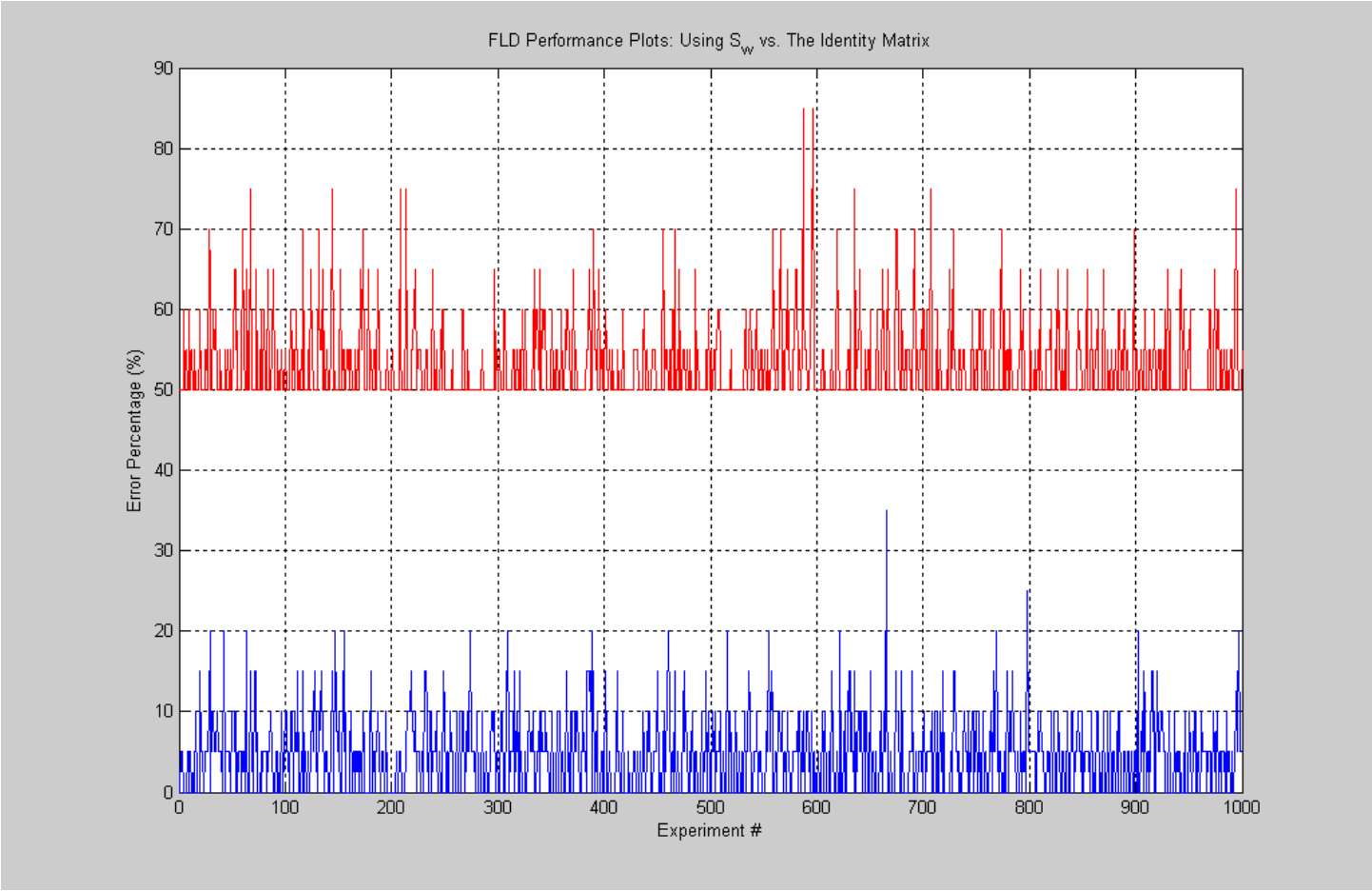
PROBLEM 1

The number of experiments was 1000. The MATLAB codes are attached. To simplify things the same set of data were used for training and testing purposes.



From the graph it is obvious that substituting the identity matrix for S_w will not get us a better, or even a similar performance. In fact, the error percentage difference is massive for the identity matrix. We can then conclude that maximizing the numerator of J(w) in the Fisher's Linear Discriminant equation is not a possible solution.

PROBLEM 2

Support Vector Machine

Advantages:

Extremely powerful non-linear classifier

Disadvantages:

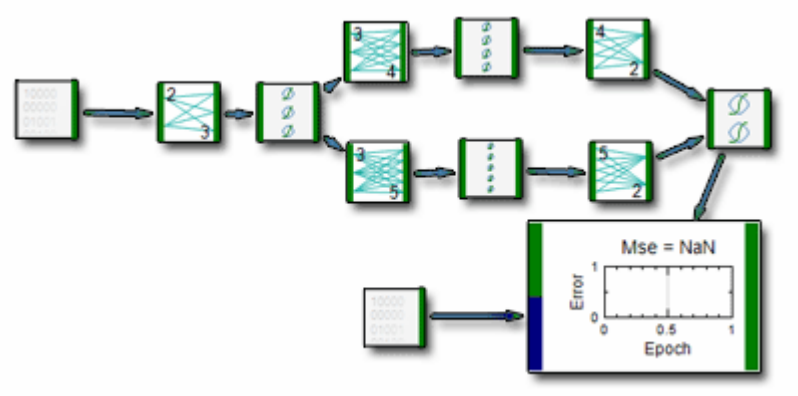
Computationally demanding to train and to run

Sensitive to noisy data

Choice of kernel function and the parameters have to be set manually and can greatly impact the results

Modular Neural Network

In this case we will be using a System deployed from Synapse that looks like this:



Advantages:

Powerful non-linear classifier

Handles noisy data well

Fast to run

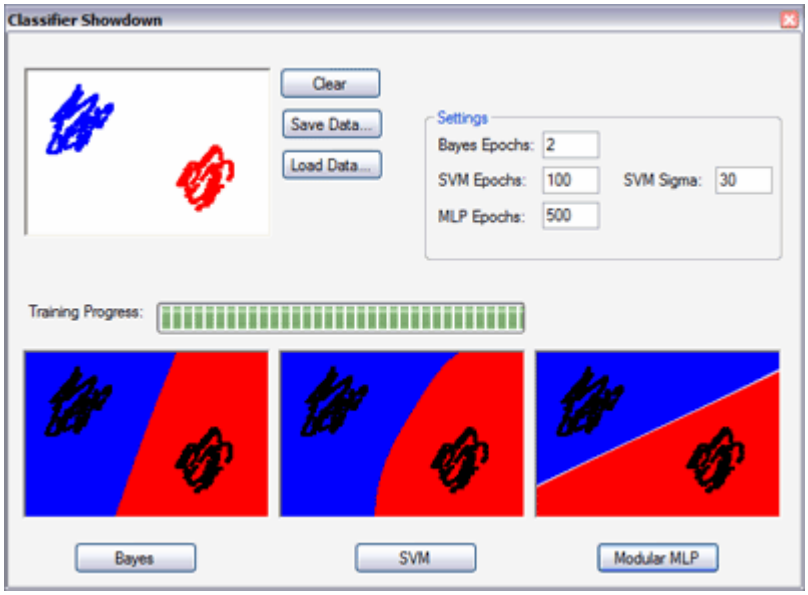
Disadvantages:

Computationally demanding to train

Has difficulty with more complex boundaries

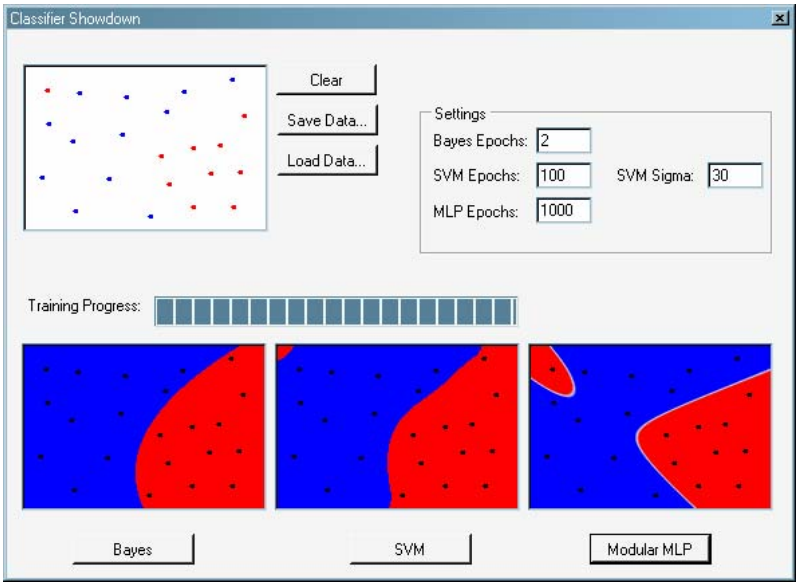
Experiments:

Simple separable dataset:



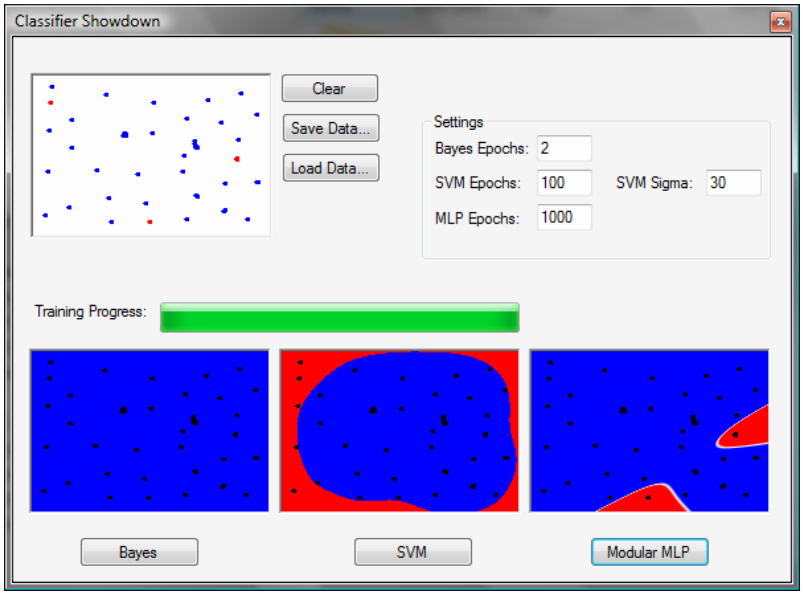
As we can see all classifiers handle that without any problems, but with various solutions.

