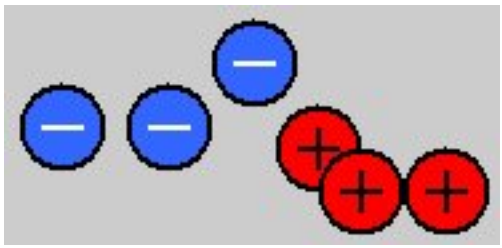
CLS Version 1.0



A Work Bench for Classification Algorithms

Written in Matlab

11th of December 1999

Thomas Hellström
Department of Computing Science

Umeå University Sweden.

email: thomash@cs.umu.se

Table of Contents

Introduction

This document describes the CLS system which is a Matlab based research tool for classification algorithms. It can be used for bench marking, testing, and developing.

The information is mainly aimed at potential users but also for those who want to get an overview of how a work bench for algorithm development can be designed and implemented in the Matlab environment.

Classification

A classifier is a decision rule or an algorithm that assigns a class label C to an object with features $X=\{x_1,..x_m\}$. The most common methods use training data (X_i,C_i), $i=1,...,n$ to produce a mapping $d(X)\rightarrow C$. The aim is to produce a rule that works well on previously unseen data. I.e. the decision rule should "generalize" well. Examples of classification problems can be found in all sort of branches:

- Medical diagnosis
  Data from a medical examination (blood pressure, age, pain indications, etc.) is used to identify patients likely to have a certain illness.
- Bankruptcy prediction
  Given a set of variables (turnover, sales prognoses, annual profit, growth, etc.) those companies likely to bankrupt within a year are identified. This information can be used by a bank when a company applies for more credit.
- Loan approval
  A loan approval process involves filling out forms, which are then used by a loan officer to take a decision. Typical variables of interest are age, occupation, salary, car, family status, etc.
- Potential customer analysis for the creation of mailing lists
  Given a set of variables (age, sex, hobbies, occupation) those who are likely to respond to a planned direct sales campaign are identified.
- Quality control in manufacturing
  Aims at finding the one defective part in a hundred or a thousand. Inputs may be: solder joints, welds, cuttings, etc.

Common techniques for classification tasks are:

- Decision trees
- Neural Networks
- Fuzzy-rule bases
- Discriminant Analysis
- K-nearest-neighbor techniques

The evaluation of performance is critical when developing classification algorithms, especially those based on techniques with weak modeling that makes few assumptions about the underlying classification process. Some applications, prediction of stocks to name one, present a delicate evaluation situation since expecting a certain performance for a working algorithm is still hard to distinguish from doing so by tossing a coin. Other needs for proper evaluation of performance arise when new classification algorithms are being developed and should be tested against other methods on a large number of benchmark problems. This is of course a time consuming process, often neglected in both research and real applications.

The CLS System

The CLS system is a Matlab-based environment for bench marking, testing, and developing classification algorithms.

The development of CLS was instigated by the following observations:

- The need for a uniform bench mark for classification algorithms
  It is a well-known fact that no universal classifier exists, which has a superior performance on all sort of applications. When developing algorithms, it is therefore important to compute the performance for the algorithms in a uniform way.
- The developer should not need to spend time with the relatively trivial tasks of data handling and performance computation.

CLS addresses these issues by providing an environment where data sets and classification algorithms can be tested, tuned, and evaluated in an efficient way.

The development of CLS has been guided by the following design criteria:

- Interactivity
  A Windows version is preferred when performing interactive testing and development of algorithms. It is also useful for educational purposes.
- Automation
  Experience shows that data analysis is an iterative process where the analyses have to be repeated many times when writing scientific reports, and in general when conducting research. A command-line version of CLS is therefore essential and can be used to create batch files that automatically apply a specific classifier to a large number of data sets. Likewise, a range of classifiers can be applied to one specific data set. The results can be easily used for presentation material or report generation.
- Easy expansion and adaptation
  New data and classification algorithms should be **easy** to interface to the system. Data can be read from ASCII files with a simple and clearly defined format. All classification algorithms are interfaced to the CLS system through a likewise simple and clearly specified interface. New classifiers can be added by any user and become totally integrated without any changes to the CLS system. More information can be found in Appendices 1 and 2.

Installation

The CLS system should be installed to a new directory on the Matlab search path. A suggested directory name is cls. The zipped installation file creates the subdirectories data and algorithms when unzipped. All three subdirectories should be added to the Matlab search path.

