CMAT© Newsletter: July 2016

Wolfgang M. Hartmann

July 2016

Contents

1 General Remarks

This newsletter refers to the new version 9 of CMAT which has been moved to the internet during the last days of September. IMHO, it is the best tested version for quite some time. I'm now 72.5 years old and will have some break from new developments of the CMAT language. I will still do some consulting work, especially for the research department of my wife's Lab at the University of Heidelberg.
A very serious bug was fixed for the `conting()` function.
The `rename` statement was added to the CMAT language. Note, the term `rename` is a token and can no longer be used as name of a variable or user function. The advantage compared to a new function is, that there is no need for some (temporary) copy of the data object.
For the events-trial specification of a model for the `glim()` funtion, the number of trial column in the data set must no longer be specified as an option. A number of bugs of the `glim()` function, especially when using a FREQ data set variable, were corrected. See also the appendix for some illustration of the `glim()` function.
The new function `outlier()` implements the following methods for finding outliers in onedimensional data:

- χ^2 method (see Dixon, 1950), this is also in R
- 3-Sigma rule, 3-MAD rule
- Tukey method for testing the interquartile range (Tukey, 1977b)
- Chauvenet (1863) Criterion
- Grubbs (1969) test, this is also in R
- Thompson (1985) τ test
- Dixon Q test (Dixon, 1950; Rohrabacher, 1991; McBane, 2006), this is similar of that in R

Dixon's test is also with the `outlier() package of R, however, using interpolated table values by Rohrabacher (1991) instead of the numerically more exact computed by McBane (2006). (BTW, I did review Mc Baneś paper for JSS. And I remember the old Prof. Dixon from an interview I had with BMDP and him in the mid 90ies in Los Angeles. At that time Prof. Dixon still had an office in the BMDP building but, as he told me, he then did only consulting and no longer was writing code for BMDP.)`
`Please, note, that the old function outl() which imlemented such multidimensional methods like MVE (minimum volume ellipsoid), MCD (minimum covariance determinant), and multivariate kurtosis, has now been renamed to outlmd() to avoid mistakes.`

The "task" and "meth" options in function svm() have been rather confusing and I have decided to separate some code for feature selection from that function into two new functions:

svmfsm() default feature selection algorithms for SVM classification (FSM method by Fung and Mangasarian, 2002, 2003) and regression (ϵ regression by Bi, Bennett, Embrechts, Breneman, & Song, 2002),

svmstw() forward and backward stepwise feature selection algorithms for SVM classification and regression (Guyon & Elisseeff, 2003).

The stepwise function svmstw() is designed **only** for linear kernel function. Whereas the two algorithms of the svmfsm() function subject the parameter estimates to an $L1$ constraint and are supposed generating sparse (linear or nonlinear) parameter vectors, the stepwise approach is using stepwise svm solutions for adding or dropping a model effect to or from the model generating a solution with a specified number of nonzero coefficients.

- The forward stepwise algorithm adds in each step that effect which is expected to improve the model fit most.
- The backward stepwise algorithm drops in each step that effect which is expected to reduce the model fit least.

Both functions, svmfsm() and svmstw() have no need for the "task" options and only the svmstw() function permits to specify the estimation method used in every step with the "meth" option.
By splitting off some code from the svm() function its usage became easier and the new function svmstw() is now much more enhanced as it has been when it was a task of svm().
For returning the results of parameter grid tuning, where the option "tun" to "grid" is specified, the functions svm() and svmfsm() have been extended by two additional return arguments:

tunerr returns a $p0 \times p1 \times ...$ matrix or tensor of error estimates, where $p0 = 2$ or $p0 = 4$ depends on the fact whether a test set is available or not. The $p0$ categories of the first dimension are:

1. for training data:
   - classification: number of misclassification
   - regression: for training data: $L1$ fit value
2. for training data: $L2$ fit value
3. for test data: $L1$ fit value
4. for test data: $L2$ fit value

tunzer this return is for svm() nonmissing only when one of the two methods of the svmfsm() function are specified which are supposed to return sufficiently sparse parameter estimates. Returned are the (integer) numbers of nonzeros

- either in the linear parameters $plane[n]$
- or the nonlinear parameters $\alpha[N]$

Then there are two forms of return objects depending on the type of the response variable:

- for regression and binary classification: returns a $p1 \times p2 \times ...$ vector, matrix, or tensor of the number of nonzeros, where $p1, ...$ correspond to the parameters of the tuning grid.
- for multicategorical classification: returns a $p0 \times p1 \times ...$ matrix or tensor where $p0$ is the number of levels of the response variable and $p1, ...$ correspond to the parameters of the tuning grid.

The output of tensors and lists was improved by permitting the specification of tensor dimension and list entry names and labels.

1.1 New Functions

error() print an error message into the log.

fnampid() create file name with PID suffix

fremove() remove file specified with path

frename() change file name

lstname() specifying names to list entries (max length is 32 chars)

lstlabel() specifying labels to list entries (max length is 80 chars)

outlier() implements various methods for detecting outliers in onedimensional data

pid() returns integer process ID of current CMAT run

svmfsm() default feature selection algorithms:

- for SVM classification: the $L1$ algorithm by Fung & Mangasarian (2003)
- for SVM regression: the $L1$ algorithm by Bi, Bennett, Embrechts, Breneman, Song (2002)

This function can be used also for nonlinear kernel functions, but the methods are fixed.

svmstw() stepwise feature selection algorithms:

- backward stepwise for approaching the *all-relevant* problem,
- forward stepwise: for approaching thw *minimal-optimal* problem.

This function is designed only for linear kernel, however is able to use different estimation methods.

warning() print a warning into the log.

1.2 Fixed Bugs

Some very ugly bug when printing matrices of mixed numeric-string type was fixed. The bug was introduced when this subroutine was modified to print strings longer than the actual line length (which can be specified by the LS runtine option).
A bug of the pcx() function was fixed which occurred when the algorithm did not converge.