Outlier:



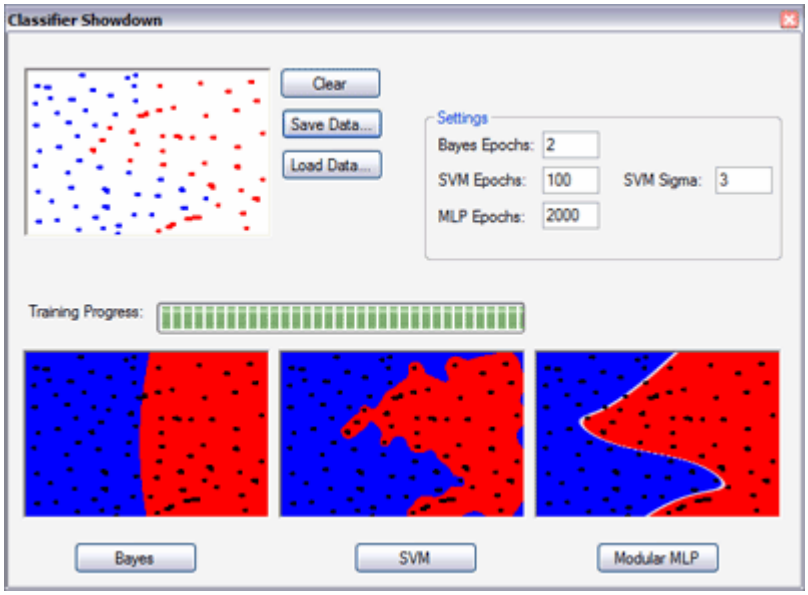
MLP classifies the outlier correctly, while SVM fails.

More: Multiple outliers:

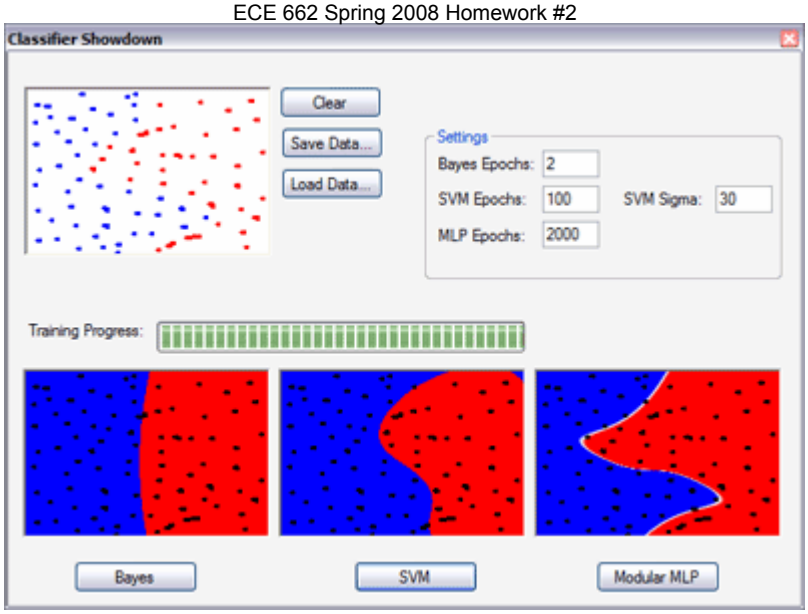


MLP manages to detect the outlier but not everything. SVM fails again.

Standard dots pattern:

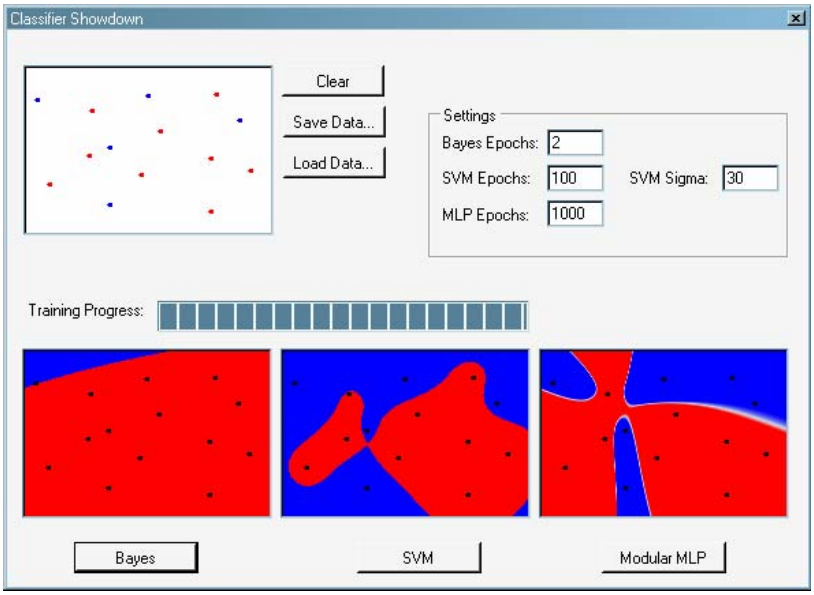


The sigma is set to 3, which gives a very tight fit around the data points. The MLP neural net on the other hand gives a good generalization (and actually draws the regions as expected). Suppose we now increase the sigma by a factor ten to 30:



The solution is more general, but far less accurate. Basically, the more the classifier is trained, the more it will learn the training set and hence increase its resolution at the cost of generalization.

Now a random non-separable dataset:



The SVM is clearly better than the MLP when it comes to handling random data.

Conclusions

The Support Vector Machine is complex, slow, takes a lot of memory, but is an immensely powerful classifier. SVMs are generally unsuitable for larger trainings sets as their speed of execution decreases with the size of the training set. However, its superiority in the classification of complex patterns should not be underestimated.

The MLP neural network is good at capturing fairly complex patterns while keeping good generalization capabilities. While it is slow to train, it is fast to use and its execution speed is independent of the size of the data it was trained on. It is very well suited for more complex real-world problems and thus on average superior to the SVM classifier.

Adapted from http://blog.peltarion.com/2006/07/10/classifier-showdown/

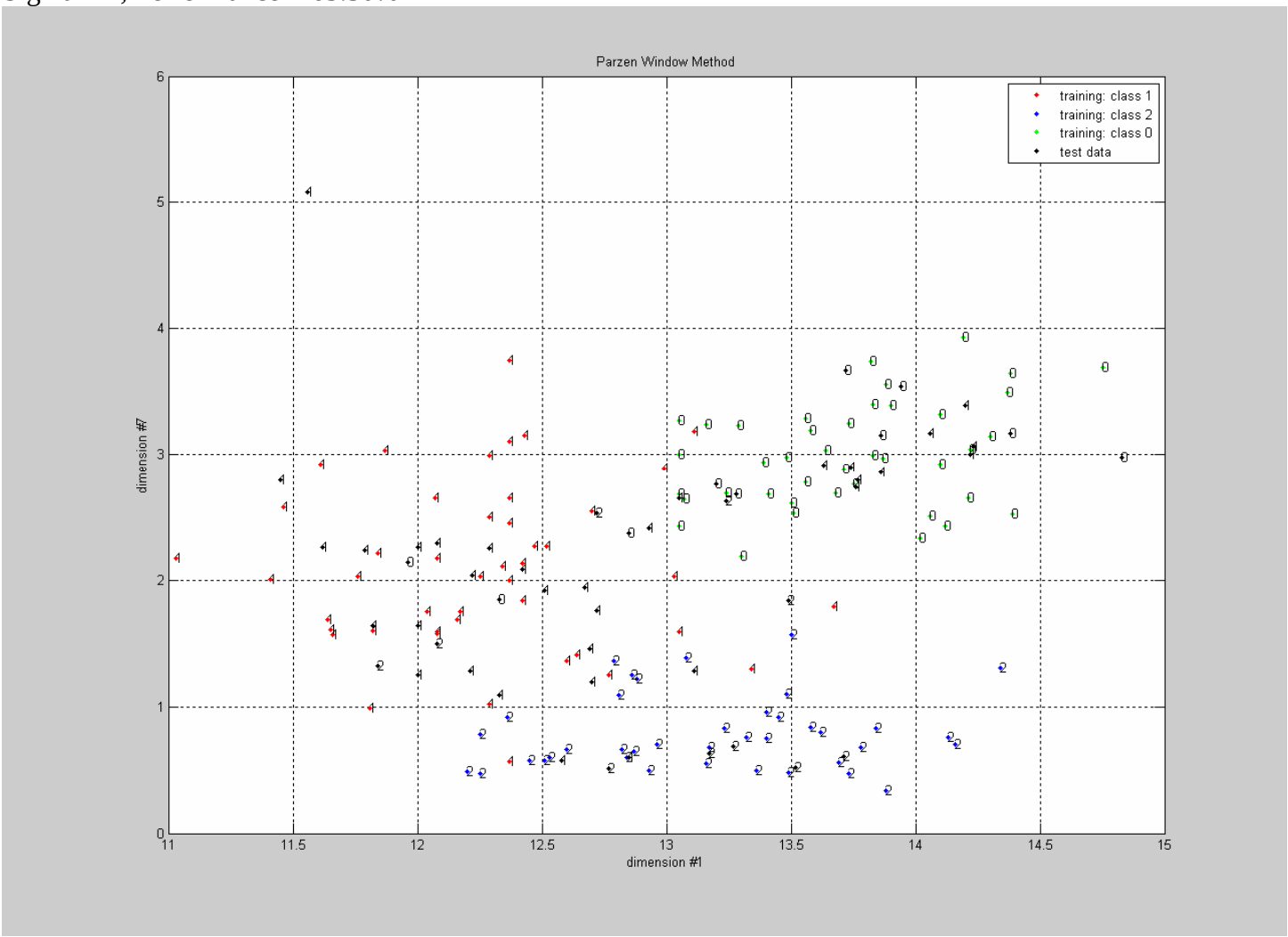
PROBLEM 3

The data consisted of 70% training and 30% test. The number of experiments was 1000. Performance indicates the mean percentage of correct classified data over 1000 experiments.