Quick Start of the Windows Version WCLS

1. To start the system type WCLS in the Matlab command prompt.
The dialog box as shown in Figure 1 is displayed.

2. Click Select Data and select from .mat or ASCII files

3. Click Select Algorithm and select from the list. The function call can be edited and the parameters can be changed.

4. Click Train&Test. The model is trained on the training data and tested on the test data

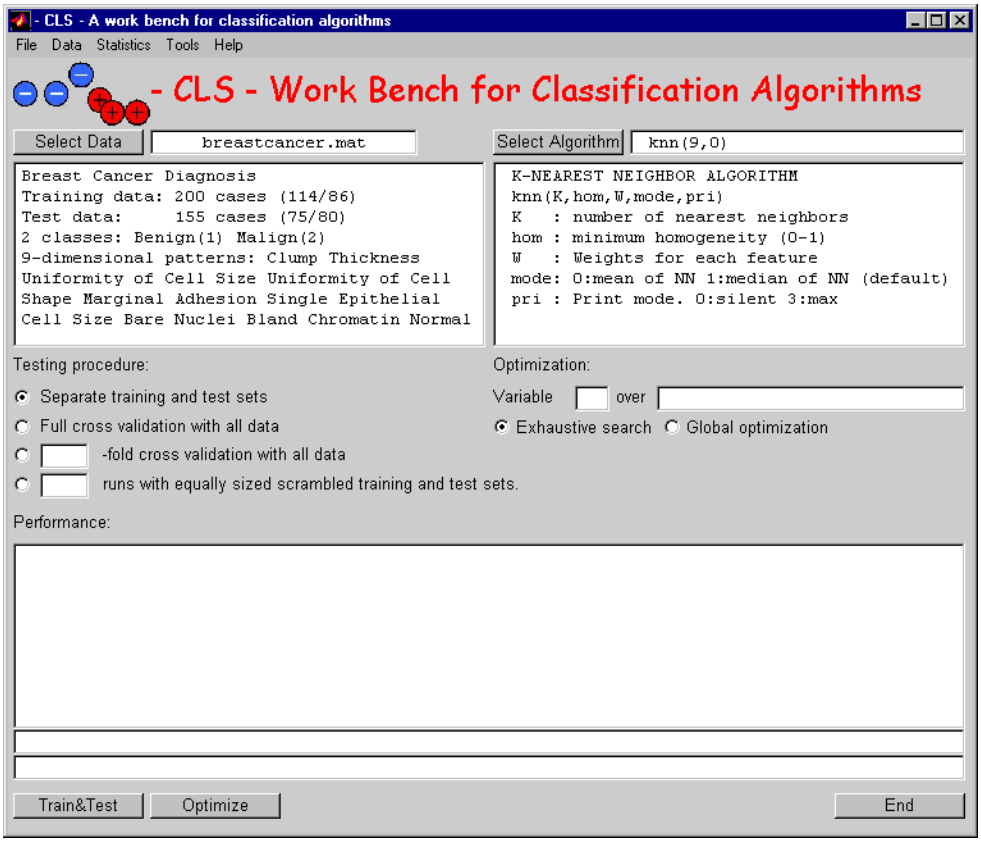


Figure 1

The Data

In the example shown in Figure 1. the .mat file “breastcancer” was loaded.

The Classification Algorithm

The selected classification algorithm was “K-NEAREST NEIGHBOR”. The text field of the button Select Algorithm shows the default function call for the selected algorithm. This field can be edited as required. The explanatory text in the larger frame below the field contains documentation for the selected algorithm. This frame can be scrolled up and down.

Supplied Data Sets

The following data sets are available for tests and comparison (they are stored on the \data directory):

- breastcancer.mat
- down.m
- genericascii.m
- diabetes.m
- letter.mat
- sonar.m

Click Select Data to display the documentation about the data sets.

New data can be easily added to the data base. Plain text files can be read by the *genericascii* interface. For further information about data formats refer to Appendix 1: Supplied Classification Algorithms

Supplied Classification Algorithms

The following classification algorithms are implemented in CLS (these algorithms are stored on the \algorithms directory):

- KNN
  K-nearest neighbor algorithm.
- ANN
  Artificial Neural Network. Requires Matlab Neural Network Toolbox 3.0.1
- LINDISC
  Linear Discriminant Analysis.
- LINREG
  Linear regression. Not quite a classifier but supplied to serve as a template for new classifiers.

Click Select Algorithm to display the documentation about the implemented classifiers. New algorithms are fairly easy to interface and require no changes in the CLS program files. Even the line documentation (shown when the algorithm is loaded) dispalys part of the algorithm and not of the CLS system. For further details and specifications refer to Appendix 2.

Performance Results

Click Train&Test to start the training phase for the selected classification algorithm using the training data. When the training is completed, the data in the test set is classified and the performance for the classifications is computed.