2 Modifications of Features

2.1 Functions `lp()`, `mpswrite()`, and `mpsread()`

The meaning of missing values has been changed when specifying lower and upper bounds for the lp() function:

lower bounds specifying missing values refers to the absolutely largest negative number ninf = -macon("mbig"); of the computer arithmetic (in double precision).

upper bounds specifying missing values refers to the absolutely largest positive number pinf = macon("mbig"); of the computer arithmetic (in double precision).

Before a missing value specified for the lower bound defaulted to zero.

2.2 Tensor Printed Output

By adding the tensor output to the functions svm() and svmfsm(), the form of the printed output of tensors has been enhanced. The following svm() application illustrates the new output:

```
  print "NIR Spectra data set: train: nr=21, test: nr=7";
  options NOECHO;
#include "..\\tdata\\nir.dat"
  options ECHO;
  nr = nrow(xtrn); nc = ncol(xtrn);
  print "nrtrn,nctrn=",nr,nc;


  cdat = xtrn -> ytrn‘; /* attrib(cdat); */
  test = xtst -> ytst‘; /* attrib(test); */
  /* print cdat[ , 200:269 ]; */
  /* model: */
  xind = [ 1:268 ]; yind = 269;
  modl = "269 = 1:268";
```

When specifying the following 3-dimensional $4 \times 5 \times 4$ grid

```
  tun = [ "C"   .1 1. 10. 100. ,
          "NU"  .01  .1  .2  .5  .8 ,
          "K1"  .001 .01 .05 .1  ];
```

with the svm() function (see $test\backslash tsvm22.inp$)

```
print "GRID Tuning with RBF2 kernel: Use FLP with PCX";
optn = [ "print"           3 ,
         "popt"            1 ,
         "task"      "nureg" ,
         "meth"        "flp" ,
         "tun"        "grid" ,
         "tunsel"     "test" ,
         "kern"       "rbf2" ];
< alfa,sres,vres,yptr,yptt,plan,tunerr,tunzer > = svm(cdat,modl,optn,.,tun,.,.,test);
print "Alfa=",alfa;
print "Tunerr=",tunerr;
print "Tunzer=",tunzer;
```

we obtain among others the following results:

Best 10 Results of Grid Search

N	C_	NU	K1	L1err	Error
65	100.00000	0.1000000	0.0010000	513.670120	3595.69084
					467.139272
74	100.00000	0.5000000	0.0100000	822.006211	5754.04348
					898.063568
78	100.00000	0.8000000	0.0100000	845.780588	5920.46411
					908.902507
72	100.00000	0.2000000	0.1000000	900.749625	6305.24738
					616.371468
76	100.00000	0.5000000	0.1000000	900.749625	6305.24738
					616.371468
80	100.00000	0.8000000	0.1000000	900.749625	6305.24738
					616.371468
54	10.000000	0.5000000	0.0100000	943.434902	6604.04432
					952.321539
58	10.000000	0.8000000	0.0100000	989.503760	6926.52632
					1102.12925
67	100.00000	0.1000000	0.0500000	1050.66376	7354.64632
					1141.20986
68	100.00000	0.1000000	0.1000000	1095.13793	7665.96553
					846.205380

```
**************************************************************
4-Dim. Tensor : Training and Test: Regression L1 and L2 Error
**************************************************************
```

Dim[1]= Type_of_Fit (4)
Dim[2]= C_Param (4)
Dim[3]= Nu_Param (5) : Rows
Dim[4]= Kernel_1 (4) : Columns

Dim[1]: [1]= L1_Err(Train)
Dim[2]: [1]= C_0.1

	K1_0.001	K1_0.01	K1_0.05	K1_0.1
NU_0.01	1953.3	1953.3	1953.3	1953.3
NU_0.1	1953.3	1953.3	1953.3	1953.3
NU_0.2	1953.3	1953.3	1953.3	1953.3
NU_0.5	1953.3	1953.3	1953.3	1953.3
NU_0.8	1953.3	1953.3	1953.3	1953.3

Dim[1]: [1]= L1_Err(Train)
Dim[2]: [2]= C_1

	K1_0.001	K1_0.01	K1_0.05	K1_0.1
NU_0.01	1953.3	1953.3	1953.3	1953.3
NU_0.1	1953.3	1953.3	1953.3	1953.3
NU_0.2	1953.3	1953.3	1953.3	1953.3
NU_0.5	1953.3	1953.3	1953.3	1953.3
NU_0.8	1953.3	1953.3	1953.3	1953.3

Dim[1]: [1]= L1_Err(Train)
Dim[2]: [3]= C_10

	K1_0.001	K1_0.01	K1_0.05	K1_0.1
NU_0.01	1953.3	1953.3	1953.3	1953.3
NU_0.1	1953.3	1953.3	1953.3	1953.3
NU_0.2	1953.3	1953.3	1953.3	1953.3
NU_0.5	1953.3	952.32	1771.4	1953.3
NU_0.8	1953.3	1102.1	1313.4	1953.3

Dim[1]: [1]= L1_Err(Train)
Dim[2]: [4]= C_100

	K1_0.001	K1_0.01	K1_0.05	K1_0.1
NU_0.01	1953.3	1953.3	1953.3	1953.3
NU_0.1	467.14	1446.6	1141.2	846.21
NU_0.2	3156.1	1493.1	1354.9	616.37
NU_0.5	2259.8	898.06	1533.5	616.37

```
   NU_0.8 |     2617.8     908.90     1533.5     616.37

.....................................................
```

```
                    Dim[1]: [4]= L2_Err(Test)
                    Dim[2]: [4]= C_100
          |   K1_0.001    K1_0.01    K1_0.05     K1_0.1
-----------------------------------------------------------
  NU_0.01 |    8641.26    8641.26    8641.26    8641.26
   NU_0.1 |    3595.69    11445.8    7354.65    7665.97
   NU_0.2 |    23136.1    11474.3    8385.51    6305.25
   NU_0.5 |    15783.1    5754.04    9359.41    6305.25
   NU_0.8 |    18423.4    5920.46    9359.41    6305.25
```