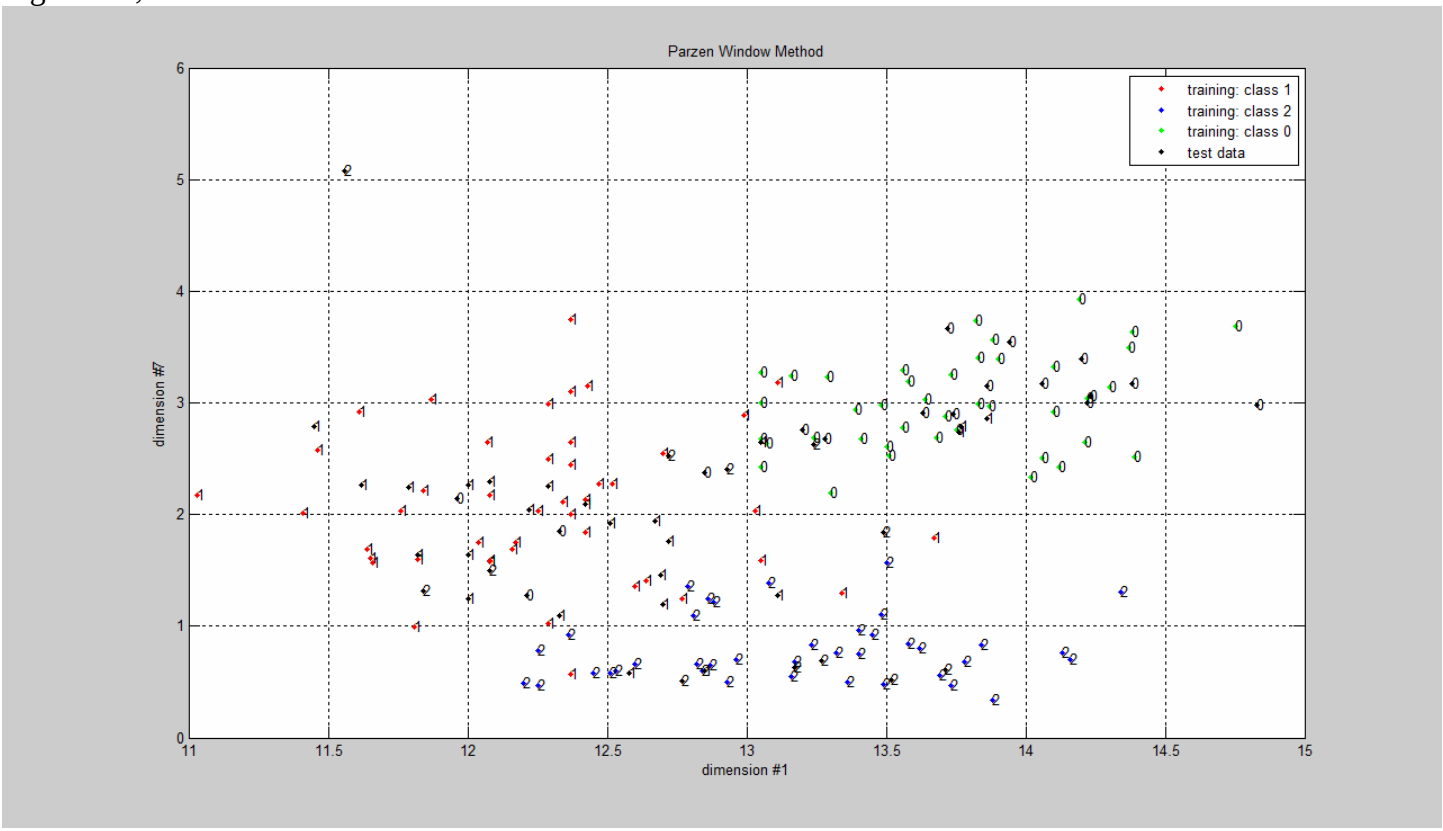
Parzen Window

The Parzen window technique estimates the probability defining a window (given the window size) and a function on this window. This computes the estimation of the probability function convolving the window function with the samples function.