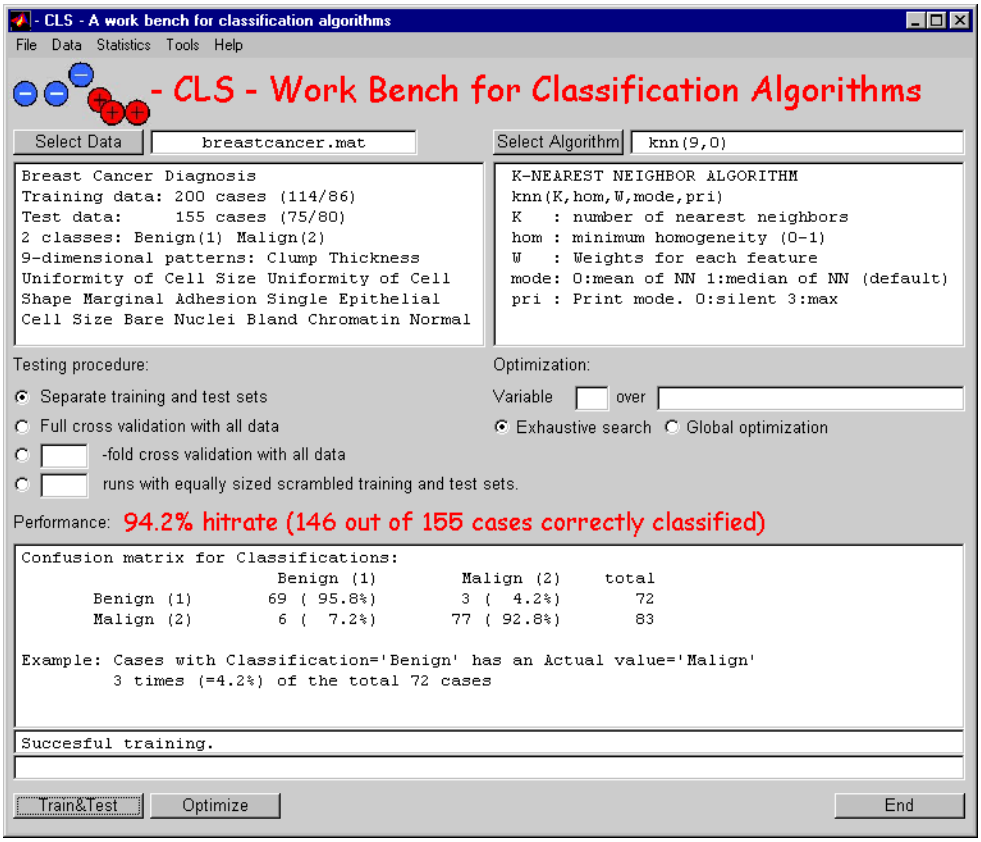


Figure 2

The results of the classification on the test data are presented in the so-called confusion matrix

```
67.2% hit rate (129 out of 192 cases correctly
classified)
Confusion matrix for Classifications:

                No (1)          Yes (2)         Total
No  (1)         72 ( 83.7%)     14 ( 16.3%)      86
Yes (2)         49 ( 46.2%)     57 ( 53.8%)     106
```

This matrix gives the classification results for all combinations of classification/real data in the examined test data set. The most interesting entity varies with the application. Sometimes it is extremely important to minimize the “false negative” classifications while at others the “false positive” ones should be minimized. Often, the total hit rate is of interest. This figure is presented in the header line, like the 67.2% in the example above.

Statistics

The menu item Statistics contains functions for graphical and numerical display of raw data and also of the computed classifications.

The following functions are available:

- 2D plot

  The class labels are plotted versus two of the input variables (features). In case there are more than 2 input variables in the loaded data sets, a popup window enables you to select 2 of them for plotting. Two graphs are generated; one for the Training data set and one for the Test data set. In addition to these graphs a third graph shows the decision boundary for the computed classifier. An example is shown in Figure 3 below. The circles denote class 1 and the dots denote class 2.

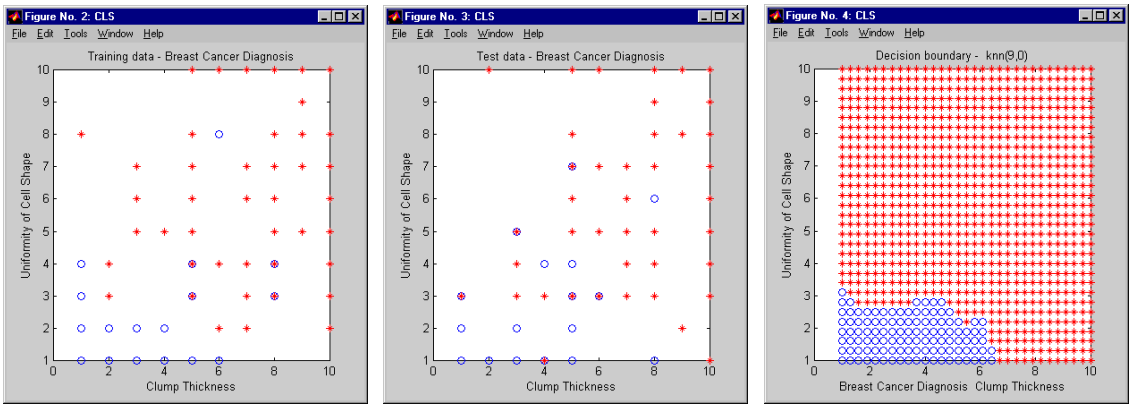


Figure 3

- 3D plot

The class labels are plotted versus two of the input variables (features). In case there are more than 3 input variables in the loaded data sets, a popup window enables you to select 3 of them for plotting. Two graphs are generated; one for the Training data set and one for the Test data set. In addition to these graphs a third graph shows the decision boundary for the computed classifier. An example is shown in Figure 4 below. The graphs may be rotated by dragging them with the mouse. The two classes are indicated by different colored dots in the graphs. A black and white printout might therefore look less informative than it does in reality…