And the following shows the number of nonzeros of the $N = 21$ parameter vector α:

```
 ********************************************************************
 3-Dim. Tensor : FSM Grid Search: N Nonzero Coefficients (In Alfa)
 ********************************************************************

                Dim[1]= C_Param (4)
                Dim[2]= Nu_Parm (5) : Rows
                Dim[3]= Kernel_1 (4) : Columns
```

```
                       Dim[1]: [1]= C_0.1
          |   K1_0.001    K1_0.01    K1_0.05     K1_0.1
-----------------------------------------------------------
  NU_0.01 |          0          0          0          0
   NU_0.1 |          0          0          0          8
   NU_0.2 |          0          0          0          9
   NU_0.5 |          0          0          0          0
   NU_0.8 |          0          0          1          0
```

```
                       Dim[1]: [2]= C_1
      LOW |   K1_0.001    K1_0.01    K1_0.05     K1_0.1
-----------------------------------------------------------
  NU_0.01 |          0          0          0          0
   NU_0.1 |         21         11          0          0
   NU_0.2 |         16          0          0          0
   NU_0.5 |          0          0          0          0
```

```
   NU_0.8 |          0          0         15         19

                      Dim[1]: [3]= C_10
          |   K1_0.001    K1_0.01    K1_0.05     K1_0.1
----------------------------------------------------------
  NU_0.01 |          0          1          0          0
   NU_0.1 |         21         21          1         21
   NU_0.2 |          0          1         11          0
   NU_0.5 |          1          5         18         12
   NU_0.8 |         21          5          4          0

                      Dim[1]: [4]= C_100
          |   K1_0.001    K1_0.01    K1_0.05     K1_0.1
----------------------------------------------------------
  NU_0.01 |          0          1          1          1
   NU_0.1 |          3          7         14         12
   NU_0.2 |          4          7         17         20
   NU_0.5 |          4         12         20         20
   NU_0.8 |          4         13         21         20
```

3 Extensions to the Language

3.1 The rename *vnew* = *vold*; Statement

The rename statement with the following syntax rename *vnew* = *vold*; was added, where *vold* is the name of an existing variable and *vnew* is the new name it should be referenced from now on in CMAT code. Whenever a variable with name *vold* does not exist or a variable with name *vnew* exists already, an error message is printed into the log and there will be no change of the variable structures. Note, that rename is a token and that this string no longer can be used for variable or function names.
The rename result can also be obtained using a simple assignment followed by a call of free:

```
a = [ 10 ,
      2 12 ];
rename b = a;
print "b=", b;


a = b; free b;
print "a=", a;
```

The advantage of using rename is however, that it does not need a copy of the object and therefore may be much more efficient in memory need and computation time.

4 Extensions to Various Functions

4.1 Extension of conting(),glim(), gee(), ... Function

The function conting() and all statistical functions which may have a binary or multinomial response variable, like glim(), the list of association coefficients now also contains the c coefficient

$$c = \frac{n_c + .5(t - n_c - n_d)}{t},$$

the Gini coefficient denoted as *Somers D*,

$$d = \frac{(n_c - n_d)}{t},$$

Goodman-Kruskal γ

$$\gamma = \frac{(n_c - n_d)}{n_c + n_d},$$

and Kendall's $\tau - a$

$$\tau_a = \frac{(n_c - n_d)}{.5 * N * (N - 1)},$$

where t is the number of all pairs, n_c the number of concordant pairs, n_d the number of discordant pairs, and $t - n_c - n_d$ is the number of tied pairs.

4.2 Extensions of svm() Function

The following two tasks were removed from the svm() function and are now implemented into the svmstw() function:

"sfsc" stepwise feature selection for SV classification,

"sfsr" stepwise feature selection for SV regression,

implementations of the greedy feature selection method by Guyon et.al. (2003).
The list of method names has been greatly simplified. Modifications of the main methods are specified by the new "modif" option.
Some specifications of tuning parameters were removed from the optn argument to a newly designed ctun input object. The new input argument ctun is $K \times k$ data object where $K \geq 1$ and $k \geq 1$ are valid choices:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of C,

- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function
     "K4" fourth parameter of kernel function
     "C" regularization parameter C
     "NU" NU classification or regression ν
     "EP" eps value for epsilon regression
     "AL" parameter α for FSM classification
     "DE" parameter δ for FSM classification
  2. remaining columns $k = 2, ...$:
     - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)
     - for "tun" = "patt" or "tun" = "opti": ctun must have only two columns $k = 2$ specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

The following options are no longer permitted:

Option Name	Sec.	Col.	Meaning
"abeg"		real	valid only for internal grid search L1FS: starting value for α parameter;
"afac","aend"		real	valid only for internal grid search L1FS: factor and end value for α parameter;
"c"		real	regularization parameter C
"dbeg"		real	valid only for internal grid search L1FS: starting value for δ parameter;
"dfac","dend"		real	valid only for internal grid search L1FS: factor and end value for δ parameter;
"epsi"		real	eps value for epsilon regression
"kfp1"		real	first parameter of kernel function
"kfp2"		real	second parameter of kernel function
"kfp3"		real	third parameter of kernel function
"nu"		real	parameter for NU classification or regression

The following is the new doc of function svm():

`alfa = svm(train,model,<,optn<,class<,ctun<,x0<,kfun<,test>>>>>>)`

`<alfa,sres,vres,yptr,yptt,plan,tunerr,tunzer> = svm(train,model,<,optn<,...>)`

Purpose: The term *support vector machines* is used for a set of predictive modeling algorithms

1. to solve the *binary classification problem* when the response variable (target) is binary (defined with class),
2. to solve the *multinomial classification problem* when the response variable (target) is multinomial with $k > 2$ categories (defined with class),
3. to solve the (nonlinear kernel) *regression problem* i.e. compute a linear or nonlinear regression w.r.t. a specific loss function when the response variable is interval scaled (response is not defined with class).

The most traditional application for SVM is for the binary classification problem. In the following, let N be the number of observations (cases, rows) and n be the number of variables of an $N \times n$ data set $\mathbf{X}$.