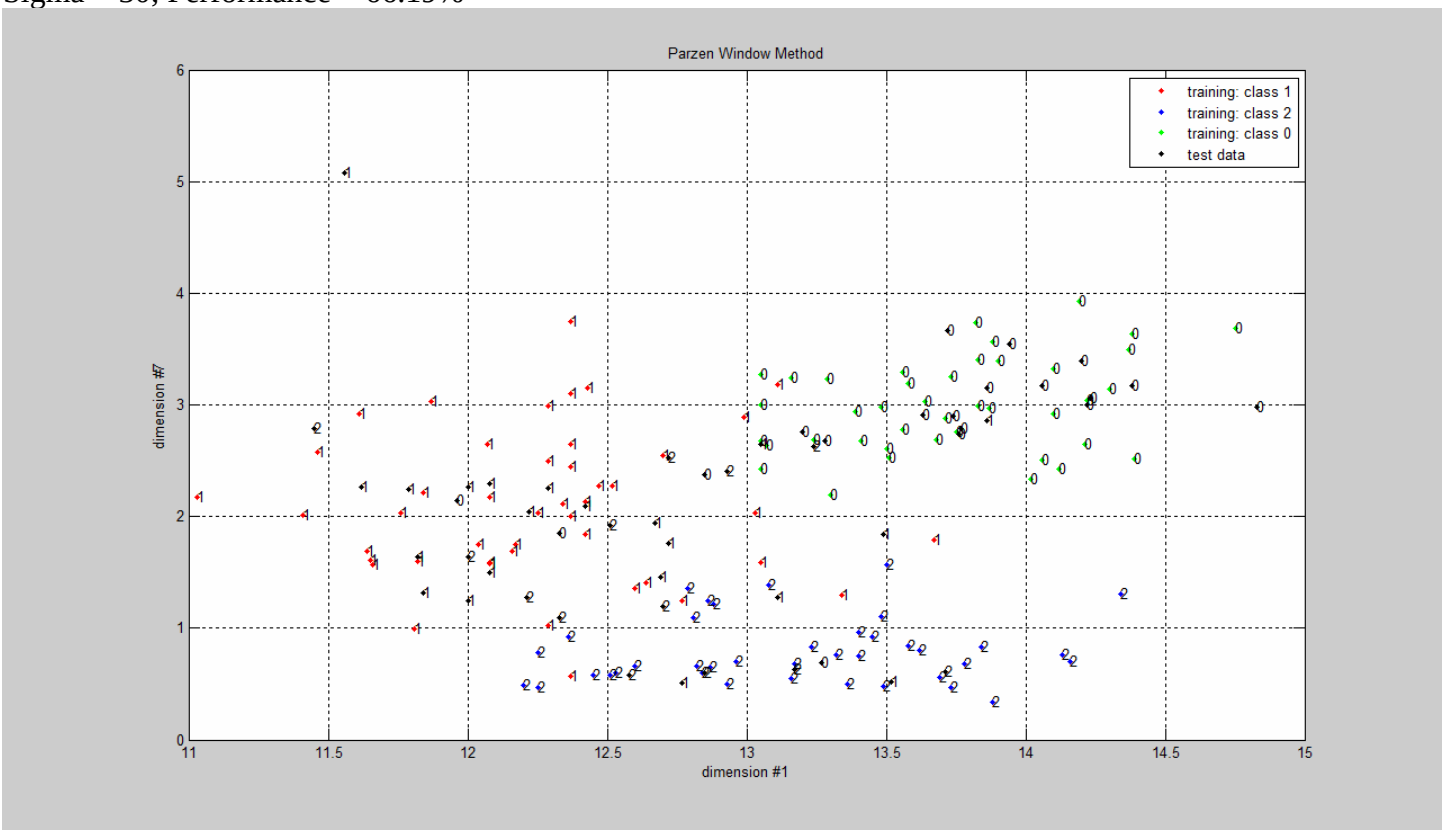
Sigma = 1; Performance = 63.50%



Sigma = 3; Performance = 71.54%



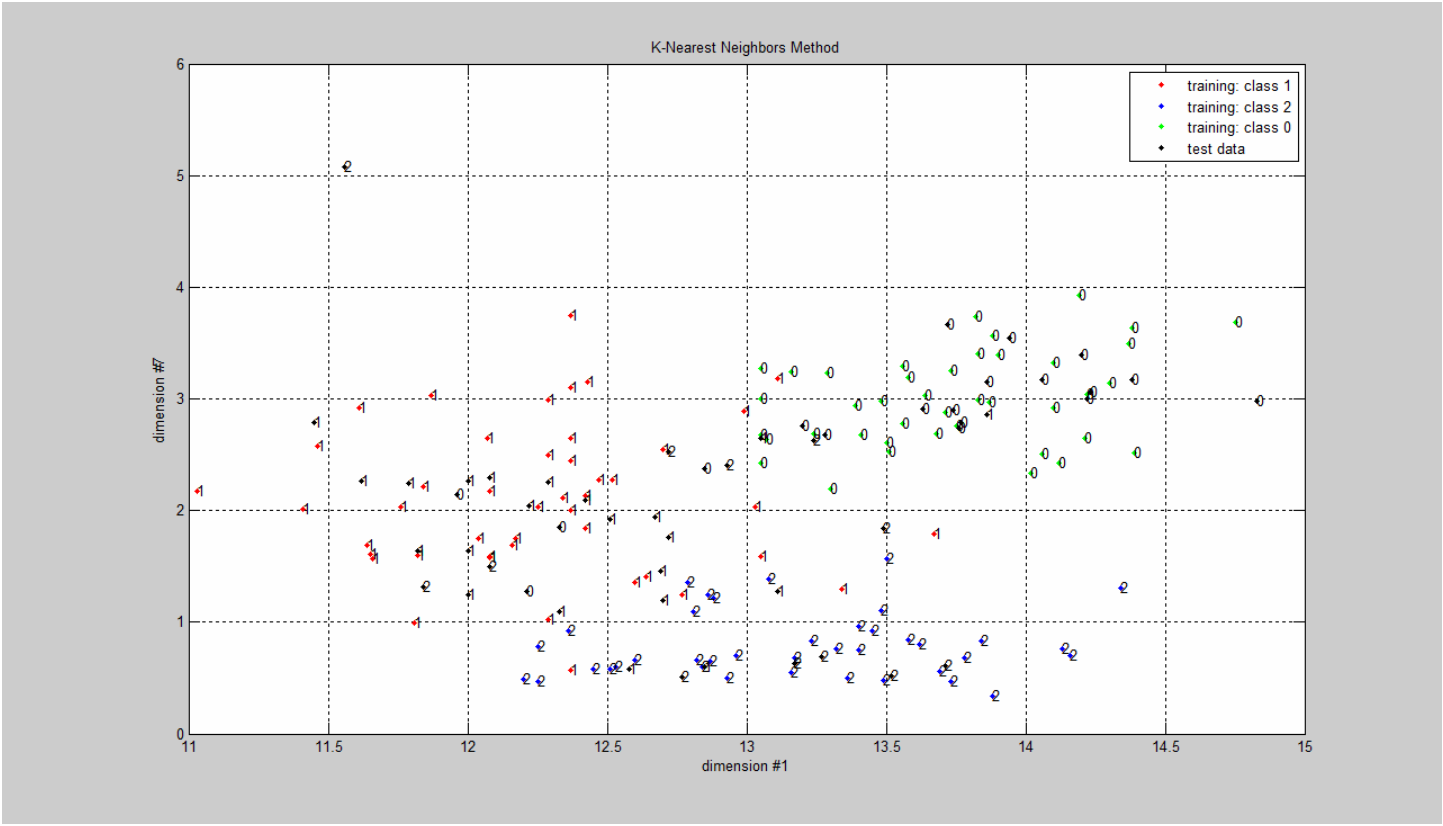
Sigma = 30; Performance = 66.19%



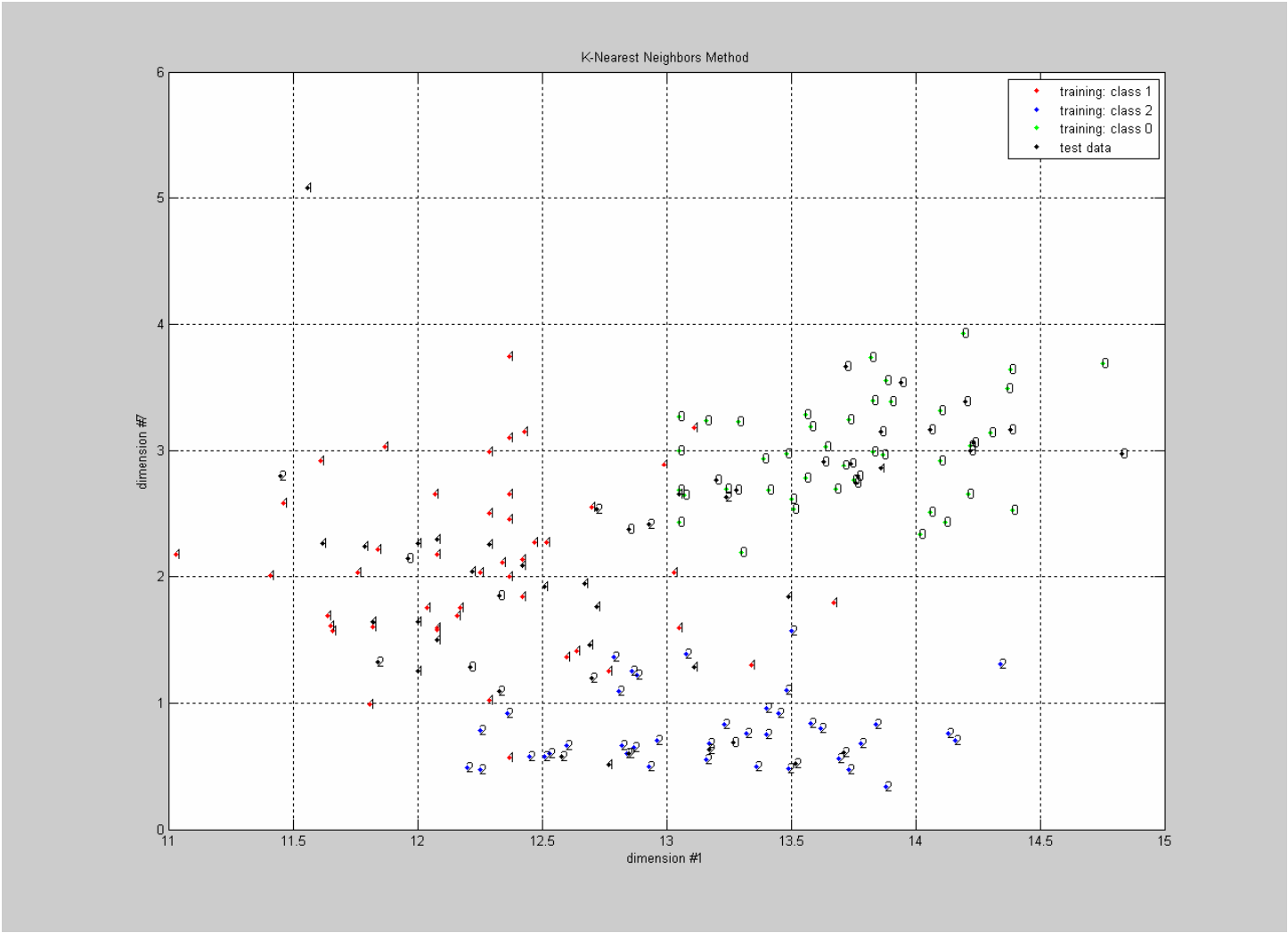
K-Nearest Neighbors

In this method, we consider a hypersphere centered on the point x with radius *r*. We increase the radius *r* until the hypersphere contains exactly *K* points. The class label *c*(x) is then given by the most numerous class within the hypersphere. This method is useful since classifications will be robust against outliers.