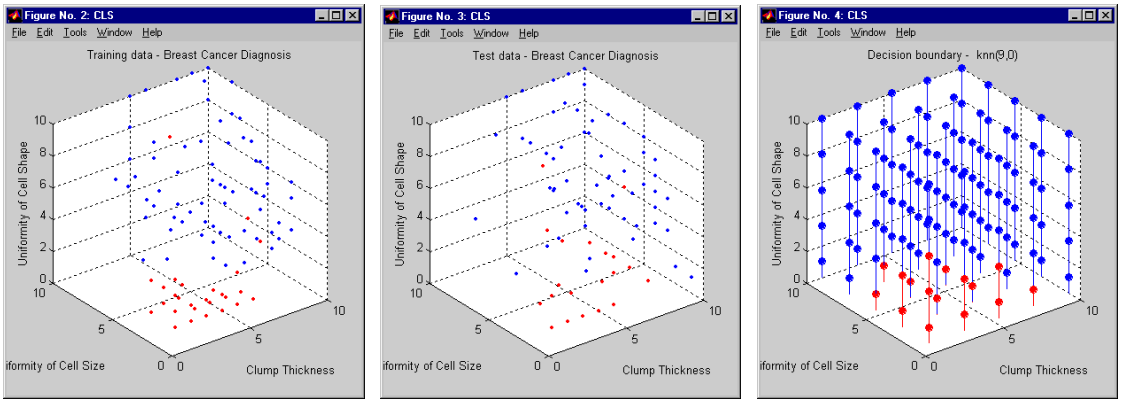


Figure 4

- Class Statistics

The distribution for all input variables is presented in histogram form with separate bars for each class, as shown in Figure 5. The mean values for each class/input variable are shown in the header of each sub-graph. Mean values and standard deviations are presented in tabular form in the CLS window. An example is shown in Figure 6 (the CLS window can be scrolled if not all the data fits.)