This svm() function is able to model the following problems (tasks):

C Classification: Different versions are implemented:

1. The original dual model which is a QP with one linear equality constraint and lower and upper bounds. Methods: "FQP", "DQP", "SMO"
2. The LS SVM modification of the classification problem by Suykens and Vandewalle (1999). Methods: "LSQU"
3. The Mangasarian & Musicant modifcation of the dual model which is a QP with only lower and upper bounds. Methods: "FQP", "DQP"
4. Further modifcations by Mangasarian which even get rid of the upper bounds. Methods: "RSVM", "PSVM", "LSVM", "LCH", "ASVM" for $N > n$ and "NSVM" for $N < n$.
5. Another modification by Fung and Mangasarian which is very useful for variable selection when $N < n$. Method: "FSM".

ν Classification: 1. The original dual model which is a QP with two linear equality constraints and lower and upper bounds. Methods: "FQP", "SMO"

2. The Mangasarian & Musicant modifcation of the dual model which is a QP with one linear equality constraint and lower and upper bounds. Methods: "FQP"

$L2$ **Regression:** 1. The dual model which is a QP with one linear equality constraint and lower and upper bounds. Methods: "FQP", "DQP"

2. The LS SVM modification of the regression problem by Suykens and Vandewalle (1999). Methods: "LSQU"

ϵ **Regression:** 1. The dual model which is a QP with one linear equality constraint and lower and upper bounds requiring twice the number of parameters. Methods: "FQP", "SMO"

2. The primal model which is a LP with a large number of linear and boundary constraints requiring about four times the number of parameters. Methods: "FLP"

ν **Regression:** 1. The original dual model which is a QP with two linear equality constraints and lower and upper bounds requiring twice the number of parameters. Methods: "FQP", "SMO"

2. The primal model which is a LP with a large number of linear and boundary constraints requiring about four times the number of parameters. Methods: "FLP"

For a table of formulas of the different models see below.

The svm() function implements a number of algorithms for parameter estimation together with methods for Loo (Leave-one-out, Jackknife), k fold cross validation, and *XiAlpha* estimation. Together with Loo estimation a sensitivity analysis is performed obtaining a list of observations which are the largest model misfits (when deleted improve the goodness of fit). You can select from a broad list of common kernel function, but you may also specify your own kernel function using the the kfun input argument.

Three methods for the tuning of the regularization parameter C and all kernel function parameters are provided. The "tun" option can have three string values, "grid", "patt", or "opti":

"grid" a grid search for specified values of C and all kernel function parameters is performed. The ctun input argument must contain a $K \times nk$ matrix, where K is 1 (for C) plus the number of kernel parameters. Each row contains a number of nonzero values for which the search is performed. The first row always corresponds to values for C.

"patt" a bounded (pattern) search is performed inside a specifed (hyper-) rectangle. The ctun input argument must contain a $K \times 2$ matrix, where K is 1 (for C) plus the number of kernel parameters. The first column contains a lower and the second column an upper bound value restricting the rectangular search area. The algorithm is a very simple but rather robust search at neighborhood points in a hypercube region.

"opti" a bounded (pattern) optimization is performed inside a specifed (hyper-) rectangle. The ctun input argument must contain a $K \times 2$ matrix, where K is 1 (for C) plus the number of kernel parameters. The first column contains a lower and the second column an upper bound value restricting the rectangular search area. The currently used Nelder-Mead simplex algorithm, however, does not seem to be very successful and may soon be replaced by something more appropriate.

The functions svmpred() and svmmat() are closely related to the svm() function. Using svmpred() an additional test data set can be scored using the parameter estimates from the svm() training. The function svmmat() can be used to compute a linear or nonlinear kernel matrix. See page ?? for more details on support vector machines.

Input: train specifies a $N \times m$ matrix or vector of training data, i.e. data which are used to obtain parameters that specify a specific model. For solving the binary classification problem the response variable must be a binary CLASS variable.

model : The analysis model is specified in form of a string, e.g. model= "3=1 2", containing column numbers for variables. The syntax of the model string argument is the same as for the glmod() function except for the additional *events / trial* response specification. ????

optn The option argument is specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See below for details.

class specifies a vector CLASS variables in form of a vector of integers which defining the column numbers of the training data set.

ctun The form of input depends on the "ctun" option:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of C,
- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function

"K4" fourth parameter of kernel function

"C" regularization parameter C

"NU" NU classification or regression ν

"EP" eps value for epsilon regression

"AL" parameter α for FSM classification

"DE" parameter δ for FSM classification

2. remaining columns $k = 2, ...$:
   - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)
   - for "tun" = "patt" or "tun" = "opti": ctun must have only two columns $k = 2$ specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

x0 specifies a vector of nr initial estimates α. If the response (target) is multinomial with K categories, this must be a $K \times nr$ matrix. That means this input argument corresponds in size to the first output argument α.

kfun optional string argument specifying a user defined function for kernel evaluation. The function definition must be of the form: function kernel(vi,vj) where the input arguments vi and vj are observation vectors of the data and the function must be a real scalar. Note, that Mercer's condition must be satisfied by the functions kernel definition, i.e. the resulting matrices **Q** of the QP must be positive (semi)definite.

test optional argument specifying a matrix or vector of test data, i.e. data for which predicted model values should be obtained based on the parametric model obtained from the training data. The column structure for the test data set should be the same as that of the training data set since the model applies also at the test data.

Content of the Options Matrix :

Option Name	Sec.	Col.	Meaning
"abstol"		real	termination criterion for optimization
"absftol"		real	termination criterion for optimization
"absgtol"		real	termination criterion for optimization
"absxtol"		real	termination criterion for optimization
"algv"		int	
"best"		int	only valid with Loo estimation: lists the $best$ observations which when deleted improve most the goodness-of-fit
"bias"		real	specifies a numeric scalar used as a model constant b for the predicted values
"biascor"		"y"	apply bias on predicted values default for linear kernel
		"n"	do not apply bias on predicted values default for nonlinear kernel
"block"		int	fold number for block cross validation
"cgp"			"LCH", "FSM", "PSVM", "LSVM", "LSQU" (with "icg"): use incomplete Cholesky preconditioning
"cgt"			"LCH", "FSM", "PSVM", "LSVM", "LSQU" (with "icg"): tolerance for conjugate gradient iteration
"cgit"			"LCH", "FSM", "PSVM", "LSVM", "LSQU" (with "icg"): maximum iterations for conjugate gradient
"chunk"		string	sample method for chunking method "LCH"
		"blo"	blockwise sampling (default)
		"spl"	split sampling
		"ran"	random sampling
"clas"			perform classification (categorical response)
"cloo"			compute Loo (leave-one-out) estimates
"clow"		real	lower range for regularization parameter C; default is 1.e-6
"cval"		string	method of cross validation
		"blo"	blockwise sampling (default)
		"loo"	leave-one-out
		"set"	
		"spl"	split sampling
		"ran"	random sampling
"cxia"			compute xi-alpha estimates
"design"		string	design of class variables
		"ful"	full rank (default)
		"ran"	rank deficient