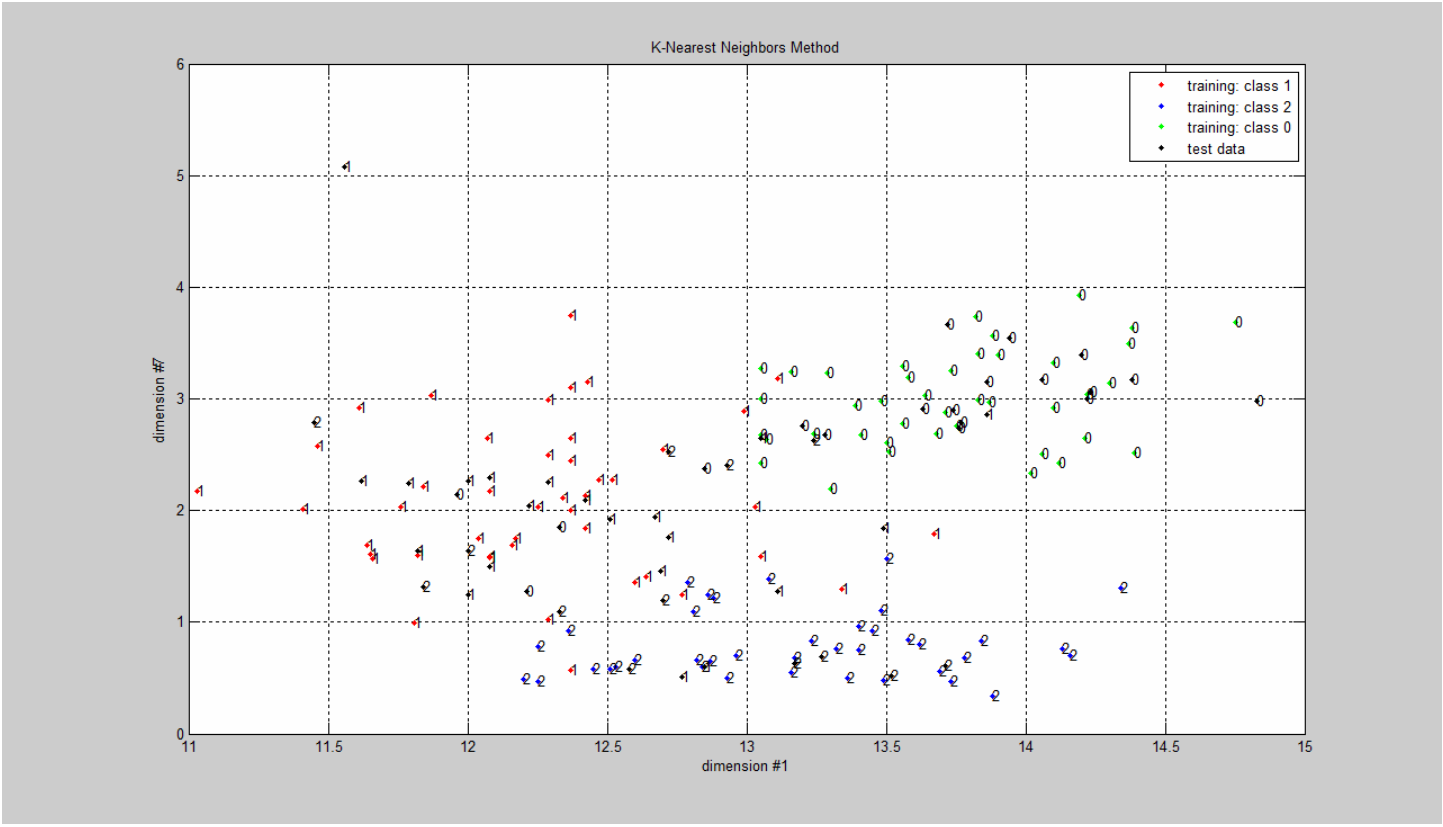
k = 2; Performance = 73.88%



k = 3, Performance = 71.05%

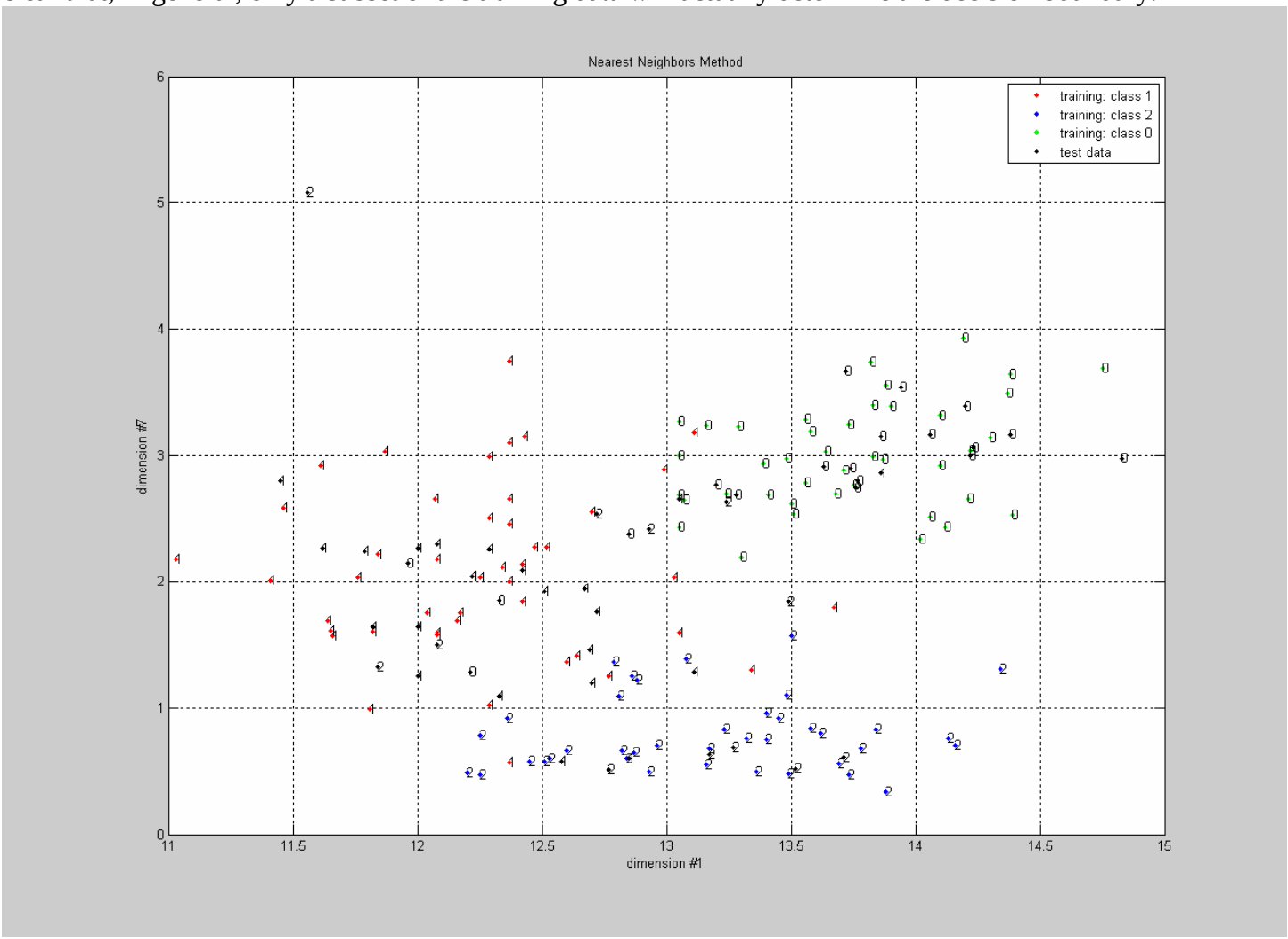


k = 5; Performance = 70.55%

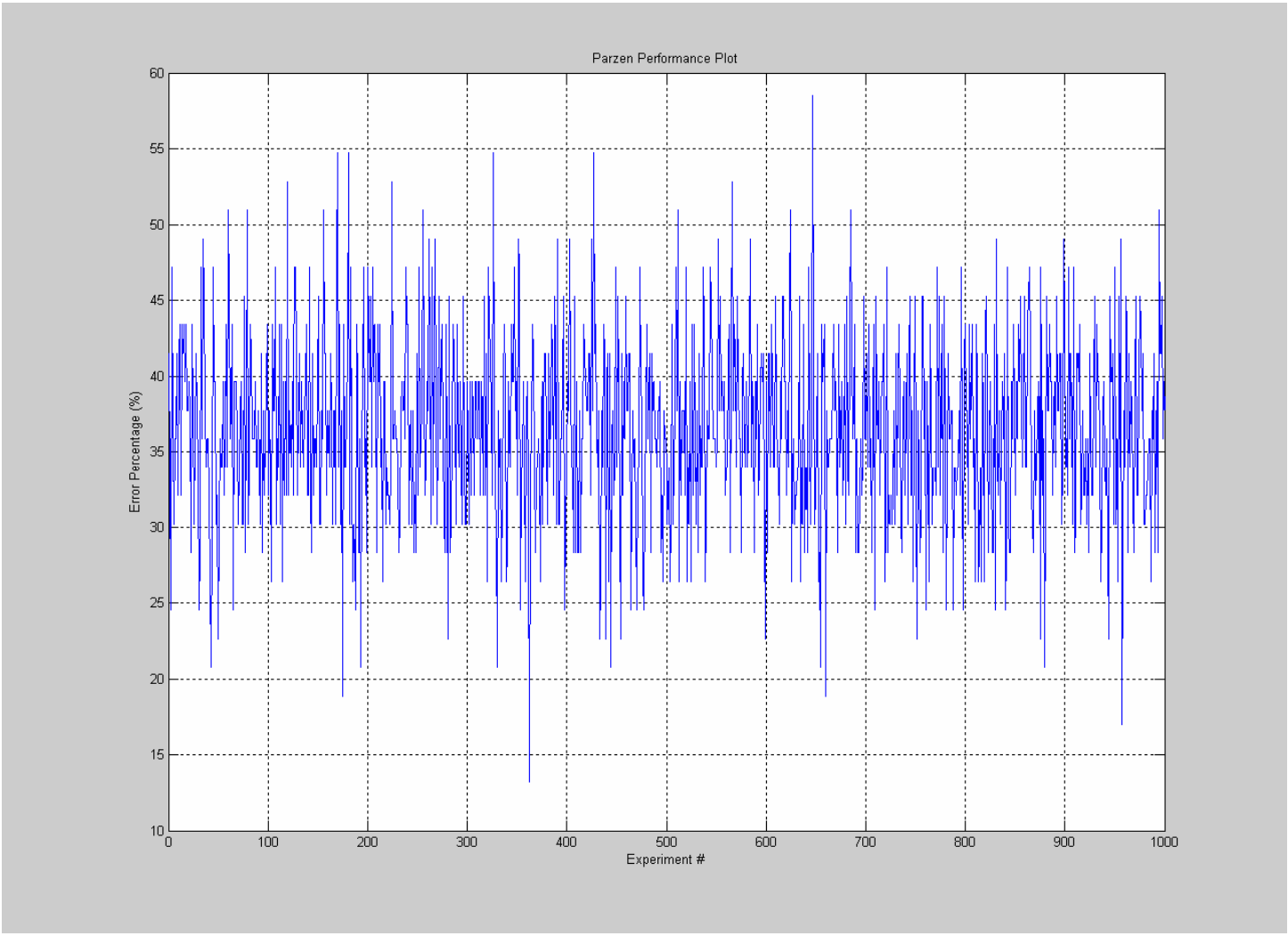


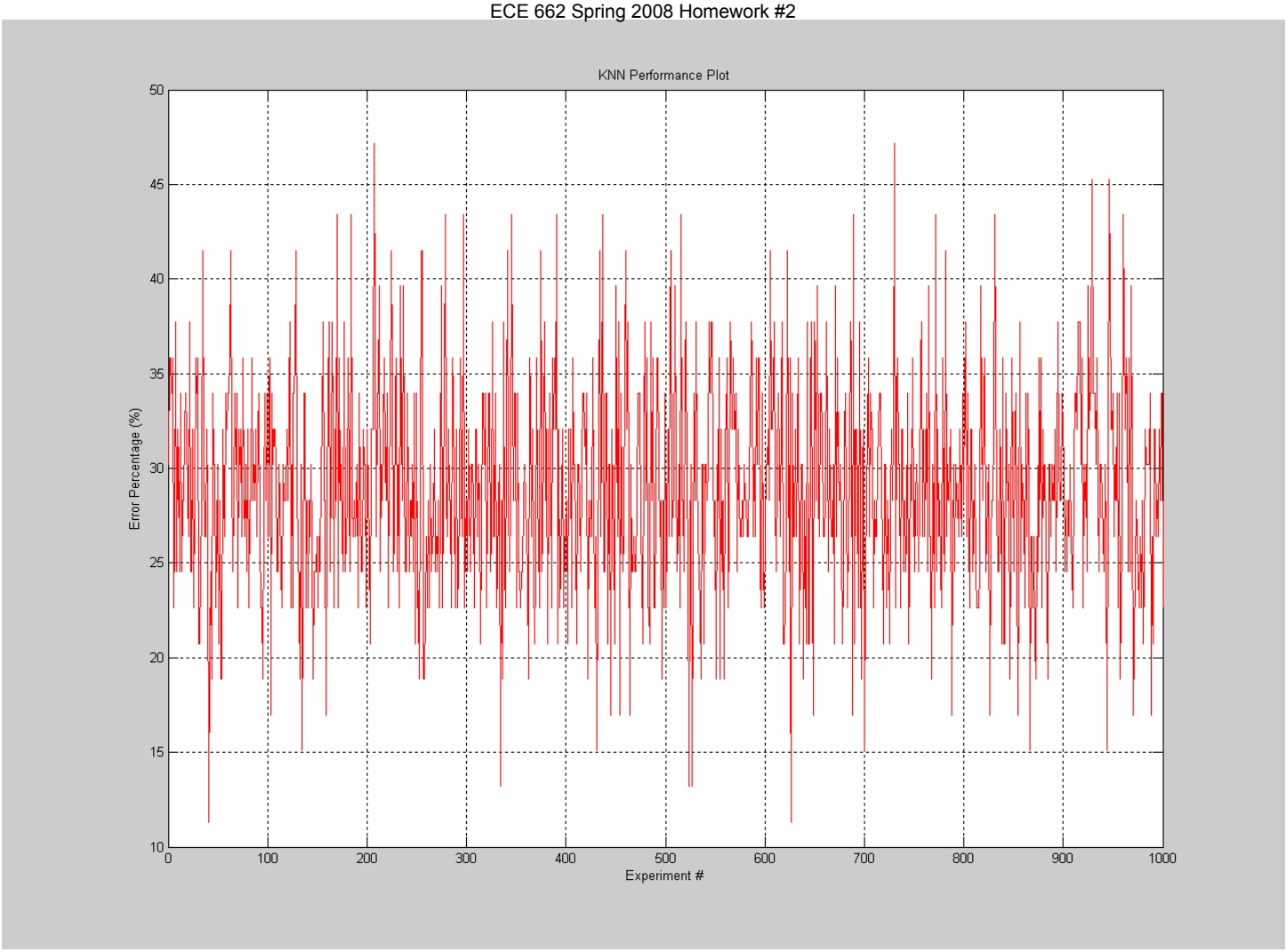
Nearest Neighbors

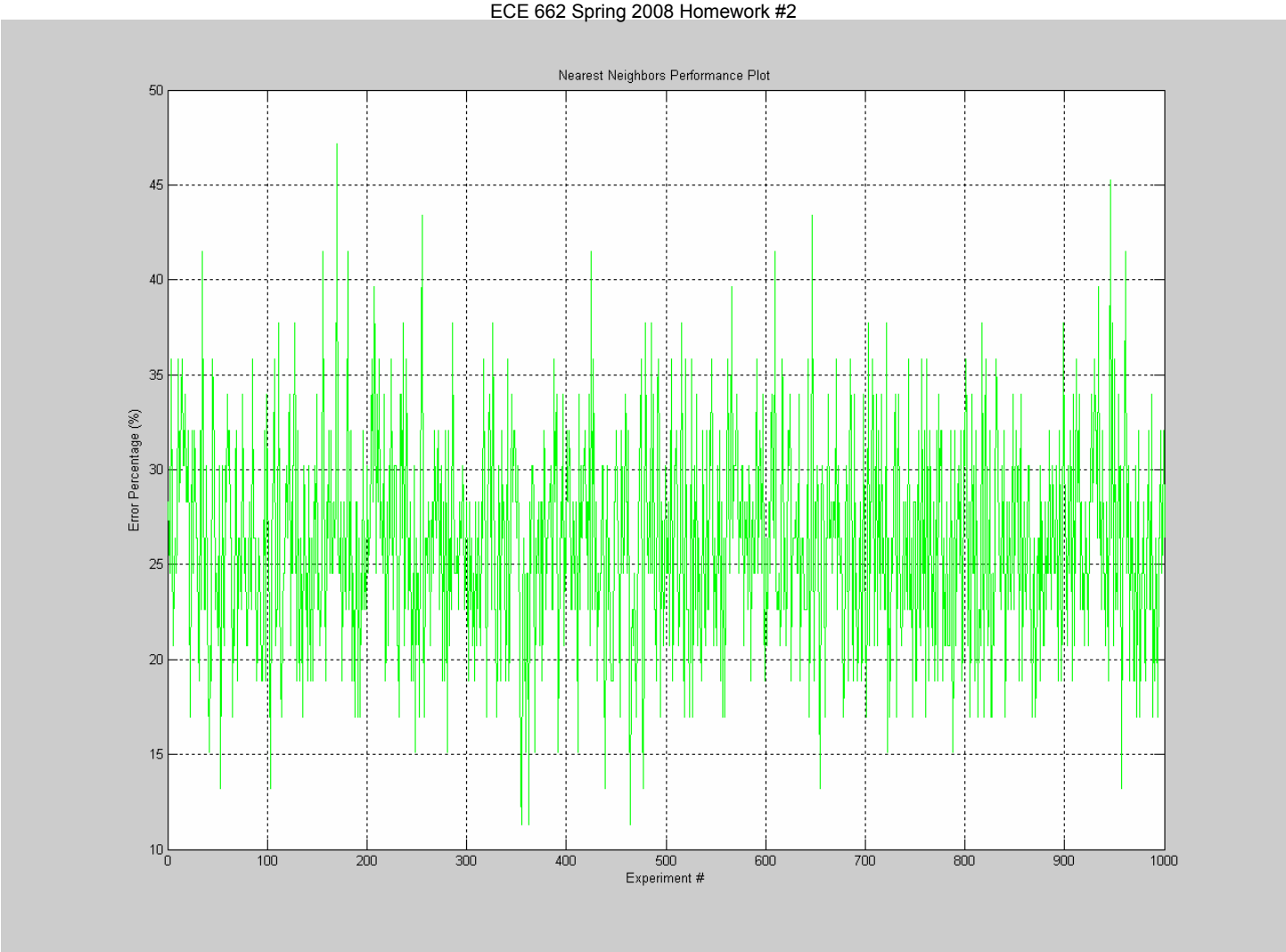
The Euclidian squares distance method is used here. The nearest neighbours algorithm is extremely simple yet rather powerful. We need to store the whole dataset in order to make a classification. However, it is clear that, in general, only a subset of the training data will actually determine the decision boundary.



Error Percentage Comparisons




KNN Performance Plot
Error Percentage (%)
Experiment #
50
45
40
35
30
25
20
15
10
0
100
200
300
400
500
600
700
800
900
1000



Comparing these three plots, KNN (k = 3) is slightly a better option than Parzen Window (sigma = 1) for classifying a general dataset. KNN performs better compared to NN according to the results obtained. In conclusion, the results agree with the fact that we are considering more neighboring points in KNN compared to NN, and thus we should get a more accurate result using KNN. However there is a trend that the performance of KNN decreases slightly as k increases, and thus we must select the value of k carefully.

APPENDIX

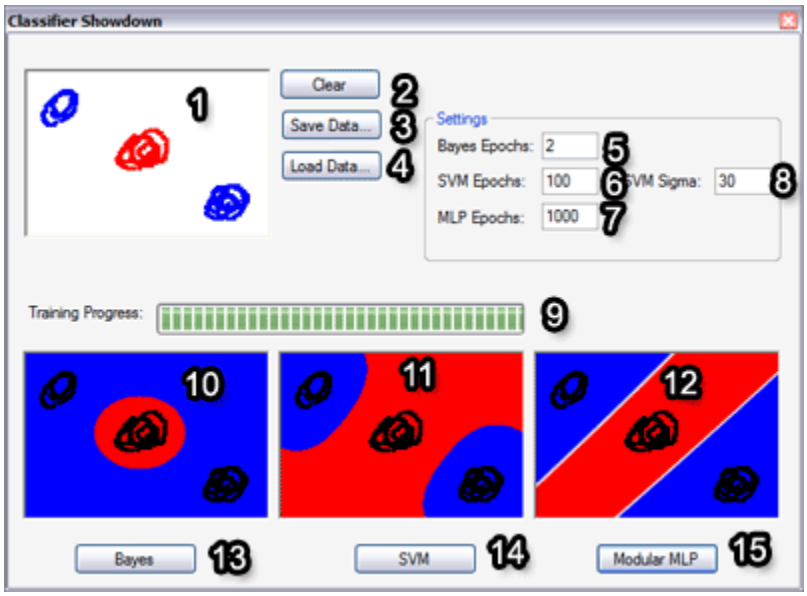
Problem 1 MATLAB Codes

```
% Created by
% This m-file tests the two methods of Fishers Linear Discriminant:
%   S_w or the Identity Matrix

clear all; close all; clc;

rand('state', 1000);
nbite = 1000;                                % # of experiments

for i = 1:nbite;

    x(:,1:10)  = randn(2,10);                % 10 two-dimensional dataset
    class(1,1:10)   = 1;                     % 1st class
    x(:,11:20) = randn(2,10) + repmat([2 2.5]',1,10);
    class(1,11:20)  = 2;                     % 2nd class

    [d n] = size(x);
    [w, f, f2] = LDAplane(x', class, x');
    Perf_real(i)   = sum(class == f')/n;  % correctness
    Perf_sub(i)    = sum(class == f2')/n;

end

% Performance Plots
figure();
n = 1:nbite;
plot(n,100-Perf_real*100);
grid on; hold on;
plot(n,100-Perf_sub*100,'r');
title('FLD Performance Plots: Using S_w vs. The Identity Matrix');
xlabel('Experiment #'); ylabel('Error Percentage (%)');

% % plot the data :
% plot(x(1,1:10),x(2,1:10),'r.');text(x(1,1:10),x(2,1:10),'1');
% hold on; grid on;
% plot(x(1,11:20),x(2,11:20),'b.');text(x(1,11:20),x(2,11:20),'2');
```

```
% Courtesy of Nuo Li
% http://www.mit.edu/~linuo/files/LDAplane.m
% Modified by
% This function now examines Fisher's Linear Discriminant with parameters
%   S_w and the identity matrix.

function [w, f, f2, b, fTrain, fTest] = LDAplane (data, label, test)