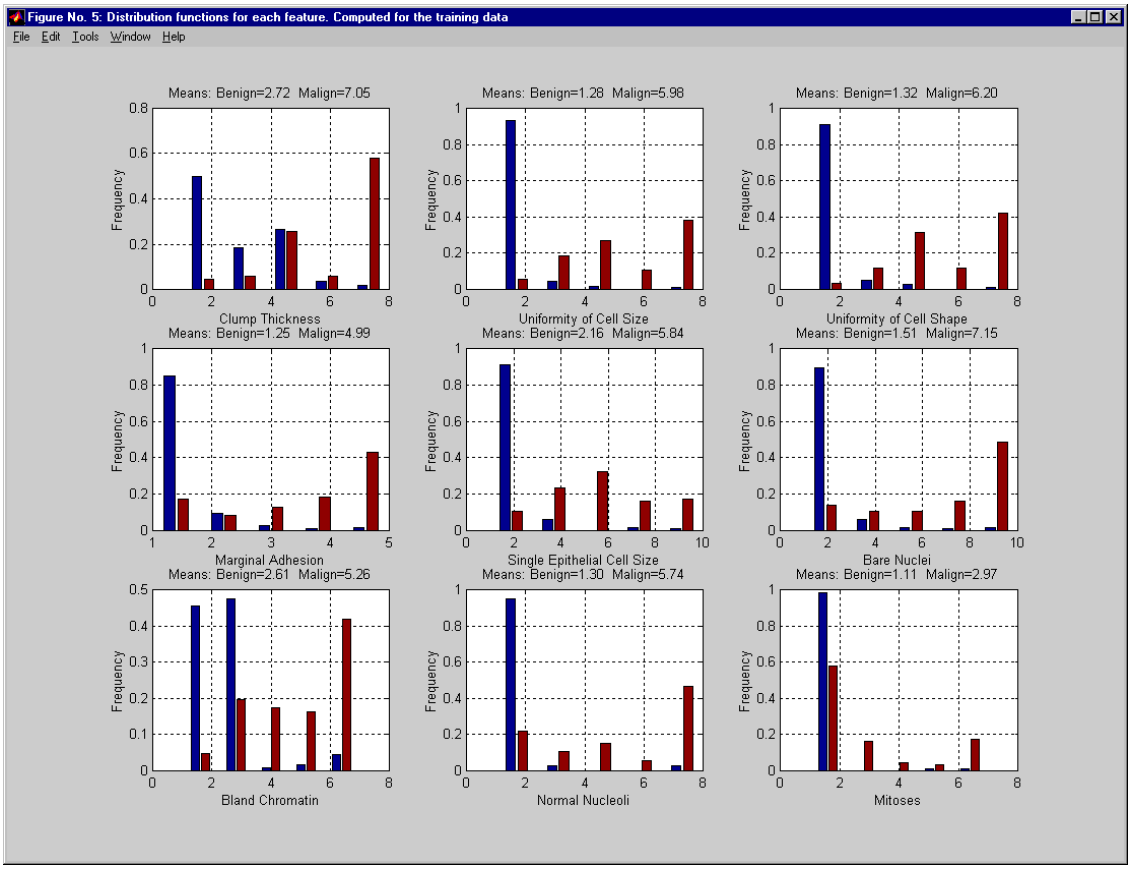


Figure 5

Analyzing the training data

Class:	Benign		Malign	
	Mean	**Std**	**Mean**	**Std**
Clump Thickness:	2.72	1.67	7.05	2.54
Uniformity of Cell Size:	1.28	0.88	5.98	2.66
Uniformity of Cell Shape:	1.32	0.93	6.20	2.47
Marginal Adhesion:	1.25	0.71	4.99	3.15
Single Epithelial Cell Size:	2.16	1.13	5.84	2.58
Bare Nuclei:	1.51	1.47	7.15	3.25
Bland Chromatin:	2.61	1.25	5.26	2.00
Normal Nucleoli:	1.30	1.13	5.74	3.36
Mitoses:	1.11	0.69	2.97	2.77

Figure 6

Testing Procedure

The computation of performance can be done in a number of ways. The following options are implemented in the CLS system:

- Separate testing and data sets

  The division of data in the input data files is used in this mode. The classifier algorithm is trained with the testing data and the performance is computed for the test data.

- Full cross validation

  The test and training sets are merged into one data set with N points. N classifiers are constructed and are iteratively trained on N-1 points, where one point is removed from the data set in each iteration. The performance for the constructed (trained) classifier is then computed using this single removed point.

- N-fold cross validation

  The test and training sets are merged into one data set with M points. This set is further divided into N smaller partitions, each with (approximately) M/N points. N classifiers are constructed, each using N-1 partitions with one partition removed. This removed partition is used for the performance computation. The total presented performance for the classifier is the mean value for the N runs.

- N runs with equally sized scrambled training and test sets

  The test and training sets are merged into one data set with M points which are randomly scrambled to avoid spurious behavior of the classifiers if the original data sets are sorted by the classification of the points. This set is split in half, thus producing a training and a test set, each with M/2 points. N classifiers are constructed, each with randomly selected data sets. The total presented performance for the classifier is the mean value for the N runs. The example in Figure 7 shows a 10-fold cross validation of a classification task for Downs syndrome. The K-nearest neighbor with K=25 is used as classifier.