Option Name	Sec. Col.	Meaning
"epsr"		compute epsilon regression ("eps" must be specified)
"freq"	int	column number of frequency variable
"fold"	int	fold number for block, split, and random cross validation
"ftol"	real	function termination criterion for optimization
"gtol"	real	gradient termination criterion for optimization
"icg"		"LCH", "FSM", "PSVM", "LSVM", "LSQU": use iterative CG method for Cholesky factor
"kern"	string	kernel function, default is linear
	"line"	linear function
	"poly"	polynomial function
	"rbf"	Gaussian radial basis function (def. scaling $[0,1]$)
	"rbf2"	mod. Gaussian radial basis function (def. scaling $[0,1]$)
	"rbfcs"	mod. Gaussian radial basis function (def. scaling $[0,1]$)
	"erbf"	exponential radial basis function
	"tanh"	sigmoid function (same as "sigm")
	"sigm"	sigmoid function (same as "tanh")
	"four"	Fourier function (def. scaling $[-\pi/2, \pi/2]$)
	"spli"	spline function (def. scaling $[0,1]$)
	"anov"	anova function
	"curv"	curvspline function (def. scaling $[0,1]$)
	"bspl"	Bspline function (def. scaling $[0,1]$)
	"anob"	anova spline function (def. scaling $[0,1]$)
	"ano1"	modified anova function (def. scaling $[0,1]$)
	"ano2"	modified anova function (def. scaling $[0,1]$)
	"ano3"	modified anova function (def. scaling $[0,1]$)
"kfp1"	real	first parameter of kernel function
"kfp2"	real	second parameter of kernel function
"kfp3"	real	third parameter of kernel function
"ktest"		compute inertia of kernel matrix, i.e. number of positive, negative, and zero eigenvalues
"maxfu"	int	max. number function calls of optimization (see nlp
"maxit"	int	max. number iterations of optimization (see nlp
"maxtim"	real	max. duration of optimization (see nlp
"memsiz"	int	memory threshold for storing train data incore; in Megabyte; default=2Mb; all training data incore for "memsiz"=0

Option Name	Sec. Col.	Meaning
"meth"		method of estimation, default depends on problem
	"asvm"	for classification: linear $L2$ SVM similar to Mangasarian & Musicant
	"fsm"	for classification: $L1$ sparse or $L2$ feature selection:
	"fsm"	"modif" =0 : Fung & Mangsarian $L1$ FSM algorithm
	"fsm"	"modif" =1 : Fung & Mangsarian $L2$ NSVM algorithm
	"flp"	currently only for ϵ and ν regression and only for linear kernel (feature selection): Bi et.al (2002)
	"flp"	"modif" =0 : using PCx algorithm (Czyzyk et al.)
	"flp"	"modif" =1 : using LP algorithm by Madsen et al.
	"flp"	"modif" =2 : using CLP algorithm by J. J. Forrest
	"fqp"	for classification and regression: full QP may be modified with "mmm" for Mangasarian & Musicant modification; null and rage space technique
	"fqp"	"modif" =1 : for classification and regression: full QP: modified version: use Penalty method
	"dqp"	for classification and regression: decomposed QP (my version) may be modified with "mmm" for Mangasarian & Musicant modification
	"dqp"	"modif" =1 : for classification and regression: decomposed QP (modified SVM Light) may be modified with "mmm" for Mangasarian & Musicant modification
	"dqp"	"modif" =2 : for classification and regression: decomposed QP (original SVM Light) may be modified with "mmm" for Mangasarian & Musicant modification
	"lsqu"	for classification and regression: least squares SVM: similar to proximal SVM and ridge regression; direct (dense or sparse) Cholesky or iterative conjugate gradient
	"lsvm"	for classification: Lagrangean $L2$ SVM similar to Mangasarian & Musicant
	"lch"	for classification: linear $L1$ SVM chunking similar to Mangasarian & Thompson for $N \gg n$ and only for linear kernel (feature selection)
	"nsvm"	for classification: $L2$ feature selection Newton SVM similar to Fung & Mangasarian, only for linear kernel (feature selection)
	"psvm"	for classification: Proximal $L2$ SVM similar to Fung & Mangasarian; direct (dense or sparse) Cholesky or iterative conjugate gradient
	"rsvm"	for classification: reduced smooth SVM similar to Lee & Mangasarian
	"smo"	for classification and regression: similar to Lin & Chen
	"smo"	"modif" =1 : SMO (sequential minimal opt.) by

Option Name	Sec. Col.	Meaning
"mmm"		solve Mangasarian-Musicant form of SVM classification problem; i.e. replace one linear equality constraint
"modif"	int	modification of "method" option (see "method")
"modmis"		model missing response category
"nchunk"	int	number of chunks for chunking method "LCH"
"nseq"	int	maximum number of sub QP for "DQP"
"nobstat"		print observation statistics
"nolin"		specifies NO line search for "FSM" method
"nomis"		do not use observations that contain missing values in one of the variables used in analysis
"nopr"		no printed output in ...txt (but warnings or errors are in ...log)
"noscale"		supress scaling of predictor and interval response
"nucl"		compute NU classication ("nu" must be specified)
"nure"		compute NU regression ("nu" must be specified)
"order"		order categories of class variables
	"dat"	in order as they appear in the rows of data matrix
	"asc"	sorted in ascending order
	"des"	sorted in descending order
"outli"	int	same as "best", only for Loo estimation
"pall"		is equivalent to "print" 10
"pcrv"		print cross validation tables
"peneps"	real	tuning parameter: 0. < peneps < 1., only for "FSM" and "LCH": default=.1 for "FSM"; default=.001 for LCH
"pinit"		print input information and model setup
"popt"	int	amount of printed output in optimization environment (i.e. in inner loop), default is 0
"pmat"		print kernel matrix (this can be big)
"pmps"		print of MPS data structure
"ppar"		print vector of parameter estimates α
"pplan"	int	$>$ 0 print coefficients for linear plane; valid only for linear kernel
"ppred"		print predicted values
"print"	int	amount of general printed output (default is 3)
"pshr"	real	percentage of f reduction for shrinking
"pshort"		less than default output
"pstart"	int	number random starting points in pattern search (default=4)
"psum"		much less than default output
"ptun"	int	amount of printed output in parameter tuning (i.e. in outer loop), default is 0
"qpsiz"	int	size of QP for "DQP" method