% Estimate the vector w normal to the linear discriminant hyperplane
% data is the matrix: each row is one data point, each column is one
% feature (dimension) of the data
%
% Note: currently, this program supports multi classification, label has
% to be adjcent integer numbers (e.g. [0 1 2 ...k] for k classes)

% sorting matrix
w=[];
f=[];
f2=[];
b=[];
fTrain=[];
fTest=[];

data=data';
featNum=size(data,1);
dataNum=size(data,2);

for Nclass=min(label):max(label)

    covB=[];
    covW=[];
    mui=[];
    classLabel=(label==Nclass);

    for k=0:1
        % Load digits
        x=data(:,find(classLabel==k));
        eval(['x',num2str(k),'=x;']);

        %mean
        mui(:,end+1)=mean(x,2);
    end

    mu=mean(mui,2);

    %covariance
    covW=0;
    covB=0;
    for k=0:1
        eval(['x=x',num2str(k),';']);
        for n=1:size(x,2)
            covW=covW+(x(:,n)-mui(:,k+1))*(x(:,n)-mui(:,k+1))';
        end
    end
    covW=covW/dataNum;

    %normal vector to Hyperplane
    classW=-inv(covW)*diff(mui,1,2);
    classW2 = diff(mui,1,2);        % when the numerator is maximized

    %offset
    prior=0;
    %prior=log(size(x0,2)/size(x1,2));
    classB=1/2*sum(mui,2)'*inv(covW)*diff(mui,1,2)+prior;
```

```
    %projecting and offseting (this computes vote)
    fTestClass   = -(test*classW+classB);
    fTrainClass  = -(data'*classW+classB);
    fTestClass2  = -(test*classW2+classB);
    fTrainClass2 = -(data'*classW2+classB);

    %offseting
    classF  = fTestClass;
    classF2 = fTestClass2;

    %Output into sorting matrix
    w(:,end+1)=classW;
    f(:,end+1)=classF;
    f2(:,end+1)=classF2;
    b(:,end+1)=classB;
    fTrain(:,end+1)=fTrainClass;
    fTest(:,end+1)=fTestClass;

end

% Classify by taking votes
[dummy f]=max(f,[],2);
f=f+min(label)-1;
[dummy f2]=max(f2,[],2);
f2=f2+min(label)-1;

return
```

Problem 2 Software Description



1. Draw Area. You input Blue class values by left clicking and Red class values by right clicking.
2. Clears the draw area.
3. Saves the pattern in the Draw Area and the settings to file.
4. Loads a pattern from file.
5. Sets the number of training epochs for the Bayes classifier. As it is instantaneously trained, there is no need to increase that value.
6. Number of SVM training epochs.
7. Number of training epochs for the MLP.
8. The sigma parameter for the kernel of the SVM. Small sigmas = better exact solution, Large sigmas = more general solutions
9. Bar that shows the training process.
10. The results of the Bayes classification.
11. The results of the SVM classification.
12. The results of the MLP classification.
13. Starts the Bayes training and evaluation.
14. Starts the SVM training and evaluation.
15. Starts the MLP training and evaluation.