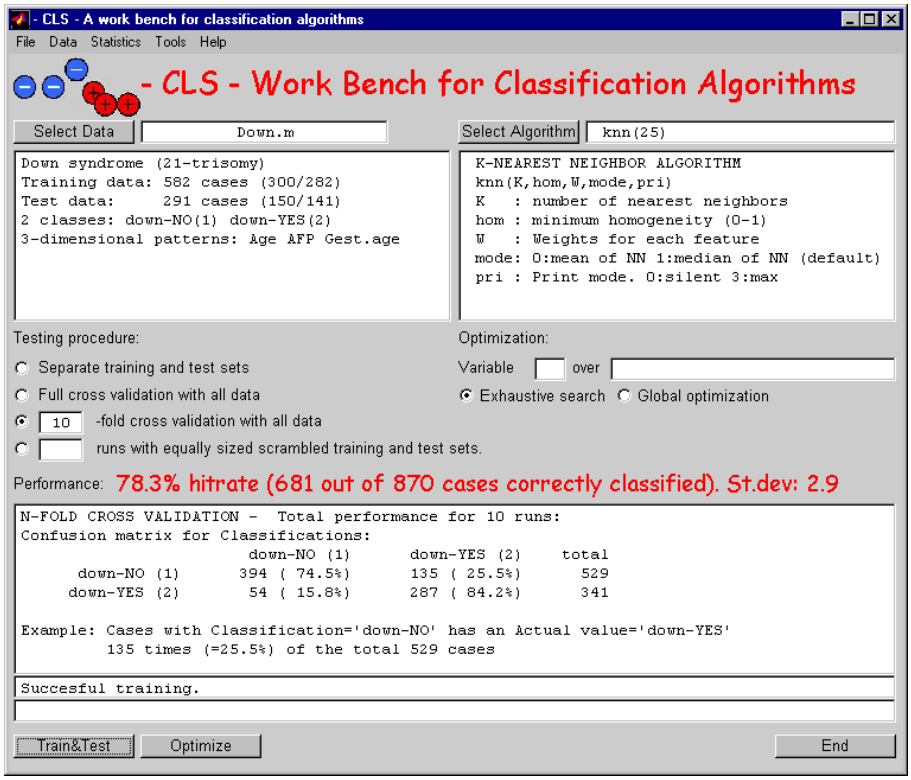

- CLS - A work bench for classification algorithms
File Data Statistics Tools Help
- CLS - Work Bench for Classification Algorithms
Select Data
Down.m
Select Algorithm
knn(25)
Down syndrome (21-trisomy)
Training data: 582 cases (300/282)
Test data: 291 cases (150/141)
2 classes: down-NO(1) down-YES(2)
3-dimensional patterns: Age AFP Gest.age
K-NEAREST NEIGHBOR ALGORITHM
knn(K,hom,W,mode,pri)
K : number of nearest neighbors
hom : minimum homogeneity (0-1)
W : Weights for each feature
mode: 0:mean of NN 1:median of NN (default)
pri : Print mode. 0:silent 3:max
Testing procedure:
Separate training and test sets
Full cross validation with all data
10 -fold cross validation with all data
runs with equally sized scrambled training and test sets.
Optimization:
Variable over
Exhaustive search Global optimization
Performance: 78.3% hitrate (681 out of 870 cases correctly classified). St.dev: 2.9
N-FOLD CROSS VALIDATION - Total performance for 10 runs:
Confusion matrix for Classifications:
down-NO (1) down-YES (2) total
down-NO (1) 394 (74.5%) 135 (25.5%) 529
down-YES (2) 54 (15.8%) 287 (84.2%) 341
Example: Cases with Classification='down-NO' has an Actual value='down-YES'
135 times (=25.5%) of the total 529 cases
Succesful training.
Train&Test
Optimize
End

Figure 7

Optimization

Many classifiers contain meta parameters that affect the performance in general or just a specific application. Examples are the number of hidden nodes in an artificial neural network and the value of K in the K-nearest-neighbor algorithm. The values most suited for a given data set might vary and should therefore be investigated in a thorough analysis. CLS supports this work by allowing the call to the classifier to contain a variable which is automatically set to different values as you may request. The name of the variable should be input in the Variable field, and a range such as 1:2:30 in the Over field. Now click Optimize. The variable x is assigned values from the Over range and a training and test is performed for each value of x. The results are presented as a graph showing the hitrate as function of the value of x. Figure 8 shows an analysis of the value of the parameter *k* in the *knn* classifier. The results are shown in the right graph in Figure 9. From this one can deduce that the optimal value for *k* is around 25.

The optimization may be combined with the variants of Testing Procedure described above. The left graph in Figure 9 shows the same analysis as before but with t different Testing procedure. The plotted hitrate for each value of k is computed as the mean of 10 runs with equally sized scrambled training and test sets. This gives statistically more reliable results. We also observe that the curve in this graph is less jagged than the one to the right.

Exhaustive search and Global optimization allow multivariable optimization, but are not yet implemented.