Option Name	Sec. Col.	Meaning
"qpmanp"		Madsen-Nielsen-Pinar QP technique (only for for boundary constrained problem, Mang.-Mus. mod. C classification)
"qpnusp"		null space QP technique
"qprasp"		range space QP technique
"qptron"		Lin-Moré TRON QP technique (only for for boundary constrained problem, Mang.-Mus. mod. C classification)
"random"	int	fold number for random cross validation
"ransam"		valid only for "RSVM": perform random sampling
"relsam"		valid only for "RSVM": perform relevance sampling
"redu"		set maximum size of sub QP to number of variables in data for "DQP"
"regr"		perform L2 SV regression (interval response)
"ridg"	real	ridge diagonal of kernel matrix
"rho"	real	parameter for Loo estimates
"samsiz"	int	size of sample for "RSVM"
"scale"	"isot"	isotropic scaling (SVM regression only)
	int	if not zero, perform scaling of predictor $\mathbf{X}$; this is default for several kernel functions this will **also** scale the response (target) y
"seed"	int	seed for random generator; default is time of day; affects the results of random cross validation: affects results of "RSVM" and random starting values
"shri"	int	test function reduction every shri iterations for shrinking the sub QP (only for "DQP")
"sing"	real	singularity criterion, default is 1.e-8
"smoeps"	real	epsilon parameter for SMO
"split"	int	fold number for split cross validation
"start"	string	method for obtaining starting values
	"zero"	α_0 is set to zero
	"rand"	α_0 is uniformly random
	"prox"	α_0 is computed by PROX algorithm
	"qua"	valid only for SVM regression
	"qp0"	valid only for "FQP" method
"task"	strung	form of SVM
	"clas"	perform C classification: methods "FQP", "DQP", "LCH", "ASVM", "RSVM", "LSVM", "PSVM", "SMO"; "LSQU", "FSM", "NSVM"
	"nucl"	perform NU classification: methods "FQP" and "SMO"
	"regr"	perform $L2$ regression methods "FQP", "LSQU", and "DQP"
	"nure"	perform NU regression: methods "FLP", "FQP", and "SMO" (for FLP "eps" is estimated, "nu" must be specified)
	"epsr"	perform EPSilon regression: methods "FLP", "FQP", and "SMO" (for FLP "eps" and "nu" must be specified)

Option Name	Sec. Col.	Meaning
"tech"	string	specifies optimization for "FQP" and "DQP"
	"qpmanp"	Madsen-Nielsen-Pinar QP technique (BC only)
	"qpnusp"	null space QP technique
	"qprasp"	range space QP technique
	"qptron"	Lin-Moré TRON QP technique (BC only)
"tun"	string	specifies form of parameter tuning
	"grid"	perform grid search for tuning
	"patt"	perform pattern search for tuning
	"opti"	perform Melder-Mead optimization for tuning
"tunitr"	int	max iterations for tuning
"tuntim"	int	max time in seconds for tuning
"tunftol"	real	f termination for tuning
"tunxtol"	real	x termination for tuning
"tunsel"	"test"	for tuning use test data evaluating goodness-of-fit (default if "tun" is specified)
	"train"	for tuning use training data evaluating goodness-of-fit (default if "tun" is not specified)
"vers"	string	version of CLP method
	"iprim"	
	"idual"	
	"sprim"	
	"sdual"	
"weight"	int	column number of weight variable
"xiad"		depth for XiA estimates
"xtol"	real	termination criterion
"yscale"		specifies the scale type of the response:
	"i"	interval (use SV regression)
	"n"	nominal (use SV classification)
	"o"	ordinal (currently treated like nominal)

The following are the default settings for the termination criteria of the QP optimization algorithms:

Method	ABSFTOL	FTOL	ABSGTOL	GTOL	ABSXTOL	XTOL
FQP	.	.	1e-8	.	0	1e-8
DQP	.	.	1e-6	.	0	1e-8
LSVM	.	.	.	.	1e-5	1e-5
SMO	.	.	1e-3	.	.	.
General	10*eps	1e-12	1e-5	1e-8	1e-5	1e-8

Here eps is the machine precision, which is the smallest number added to one which results in a value greater than one. On a IBM PC this is about 10^{-15}.

Note: "freq" and "weight" are not working currently.

Output: alfa returns the parameters of the model. This is either a

N row vector or a $K \times N$ matrix, where N is the number of observations of the training data set, $K = 1$ for interval or binary response, and $K > 2$ for multinomial target (response) with $K > 2$ categories. If "nomiss" is specified, then N excludes observations with missing predictor variables.

sres returns a vector of scalar data describing the execution and some scalar results. See below for details.

vres returns either a row vector or matrix. A $K \times p$ matrix is returned for multinomial target (response) with $K > 2$ categories. See below for details.

yptr returns the N row vector of predicted value $\hat{(y)}_i$ for the training data set.

yptt (only when the input argument test is specified) returns the vector of predicted values $\hat{(y)}_i$ for the test data set.

plan can only be returned for linear kernel and is either a $n_{eff}+1$ vector or $K \times (n_{eff}+1)$ matrix if the response is multinomial with $K > 2$ categories. The first n_{eff} columns contain the coefficient of the linear model with effects in the order of the model input specification. The last column contains the intercept of the linear model.