Problem 3 MATLAB and C Codes

```
% Created by Sébastien PARIS
% Modified by
% Parzen Window and KNN Demos

clear all; close all; clc;

rand('twister',5489);

load wine

[d , N]    = size(X);           % dimension and # of vectors
n1         = round(0.7*N);      % 70% training, 30% test
n2         = N - n1;
nbite      = 1000;              % # of experiments
sigma      = 1;
Perf       = zeros(1 , nbite);  % reserve some space for Perf
k          = 3;

for i=1:nbite

    ind        = randperm(length(y));   % random permutation from 1 to n

    ind1       = ind(1:n1);
    ind2       = ind(n1+1:N);

    Xtrain     = X(: , ind1);           % take samples for Xtrain from X
    ytrain     = y(ind1);
    Xtest      = X(: , ind2);
    ytest      = y(ind2);

    % At this point, have Xtrain, ytrain, Xtest, ytest
    ytest_est_parzen  = parzen_classif(Xtrain , ytrain , Xtest , sigma );
    ytest_est_knn     = knn(Xtrain , ytrain , Xtest , k );

    Perf_parzen(i)    = sum(ytest == ytest_est_parzen)/n2;  % correctness
    Perf_knn(i)    = sum(ytest == ytest_est_knn)/n2;  % correctness

end

Xtrain0 = Xtrain(:,find(ytrain==0));
Xtrain1 = Xtrain(:,find(ytrain==1));
Xtrain2 = Xtrain(:,find(ytrain==2));

% plot training data :
% MODIFY THESE TWO LINES FOR DIFFERENT DATA SETS
dim_horz = 1;
dim_vert = 7;

% PARZEN WINDOW
figure();
plot(Xtrain1(dim_horz,:),Xtrain1(dim_vert,:),'r.');text(Xtrain1(dim_horz,:),Xtrain1(dim_v
ert,:),'1');
hold on;
plot(Xtrain2(dim_horz,:),Xtrain2(dim_vert,:),'b.');text(Xtrain2(dim_horz,:),Xtrain2(dim_v
ert,:),'2');
plot(Xtrain0(dim_horz,:),Xtrain0(dim_vert,:),'g.');text(Xtrain0(dim_horz,:),Xtrain0(dim_v
ert,:),'0');
grid on; title('Parzen Window Method');
xlabel(sprintf('dimension #%d', dim_horz)); ylabel(sprintf('dimension #%d', dim_vert));
% plot test data
xquery = Xtest;
```

```
plot(xquery(dim_horz,:),xquery(dim_vert,:),'k.')
nn_label = ytest_est_parzen;
nn_label1=find(nn_label==1);nn_label2=find(nn_label==2);nn_label3=find(nn_label==0);
text(xquery(dim_horz,nn_label1),xquery(dim_vert,nn_label1),'1');
text(xquery(dim_horz,nn_label2),xquery(dim_vert,nn_label2),'2');
text(xquery(dim_horz,nn_label3),xquery(dim_vert,nn_label3),'0'); hold off;
legend('training: class 1','training: class 2','training: class 0','test data');


% K-NEAREST NEIGHBORS
figure();
plot(Xtrain1(dim_horz,:),Xtrain1(dim_vert,:),'r.');text(Xtrain1(dim_horz,:),Xtrain1(dim_v
ert,:),'1');
hold on;
plot(Xtrain2(dim_horz,:),Xtrain2(dim_vert,:),'b.');text(Xtrain2(dim_horz,:),Xtrain2(dim_v
ert,:),'2');
plot(Xtrain0(dim_horz,:),Xtrain0(dim_vert,:),'g.');text(Xtrain0(dim_horz,:),Xtrain0(dim_v
ert,:),'0');
grid on; title('K-Nearest Neighbors Method');
xlabel(sprintf('dimension #%d', dim_horz)); ylabel(sprintf('dimension #%d', dim_vert));
% plot test data
xquery = Xtest;
plot(xquery(dim_horz,:),xquery(dim_vert,:),'k.')
nn_label = ytest_est_knn;
nn_label1=find(nn_label==1);nn_label2=find(nn_label==2);nn_label3=find(nn_label==0);
text(xquery(dim_horz,nn_label1),xquery(dim_vert,nn_label1),'1');
text(xquery(dim_horz,nn_label2),xquery(dim_vert,nn_label2),'2');
text(xquery(dim_horz,nn_label3),xquery(dim_vert,nn_label3),'0'); hold off;
legend('training: class 1','training: class 2','training: class 0','test data');

disp(sprintf('Parzen Performance = %4.2f%%' , 100*mean(Perf_parzen)))
disp(sprintf('KNN Performance = %4.2f%%' , 100*mean(Perf_knn)))

% Performance Plot
figure();
n = 1:nbite;
plot(n,100-Perf_parzen*100);
grid on; title('Parzen Performance Plot');
xlabel('Experiment #'); ylabel('Error Percentage (%)');

figure();
n = 1:nbite;
plot(n,100-Perf_knn*100,'r');
grid on; title('KNN Performance Plot');
xlabel('Experiment #'); ylabel('Error Percentage (%)');
```

```
/*
 parzen_classif : Returns the estimated Labels ytest [1 x Ntest] given Xtrain data [d x
Ntrain] and train label ytrain [1 x Ntrain]

 Usage
 -------
 [ytest , densite]    = parzen_classif(Xtrain , ytrain , Xtest , [sigma] );

 Inputs
 -------
 Xtrain        Train data   (d x Ntrain)
 ytrain        Train labels (1 x Ntrain)
 Xtest         Test  data   (d x Ntest)
 sigma         Noise Standard deviation of the rbf (default sigma = 1.0)

 Outputs
 -------
 ytest         Estimated labels (1 x Ntest)
 densite       Estimated density (m x Ntest) where m denotes the number of class

 To compile
 ----------
 mex -output parzen_classif.dll parzen_classif.c
 mex  -f mexopts_intelamd.bat -output parzen_classif.dll parzen_classif.c

Author : Sébastien PARIS : sebastien.paris@lsis.org
-------

Reference  : Vincent, P. and Bengio, Y. (2003). Manifold parzen windows. In Becker, S.,
Thrun, S., and Obermayer, K., editors,
-----------  Advances in Neural Information Processing Systems 15, Cambridge, MA. MIT
Press.
*/

#include <math.h>
#include "mex.h"

void qs( double * , int , int  );
void parzen_classif(double * , double * , double * , double  , double * , double *, int  ,
int , int  , int , double * , double* );

void mexFunction( int nlhs, mxArray *plhs[] , int nrhs, const mxArray *prhs[] )
{
      double *Xtrain , *ytrain , *Xtest;
      double *ytest, *densite;
      int  d , Ntrain , Ntest , m=0 , i , currentlabel;
      double sigma = 1.0;
      double *ytrainsorted , *labels;

      /*-------------------------------------------------------------------------
*/
      /*-------------------------------------------------------------------------
*/
      /* ------------------------- Parse INPUT  ----------------------------------
*/
      /*-------------------------------------------------------------------------
*/
      /*-------------------------------------------------------------------------
*/


      /* ----- Input 1 ----- */
      Xtrain           = mxGetPr(prhs[0]);
```

```
	d               = mxGetM(prhs[0]);
	Ntrain          = mxGetN(prhs[0]);

	/* ----- Input 2 ----- */
	ytrain          = mxGetPr(prhs[1]);

	if(mxGetN(prhs[1]) != Ntrain)
	{
		mexErrMsgTxt("ytrain must be (1 x Ntrain)");
	}

	/* ----- Input 3 ----- */
	Xtest           = mxGetPr(prhs[2]);

	if(mxGetM(prhs[2]) != d)
	{
		mexErrMsgTxt("Xtest must be (d x Ntest)");
	}

	Ntest           = mxGetN(prhs[2]);

	/* ----- Input 4 ----- */
	if (nrhs > 3)
	{
		sigma       = (double)mxGetScalar(prhs[3]);
	}

	/* Determine unique Labels */
	ytrainsorted  = mxMalloc(Ntrain*sizeof(double));
	for ( i = 0 ; i < Ntrain; i++ )
	{
		ytrainsorted[i] = ytrain[i];
	}
	qs( ytrainsorted , 0 , Ntrain - 1 );

	labels       = mxMalloc(sizeof(double));
	labels[m]    = ytrainsorted[0];
	currentlabel = labels[0];

	for (i = 0 ; i < Ntrain ; i++)
	{
		if (currentlabel != ytrainsorted[i])
		{
			labels       = (double *)mxRealloc(labels , (m+2)*sizeof(double));
			labels[++m]  = ytrainsorted[i];
			currentlabel = ytrainsorted[i];
		}
	}

	m++;

	/*-------------------------------------------------------------------------
*/
	/*-----------------------------------------,-------------------------------
*/
	/* -------------------------- Parse OUTPUT  ----------------------------------
*/
	/*-------------------------------------------------------------------------
*/
	/*-------------------------------------------------------------------------
*/

	/* ----- output 1 ----- */
```

```
      plhs[0]        = mxCreateDoubleMatrix(1 , Ntest , mxREAL);
      ytest          = mxGetPr(plhs[0]);
      plhs[1]        = mxCreateDoubleMatrix(m , Ntest , mxREAL);
      densite        = mxGetPr(plhs[1]);

      /*-------------------------------------------------------------------------------
*/
      /*-------------------------------------------------------------------------------
*/
      /* ---------------------- MAIN CALL  ---------------------------------------
*/
      /*-------------------------------------------------------------------------------
*/
      /*-------------------------------------------------------------------------------
*/
      /*-------------------------------------------------------------------------------
*/

      parzen_classif(Xtrain , ytrain , Xtest , sigma ,
                       ytest , densite ,
                           d , m , Ntrain , Ntest , labels , ytrainsorted);
      mxFree(labels);
      mxFree(ytrainsorted);

      /*-----------------------------------------------*/
      /*-----------------------------------------------*/
      /* ------------ END of Mex File ---------------- */
      /*-----------------------------------------------*/
      /*-----------------------------------------------*/
}

void parzen_classif(double *Xtrain , double *ytrain , double *Xtest , double sigma ,
                            double *ytest , double *densite,
                            int d , int m, int Ntrain , int Ntest , double *labels ,
double *ytrainsorted)
{
      int i , j , l , t , id , jd , im , ind;
      double temp , res  , maxi;
      double cte = -0.5/(sigma*sigma), cteprior , epsilon = 10e-50 ;
      double  *prior;

      /*             Prior            */
      prior    =  mxMalloc(m*sizeof(double));
      for (t = 0 ; t < m ; t++)
      {
            prior[t]    = 0.0;
      }

      for (i = 0 ; i < Ntrain ; i++)
      {
            for (t = 0 ; t < m ; t++)
            {
                  if (labels[t] == ytrainsorted[i])
                  {
                        prior[t]++;
                  }
            }
      }

      for (t = 0 ; t < m ; t++)
      {
            prior[t]    /= Ntrain;
```

```
	}

	/* Classify */
	for (i = 0; i < Ntest ; i++)
	{
		id    = i*d;
		im    = i*m;

		for (j = 0 ; j < Ntrain ; j++)
		{
			jd  = j*d;
		for(t = 0 ; t < m ; t++)
			{
				if(ytrain[j] == labels[t])
				{
					ind = t;
				}
			}

			res = 0.0;
			for (l = 0 ; l < d ; l++)
			{
				temp  = (Xtest[l + id] - Xtrain[l + jd]);
				res  += (temp*temp);
			}
			densite[ind + im] += exp(cte*res) + epsilon;
		}

		cteprior = 0.0;
		for(t = 0 ; t < m ; t++)
		{
			densite[t + im] *=prior[t];
			cteprior        += densite[t + im];
		}

		cteprior = 1.0/cteprior;
		ind      = 0;
		maxi     = 0.0;

		for(t = 0 ; t < m ; t++)
		{
			temp            = densite[t + im]*cteprior;