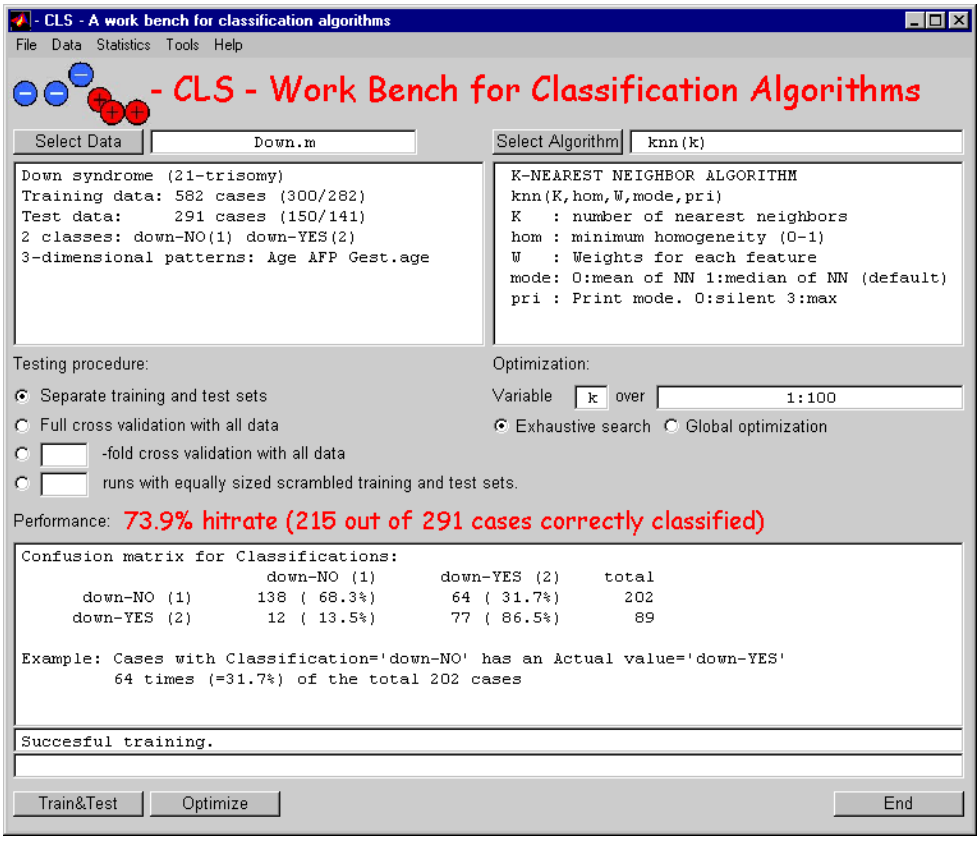


Figure 8

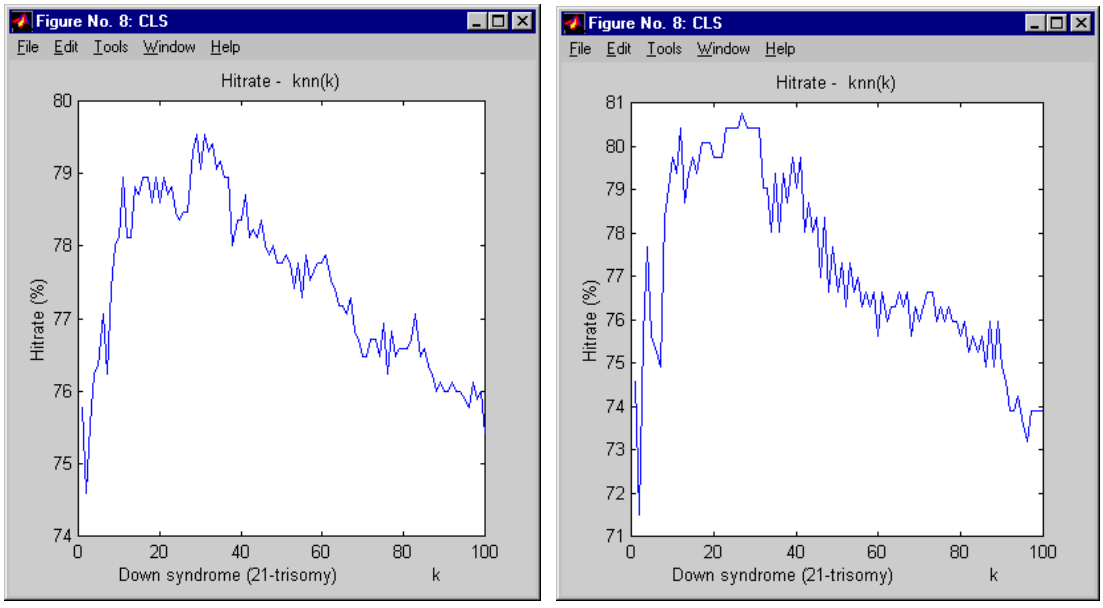


Figure 9

The Command Line Version CLS0

As described in the previously, CLS can be called from the command line in Matlab as well. This is particularly useful when setting up a systematic test scheme for a certain classification algorithm or for a certain data set. The procedure for using the command-line version of CLS is as follows:

- Select a data set from one of the supplied examples (listed in Section Supplied Data Sets) on the \data directory, or set up your own data in either ASCII format or as a .mat file. The data format is described in Appendix 1.
- Choose one of the predefined classification algorithms (listed in Section Supplied Classification Algorithms.)
- Call the function cls0. Example:

```
p=cls0('knn(9)','breastcancer.m');
» p.hitrate
ans =
   94.1935
» p.perf1.ncount
ans =
    69     3
     6    77
```

The shown example is also run in the Windows version in Figure 2.

The help text for *CLS* is shown below:

```
function perf = cls0(testfkn, problem, par)
Stand alone call function for cls.
Tests a classification algorithm on a data set and returns the
classification performance.
testfkn : string with function call to classifier
problem : string with name of mat file or m file with cls data.
par     : Optional struct with additional parameters:
          par.confmatrices
             set to 1 to enable computation of confusion mat.,
             perf.perf1.ncount(i,j) is in such case the # of
             cases where the classification is "i" and the
             actual value is "j".
          par.errorhandling 1=enabled (default)
          par.printoption : 0=silent (default)
          par.testmode :
              1:separate train and test set  (default)
              2:full cross validation
              3:n-fold cross validation  where n should be set
                in par.nfold
              4:n runs with random data sets where n should be
                set in par.nruns
Example of usage:
cls0('knn(9)','ex1')
```

Data Transformations

A few tools for data transformation are available from the Tools menu. The data can be transformed with the following functions:

- 0-1 normalization
- Each feature is scaled so it covers the 0-1 range. The transformation is performed for Training data and Test data separately.
- Gaussian normalization
- Each feature is transformed by subtracting the mean and dividing by the estimated standard deviation. The transformation is performed for Training data and Test data separately.
- Principal components
- All features are linearly transformed through a PCA. The transformation is performed for Training data and Test data separately.
- Add noise
- The classifications for 10% (randomly selected) of the training and test data are set to random classes (only for binary classifications).

Appendix 1

Data Format

CLD can read data of the following 3 types:

- ASCII files with numeric data
  The classification should be in one column and the rest of the features in the other columns. Use the *genericascii.m* option in the Select Data function from the CLS screen.
- Interfaced data where raw data, labels, etc. are defined in a short Matlab script for each data set.
- .mat files generated by the "Save data as .mat" command in the Data menu. This format can be used to speed up data handling once it has been read into CLS using one of the other data formats.

The first type (ASCII files) is straight forward and requires no further explanation.