tunerr only if the option "tun" to "grid" is specified, returns a $p0 \times p1 \times ...$ matrix or tensor of error estimates, where $p0 = 2$ or $p0 = 4$ depends on the fact whether a test set is available or not. The $p0$ categories of the first dimension are:

1. for training data:
   - classification: number of misclassification
   - regression: for training data: $L1$ fit value
2. for training data: $L2$ fit value
3. for test data: $L1$ fit value
4. for test data: $L2$ fit value

tunzer only if the option "tun" to "grid" is specified, this return is for svm() nonmissing only when one of the two methods of the svmfsm() function are specified which are supposed to return sufficiently sparse parameter estimates. Returned are the (integer) numbers of nonzeros

- either in the linear parameters $plane[n]$
- or the nonlinear parameters $\alpha[N]$

Then there are two forms of return objects depending on the type of the response variable:

- for regression and binary classification: returns a $p1 \times p2 \times ...$ vector, matrix, or tensor of the number of nonzeros, where $p1, ...$ correspond to the parameters of the tuning grid.

- for multicategorical classification: returns a $p0 \times p1 \times$ ... matrix or tensor where $p0$ is the number of levels of the response variable and $p1, ...$ correspond to the parameters of the tuning grid.

Note, that "scoring" (computing predicted values) of an additional test data set using the svmpred() function may be much more efficient (in memory and computer time) using the parameters in plan than those in alfa.

Contents of the sres vector:

1. the value of C (for the best C solution when multiple C values are specified).
2. error of training data (when multiple C values are specified: the best value)
   (a) for SVM Classification number of misclassifications
   (b) for SVM Regression average sum-of-squares error

$$\frac{\sum_i^N (y - \hat{y})^2}{N}$$

3. error of test data (see above)
4. total number of kernel calls as indicator for the computational expense
5. total computer time in seconds
6. computer time for optimization in seconds

Column contents of the vres vector or matrix:

1. number of support vectors, i.e. the number of nonzero entries of α.
2. number of support vectors on margin, i.e. the number of entries of α on the upper bound specified by option optn[8].
3. function value for QP solution $f = \alpha \mathbf{Q} \alpha^T + q^T \alpha$
4. L_1 function value for QP solution
5. maximum absolute gradient value for QP solution $g_{max} = max_{i=1}^N (\mathbf{Q}\alpha^T + q)_i$
6. beta value
7. estimated SV radius (radius of ball containing the SV) $rad = max_{i \in SV} \sqrt{K(x_i, x_i) - 2. * K(x_i, 0) + K(0, 0)}$
8. the bias which is computed as the mean of the residuals defined as difference between observed target y and predicted target values $\hat{y}$ for the support vectors with $alpha_i > 0$.

9. $wnorm = \alpha^T * \mathbf{Q} * \alpha$
10. estimated margin: $margin = 2./\sqrt{wnorm}$
11. estimated VC dimensionality $wnorm * rad^2 + 1$.

Restrictions: 1. There is no complex version of this function.

2. Observations with missing values in the X data (predictors) are not used.
3. Note, that for interfacing with CLP a temporary _svm....mps file is written into the work directory. The unique process ID is added to the file name to prevent confusion when running multiple processes at the same computer.

Relationships: svmmat(), svmpred(), svmfsm(), svmstw(), mvsvm(), smsvm(), qp(), lp(), nlreg(), nlp()

Types of Some SVM Estimation Problems Here, $e = (1, \ldots, 1)$, $y \in [-1, 1]$, $\mathbf{Q}$ is the $N \times N$ kernel matrix, and $\mathbf{A}$ is the $N \times m$ data matrix.

C Classification 1. Primal:

$$\min \frac{1}{2} w^T w + C e^T \xi$$

subject to:

$$y_i(w^T K_i + b) \geq 1 - \xi_i \quad , \quad \xi_i \geq 0 \quad , \quad i = 1, \ldots, N$$

2. Dual:

$$\min \frac{1}{2} \alpha^T \mathbf{Q} \alpha - e^T \alpha$$

subject to:

$$y^T \alpha = 0 \quad , \quad 0 \leq \alpha_i \leq C \quad , \quad i = 1, \ldots, N$$

3. Mangasarian-Musicant modification of the Dual:

$$\min \frac{1}{2} \alpha^T (\mathbf{Q} + yy^T) \alpha - e^T \alpha$$

subject to:

$$0 \leq \alpha_i \leq C \quad , \quad i = 1, \ldots, N$$

4. In "SVM" and "LSVM" we optimize for linear kernel $\mathbf{Q} = \mathbf{A}\mathbf{A}^T$:

$$\min \frac{1}{2} \alpha^T (\frac{\mathbf{I}}{C} + diag(y)(\mathbf{A}\mathbf{A}^T + ee^T) diag(y)) \alpha - \alpha^T e$$

subject to: $0 \leq \alpha$

ν **Classification** `1. Primal:`

$$\min \frac{1}{2} w^T w - \nu\rho + e^T \xi / N$$

`subject to:`

$$y_i(w^T K_i + b) \geq \rho - \xi_i \quad , \quad \xi_i \geq 0 \quad , \quad i = 1, \ldots, N$$

`2. Dual (scaled):`

$$\min \frac{1}{2} \alpha^T \mathbf{Q} \alpha$$

`subject to:`

$$y^T \alpha = 0 \quad , \quad e^T \alpha = N\nu \quad , \quad 0 \leq \alpha_i \leq 1 \quad , \quad i = 1, \ldots, N$$

`3. Mangasarian-Musicant modification of the Dual (scaled):`

$$\min \frac{1}{2} \alpha^T (\mathbf{Q} + yy^T) \alpha$$

`subject to:`

$$e^T \alpha = N\nu \quad , \quad 0 \leq \alpha_i \leq 1 \quad , \quad i = 1, \ldots, N$$

$L2$ **Regression** `1. Primal:`

`2. Dual:`

$$\min \frac{1}{2} \alpha^T \mathbf{Q} \alpha - y^T \alpha$$

`subject to:`

$$e^T \alpha = 0 \quad , \quad -C \leq \alpha_i \leq C \quad , \quad i = 1, \ldots, N$$

