To test K-nearest neighbors, we ran K-nearest neighbors on the testing data with values ranging from 1 to 50, as seen in Figure 6. To my surprise, KNN continues to give better results as we increase k, even up to k=18. Beyond this point, we considered the gains to be marginal, so we selected k=18 for use in future experiments.

4.3 Variation with Training Size

How do Parzen Windows and KNN improve as we increase the amount of data for training? We test the performance of both methods on data sizes ranging from 30 to 100 points, as shown in Figure 6. It is interesting to note that the very simple method of KNN gives a similar error to the Neural Networks approach, even when it is using only the first two dimensions of the data!

4.4 Decision Surface

We compute the decision surfaces for some of the non-parametric methods discussed in this question (Question 3). These are shown in Figure 8. We compute the decision surfaces by classifying each point in a grid with spacing 0.025 in each dimension. We then draw the contour that separates the points in one class from the other using a standard contour plotting function.

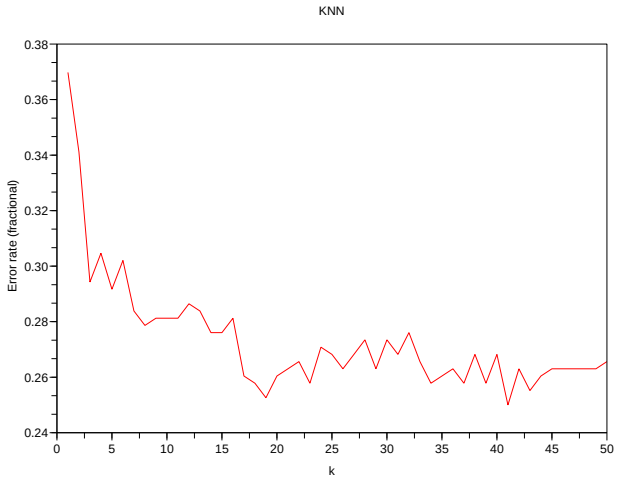


Figure 6: Classification error with KNN as k varies.

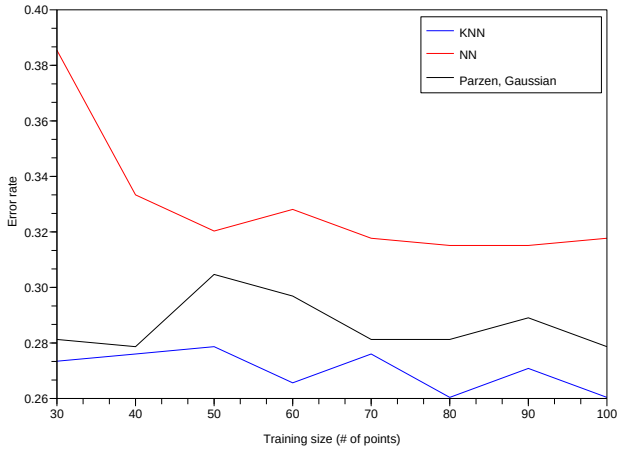


Figure 7: Improvement of Parzen Windows and KNN with training size. Nearest Neighbors has the highest error. KNN (k=18) has the lowest error. Parzen Windows (h=0.5) gives an error in the middle. NN improves the most as we increase the amount of data. The other two methods do not appear sensitive to the amount of training data.6

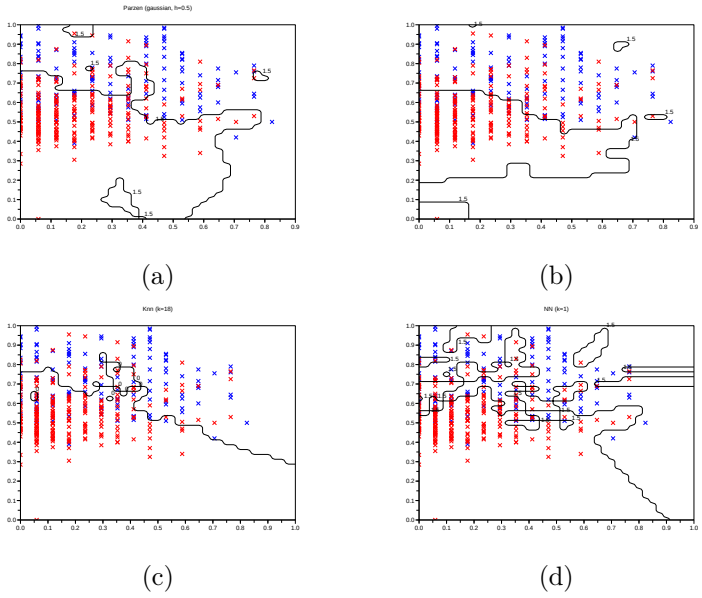


Figure 8: Decision Surfaces for Non-Parametric Methods. (a) Parzen window, Gaussian, h=0.5 (b) Parzen window, cubic, h=0.2 (c) K-nearest neighbors, k=18, (d) nearest neighbors

It is interesting to see the overfitting showing up so clearly in the nearest neighbors decision surface (Fig. 8 (d)), as many little islands around each point in the data. If there is overfitting in the decision surfaces for the Parzen window methods, it is harder to see. However, we can see that the Parzen window method defaults to the blue dot class when there are no data points within the window.

4.5 Comparison of Parzen Window and KNN techniques

The KNN technique gives better performance on the diabetes set than the Parzen windows technique. I believe this is because the K nearest neighbors automatically adapts the size of its window to the density of the data in the neighborhood of the query point. Given its simplicity and accuracy, KNN is a great method to try on data as a first technique. It may well outperform more advanced methods like Support Vector Machines. However, it is important to note that the Nearest Neighbor technique (k=1) does not share the success properties of KNN. NN will always overfit the training data; by definition it achieves perfect classification when applied to the training data. Not surprisingly, it performed the worst of the methods in Question 3.

5 Conclusion

In this homework, we have experimented with grid sampling as a numerical technique. In Question 2, we found grid sampling of classifier parameter space to be an effective way to select a small number of parameters and to see the effect these parameters have on performance. In Question 3, we found grid sampling of the feature space could allow us to visualize the decision surfaces and the overfitting behavior of classifiers.

Although grid sampling techniques may seem inefficient at first, technology tends toward such techniques. Years ago, graphical displays were designed by scanning an electron beam in specially-designed paths to produce letters on the display. Who would have guessed that today all displays would draw letters by coloring them in one dot at a time on the screen? In a similar way, sampling parameter and feature spaces is a simple and effective way to do computations for low-dimensional spaces.

```
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
```