The second type (a specific interface script) is described below.

Data scripts

Data may be interfaced to the CLS system through a Matlab script customized for each data set. This script may extract data from any source (most often by reading raw data from files,) and should return the data in predefined variables that CLS reads while performing the data analysis. The process of interfacing a data set is best explained by an example. The following is a listing of the file *down.m* which interfaces data for classification of the Downs Syndrome.

```
[ptrain, ctrain,ptest, ctest,ok]=
loadfiles('downlrn1.data','downtes1.data');
if ok
   features={'Age' 'AFP' 'Gest.age'};
   classlabels={'down-NO' 'down-YES'};
   classes=[0 1];
   title='Down syndrome (21-trisomy)';
end
```

The *loadfiles* function is a utility which reads ASCII data from files. More information can be found further down in this Appendix.
The variables that should be defined by the script are:

- ptrain
  A matrix with features for the observations in the training data set. Row *i* in *ptrain* contains the features for row *i*.
- ptest
  The same syntax as *ptrain* but for the test data set.
- ctrain
  A vector with the classification for each observation in the training data set. Classifications should be integers.
- ctest
  The same syntax as *ctrain* but for the test data set.
- Features (optional)
  A cell array of strings with the names of the features
- classlabels
  A cell array of strings with the names of the classes.
- title
  A string with a description of the data set

loadfiles

The function *loadfiles* is a convenient way to read ASCII files into the matrices and vectors that the CLS system requires. It was used by the script described above and the following Help text for the function:

```
function [ptrain,ctrain,ptest,ctest,ok]=
loadfiles(trainfile,testfile,trainlines,testlines,pcols,ccol,fmt)
% Loads ascii files with numeric data for classification problems.
% The files should contain lines with equal number of separated
% field. Valid separators are white space and comma.
%
% In parameters:
% trainfile  : name of file with training data
% testfile   : name of file with test data
% trainlines : rows to use for training data. E.g: [ 1 100] for the %
%              first 100 lines in file
%              Default [] for all rows in file
% testlines  : rows to use for training data. E.g: [ 1 100] for the
%              first 100 lines in file
%              Default [] for all rows in file
% pcols      : Vector with column numbers for patterns. Default []  %
%              for all but the last column.
%              E.g: 2:9 for columns 2 up to 9.
% ccol       : Column number for class identity for pattern
%
% fmt        : Format string for sscanf. E.g: '%i,%i,%s,%i,%i'
%              %s-fields will be converted to corresponding ascii
%              integer.
%              Only one-character strings are allowed.
%              Default ''  will uses str2num instead of sscanf anf
%              works  fine most of the times.
```

Appendix 2

Algorithm Interface

New classifiers can be easily interfaced to the CLS system by writing a simple interface routine and placing it on the *algorithms* directory under the CLS installation directory. The layout of the interface routine is described by an example:

```
function c = A0(p1,p2,pN)
% ------------------------------------------------------------------------
--
% function c = A0(p1,p2,pN)
% Template for algorithm-interface for the cls system.
%
% An algorithm file is called in two modes:
%
% if cls.Train
%    The model should be trained on cls.Ptrain and cls.Ctrain.
%    c should be returned as '' if the training went ok.
%    otherwise it should contain a suitable error message
% else
%    the (trained) model should be applied on cls.Ptest and
%    produce classifications which are returns in c.
%    On error, c should return a string with an error text.
% end
%
% Written 6th of Dec 1999. Last modified 6th of Dec 1999
% Thomas Hellstrom Umea Sweden. Email: thomash@cs.umu.se
% ------------------------------------------------------------------------
--
% The following comments are the automatic interface for the WCLS system
% They will be shown when the algorithm is selected in WCLS.
% The comment lines should start with %x where x~=' '.
% Row 1: default function call. Row 2: Description. Rows 3- : Free text
% Example:
%- A0([5 3 0],300,0.01)
%- A hell of a Classifier!!!
%- A0(p1,p2,pN)
%- p1   : explanatory text
%- p2   : explanatory text
%- pN   : explanatory text
%
% The blank comment line terminates the comment section.

% cls is the structures with all variables for the wcls system
global  cls
% It is often convenient to save the results of the algorithm training
% in a global variable that can be accessed when the classifier should be
% applied on data (i.e. the th emodes of theis reoutine):
global net

% It ia good practise to define defaults for the in parameters:
if nargin<1, p1 = [5 3 0]; end
if nargin<2, p2=300; end
if nargin<3, p3=0.01; end
```

```
% This routine has two modes: training and not training.
% The training mode should train the model and save the result.
% The non training mode should used the trained model and classify
% data points cls.Ptest:
if cls.Train
   % A0train should return '' iff training went fine. Otherwise it should
   % return a string with an error message:
   net = A0train(cls.Ptrain, cls.Ctrain, p1,p2,p3);
   c = '';
else
   % A0class should return a vector with classifications for each element in
cls.Ptrain:
   % If an error is encountered, A0class should return an error message.
   c = A0class(net,cls.Ptest);
end
return
```