			densite[t + im] = temp;
			if(temp > maxi)
			{
				maxi = temp;
				ind  = t;
			}
		}
		ytest[i] = labels[ind];
	}
	mxFree(prior);
}

/*-----------------------------------------------------------------------------------
----------*/

void qs( double *array, int left, int right )
{
	double pivot;     // pivot element.
	int holex;  // hole index.
	int i;
	holex          = left + ( right - left )/2;
```

```
        pivot           = array[ holex ];               // get pivot from middle of array.
        array[holex]    = array[ left ];              // move "hole" to beginning of
        holex           = left;                           // range we are sorting.

        for ( i = left + 1 ; i <= right ; i++ )
        {
                if ( array[ i ] <= pivot )
                {
                        array[ holex ] = array[ i ];
                        array[ i ]     = array[ ++holex ];
                }
        }

        if ( holex - left > 1 )
        {
                qs( array, left, holex - 1 );
        }
        if ( right - holex > 1 )
        {
                qs( array, holex + 1, right );
        }
        array[ holex ] = pivot;
}
```

```
/*-------------------------------------------------------------------------

 K-NN classifier.

Synopsis:
[ytest , proba] = knn(Xtrain , ytrain , Xtest , [k]);

Inputs:
------
Xtrain           Training data [d x Ntrain]
ytrain           Labels of training data [1 x Ntrain]
Xtest            Data to be classified [d x Ntest]
k                Number of neighbours (default k = 3)

Output:
-------
ytest            Estimated labels of testing data [1 x Ntest]
proba            Estimated Pr(w=j|x), j=1,...,m where m is the number of classes. (m x
Ntest)

 Author : Sébastien PARIS : sebastien.paris@lsis.org
 -------  Date : 15/03/2007

mex -f mexopts_intelamd.bat -output knn.dll knn.c
-------------------------------------------------------------------

*/
#include <math.h>
#include <limits.h>
#include "mex.h"

#define MAX_INF INT_MAX
#define MAX(A,B)   (((A) > (B)) ? (A) : (B) )
#define MIN(A,B)   (((A) < (B)) ? (A) : (B) )

void qs( double * , int , int  );
void mexFunction( int nlhs, mxArray *plhs[] , int nrhs, const mxArray*prhs[] )
{
      double *Xtest, *Xtrain , *ytrain;
      double *ytest , *proba ;
      double *dist;
      int    *max_labels;
      int  Ntest, Ntrain, k = 3 , i, j, l, d , id , jd , im;
      double temp , adist, max_dist;
      int  max_inx, best_count, best_label, count , ind , m=0;
      double  currentlabel , sumdistmini;
      double *ytrainsorted , *labels , *distmini;

      if( nrhs < 3)
      {
            mexErrMsgTxt("Incorrect number of input arguments.");
      }

      /* -- gets input arguments -------------------------- */

      Xtrain   = mxGetPr(prhs[0]);
      d        = mxGetM(prhs[0]);              /* data dimension */
      Ntrain   = mxGetN(prhs[0]);              /* number of data */
      ytrain   = mxGetPr(prhs[1]);
      Xtest    = mxGetPr(prhs[2]);
      Ntest    = mxGetN(prhs[2]);              /* number of data */

      if( d != mxGetM( prhs[2] ))
```

```
{
      mexErrMsgTxt("Dimension of training and testing data differs.");
}

if(nrhs>3)
{
      k          = (int) mxGetScalar(prhs[3]);
}
/* Determine unique Labels */

ytrainsorted  = mxMalloc(Ntrain*sizeof(double));

for ( i = 0 ; i < Ntrain; i++ )
{
      ytrainsorted[i] = ytrain[i];
}

qs( ytrainsorted , 0 , Ntrain - 1 );
labels       = mxMalloc(sizeof(double));
labels[m]    = ytrainsorted[0];
currentlabel = labels[0];

for (i = 0 ; i < Ntrain ; i++)
{
      if (currentlabel != ytrainsorted[i])
      {
            labels       = (double *)mxRealloc(labels , (m+2)*sizeof(double));
            labels[++m]  = ytrainsorted[i];
            currentlabel = ytrainsorted[i];
      }
}

m++;

/*  output labels*/
plhs[0]    = mxCreateDoubleMatrix(1 , Ntest , mxREAL);
ytest      = mxGetPr(plhs[0] );
plhs[1]    = mxCreateDoubleMatrix(m , Ntest , mxREAL);
proba      = mxGetPr(plhs[1] );

/*--------------------------*/
dist       = mxMalloc(Ntrain*sizeof(double));
max_labels = mxMalloc(k*sizeof(int));
distmini   = mxMalloc(m*sizeof(double));

for( i = 0 ; i < Ntest; i++ )
{
      id      = i*d;
      im      = i*m;
      for (j = 0 ; j < m ; j++)
      {
            distmini[j] = MAX_INF;
      }

      for( j = 0 ; j < Ntrain ; j++ )
      {
            jd    = j*d;
            adist = 0.0;
            for( l = 0 ; l < d ; l++ )
            {
                  temp   = (Xtest[l + id] - Xtrain[l + jd]);
                  adist += temp*temp;
            }
```

```
      dist[j]  = sqrt(adist);

      for (l = 0 ; l < m ; l++)
      {
            if (ytrain[j] == labels[l])
            {
                  ind = l;
            }
      }

      if (dist[j] < distmini[ind])
      {
            distmini[ind] = dist[j];
      }
}

sumdistmini  = 0.0;
for (j = 0 ; j < m ; j++)
{
      sumdistmini += distmini[j];
}

sumdistmini  = 1.0/sumdistmini;
for( l = 0 ; l < k ; l++)
{
      max_dist = MAX_INF;
      for( j = 0 ; j < Ntrain ; j++ )
      {
            if( max_dist > dist[j] )
            {
                  max_inx  = j;
                  max_dist = dist[j];
            }
      }

      dist[ max_inx ] = MAX_INF;
      max_labels[l]   = (int) ytrain[max_inx];
}

best_count = 0;
for( l = 0 ; l < k; l++)
{
      count = 0;
      ind   = max_labels[l];
      for( j = 0 ; j < k; j++)
      {
            if( ind == max_labels[j] )
            {
                  count++;
            }
      }
      if( count > best_count )
      {
            best_count = count;
            best_label = max_labels[l];
      }
}

ytest[i] = best_label;
for (l = 0 ; l < m ; l++)
{
      proba[l + im] = 1.0 - (m - 1.0)*distmini[l]*sumdistmini;
```

```
            }
      }

  mxFree(dist);
  mxFree(max_labels);
  mxFree(labels);
  mxFree(ytrainsorted);
  mxFree(distmini);

}

/*-------------------------------------------------------------------------------------
----------------------- */
void qs( double *array , int left , int right )
{
      double pivot;     // pivot element.
      int holex;  // hole index.
      int i;
      holex          = left + ( right - left )/2;
      pivot          = array[ holex ];              // get pivot from middle of array.
      array[holex]   = array[ left ];             // move "hole" to beginning of
      holex          = left;                          // range we are sorting.

      for ( i = left + 1 ; i <= right ; i++ )
      {
            if ( array[ i ] <= pivot )
            {
                  array[ holex ] = array[ i ];
                  array[ i ]     = array[ ++holex ];
            }
      }

      if ( holex - left > 1 )
      {
            qs( array, left, holex - 1 );
      }
      if ( right - holex > 1 )
      {
            qs( array, holex + 1, right );
      }
      array[ holex ] = pivot;
}

/* ------------------------------------------------------------------------- */
```

```
% Created by David Barber
% Modified by
% Nearest Neighbors Demo : 3 classes

clear all; close all; clc;

rand('twister',5489);

load wine

[d , N]    = size(X);           % dimension and # of vectors
n1         = round(0.7*N);      % 70% training, 30% test
n2         = N - n1;
nbite      = 1000;              % # of experiments
sigma      = 1;
Perf       = zeros(1 , nbite);  % reserve some space for Perf
k          = 3;

for i=1:nbite

    ind        = randperm(length(y));   % random permutation from 1 to n

    ind1       = ind(1:n1);             % 1st half of ind
    ind2       = ind(n1+1:N);           % 2nd half of ind

    Xtrain     = X(: , ind1);           % take samples for Xtrain from X
    ytrain     = y(ind1);
    Xtest      = X(: , ind2);
    ytest      = y(ind2);

xquery = Xtest;
xtrain = Xtrain;
label = ytrain;

nn_label = nearest_neighbour(xtrain, xquery, label);
ytest_est_nn = nn_label;
Perf_nn(i)    = sum(ytest == ytest_est_nn)/n2;  % correctness

end

Xtrain0 = Xtrain(:,find(ytrain==0));
Xtrain1 = Xtrain(:,find(ytrain==1));
Xtrain2 = Xtrain(:,find(ytrain==2));

% plot training data :
% MODIFY THESE TWO LINES FOR DIFFERENT DATA SETS
dim_horz = 1;
dim_vert = 7;

figure();
plot(Xtrain1(dim_horz,:),Xtrain1(dim_vert,:),'r.');text(Xtrain1(dim_horz,:),Xtrain1(dim_v
ert,:),'1');
hold on;
plot(Xtrain2(dim_horz,:),Xtrain2(dim_vert,:),'b.');text(Xtrain2(dim_horz,:),Xtrain2(dim_v
ert,:),'2');
plot(Xtrain0(dim_horz,:),Xtrain0(dim_vert,:),'g.');text(Xtrain0(dim_horz,:),Xtrain0(dim_v
ert,:),'0');
grid on; title('Nearest Neighbors Method');
xlabel(sprintf('dimension #%d', dim_horz)); ylabel(sprintf('dimension #%d', dim_vert));
% plot test data
plot(xquery(dim_horz,:),xquery(dim_vert,:),'k.')
nn_label1=find(nn_label==1);nn_label2=find(nn_label==2);nn_label3=find(nn_label==0);
text(xquery(dim_horz,nn_label1),xquery(dim_vert,nn_label1),'1');
```

```
text(xquery(dim_horz,nn_label2),xquery(dim_vert,nn_label2),'2');
text(xquery(dim_horz,nn_label3),xquery(dim_vert,nn_label3),'0'); hold off;
legend('training: class 1','training: class 2','training: class 0','test data');


% Performance Plot
figure();
n = 1:nbite;
plot(n,100-Perf_nn*100,'g');
grid on; title('Nearest Neighbors Performance Plot');
xlabel('Experiment #'); ylabel('Error Percentage (%)');
```

```
% Created by David Barber
% http://www.anc.ed.ac.uk/~amos/lfd/

function y = nearest_neighbour(xtrain, xtest, t)
% calculate the nearest (single) neighbour classification
% (uses the squared distance to measure dissimilarity)

ntrain = size(xtrain,2); % number of training points
ntest = size(xtest,2); % number of test points

% Compute squared distances between vectors from the training and test sets
% This is the obvious (but very slow way) to calculate distances :
% for i = 1:ntrain
%     for j = 1:ntest
%         sqdist(i,j) = sum((xtrain(:,i)-xtest(:,j)).^2);
%     end
% end

% This is the super fast way (in MATLAB) to do this :
sqdist = repmat(sum(xtrain'.^2,2),1,ntest)+ ...
repmat(sum(xtest'.^2,2)',ntrain,1)-2*xtrain'*xtest;
[vals, kindex] = min(sqdist); y = t(kindex);
```