ϵ **Regression** `1. Primal:`

$$\min \frac{1}{2} w^T w + Ce^T \xi + Ce^T \xi^*$$

`subject to:`

$$(w^T K_i + b) - y_i \leq \epsilon + \xi_i \quad , \quad y_i - (w^T K_i + b) \leq \epsilon + \xi_i^* \quad , \quad i = 1, \ldots, N$$

`2. Dual:`

$$\min \frac{1}{2} (\alpha - \alpha^*)^T \mathbf{Q} (\alpha - \alpha^*) + \epsilon e^T (\alpha + \alpha^*) + y^T (\alpha + \alpha^*)$$

`subject to:`

$$e^T (\alpha - \alpha^*) = 0 \quad , \quad 0 \leq \alpha_i, \alpha_i^* \leq C \quad , \quad i = 1, \ldots, N$$

ν Regression 1. Primal:

$$\min \frac{1}{2} w^T w + C[\nu\epsilon + \frac{1}{N} e^T(\xi + \xi^*)]$$

subject to:

$$(w^T K_i + b) - y_i \leq \epsilon + \xi_i \quad , \quad y_i - (w^T K_i + b) \leq \epsilon + \xi_i^* \quad , \quad i = 1, \ldots, N$$

2. Dual (scaled):

$$\min \frac{1}{2} (\alpha - \alpha^*)^T \mathbf{Q} (\alpha - \alpha^*) + y^T (\alpha - \alpha^*)$$

subject to:

$$e^T(\alpha - \alpha^*) = 0 \quad , \quad e^T(\alpha + \alpha^*) = NC\nu \quad , \quad 0 \leq \alpha_i, \alpha_i^* \leq C \quad , \quad i = 1, \ldots, N$$

Estimation Methods (Algorithms) :

SVM Classification 1. "FQP": Fully Dense QP (Quadratic Programming): Only small applications up to $N = 1000$ observations can be solved due to storing the dense $N \times N$ matrix $\mathbf{Q}$. Both, C and ν classification problems can be solved with "FQP". Two versions can be used:

- The ordinary method specified by "method" "fqp" solves the original problem with one linear constraint and lower and upper boundary constraints for each parameter of the dual SVM problem.
- The modified method specified by "method" "fqp" and option "mmm" solves the Mangasarian-Musicant dual SVM problem using only lower and upper boundary constraints.

Four different QP optimization methods are available:

(a) null space version (default when linear constraint is involved, i.e. "mmm" is not specified)

(b) range space version (sometimes faster than null space method)

(c) TRON method by Mor'e and Lin

(d) Madsen, Nielsen, Pinar method can only be used when "mmm" is specified, i.e. model does not contain a linear constraint.

Method "FQP" takes advantage of good starting values when multiple C values are specified.

2. "DQP": Decomposition QP Methods: These methods are based on papers of Osuna et al. which state rules for partitioning the set of parameters into two sets, a small part which is being "optimized" and the remaining larger part which

is kept constant. That means the large QP is solved by an iterative process solving one much smaller QP in each iteration. Kaufmann (1998) uses slightly different rules for selecting the active variables. Two implementations are posted on the net:

SVM^{light} by T. Joachims (1999) which is using LOQO by Vanderby and Shanno. This method solves the original problem with the linear constraint. A modification of this method is available by specifying "method" "dqp".

BSVM by C.W. Hsu and C-J. Lin (1999), which is using the QP solver TRON from Argonne Institute. This method solves the Mangasarian-Musicant modification of a similar but only boundary constrained problem. A modification of this method is available by specifying "method" "dqp" and "mmm".

Both methods are completely new designed in function svm(). They work with an adaptive shrinking and expanding strategy. This method takes advantage of good starting values when multiple C values are specified.

3. "LSQU": Least Squares SVM (Suykens et.al.) For classification, this method is very similar to the "PSVM" algorithm by Fung & Mangasarian. It only requires to solve a (ridged, i.e. positive definite) large $N \times N$ linear system. Two solution methods are implemented:
   (a) The direct Cholesky method (dense or sparse) requires the coefficient matrix in memory and is suitable for only small N.
   (b) The iterative conjugate gradient algorithm does not require the coefficient matrix incore and computes the matrix-vector product by creating the kernel matrix entries "on the fly". This approach, however, requires the $N \times n$ data matrix incore.

   Both, LS SVM for classification and for regression were developed and researched by the Suykens and Vandevalle team of the Kath. University of Leeuven. The method is (like "PSVM") very fast but creates nonsparse parameter vectors.
4. "RSVM" (Reduced Support Vector Machine):
   "SSVM" (Smooth Support Vector Machine):
   This method is only suitable for C classification and is appropriate for $N > n$. These algorithms were developed by Lee and Mangasiarian Musicant (1999) and use a similar simplified objective function. The modified problem has no longer a linear equality constraint and no lower or

upper bounds. "RSVM" which is a modification of SSVM which selects a sample of observations for constructing a rectangular (nonlinear) kernel matrix. This method can be applied to very large optimization problems with general nonlinear kernels, but the quality of the parameter estimates depends on the selected sample which is used for the rectangular kernel matrix. For linear kernels other methods like "ASVM" or "PSVM" can yield better results and may be much faster.

5. "ASVM" (Active Support Vector Machine):
   "LSVM" (Lagrangean Support Vector Machine):
   "PSVM" (Proximal Support Vector Machine):
   These methods are only suitable for C classification and should only be used for $N > n$. These algorithms were developed by Mangasiarian and Musicant (1999,2000) and use an even more simplified objective function than that used in BSVM by C.W. Hsu and C-J. Lin (1999). The modified QP problem has no longer a linear equality constraint and no upper bounds. For "PSVM" even negative parameter estimates are possible. For "ASVM" and "LSVM" the only constraints are nonnegativity constraints. The original versions by Mangasarian and Musicant are useful especially for linear kernels by applying the Sherman-Morrison-Woodbury formula which reduces the large $N \times N$ kernel matrix $\mathbf{Q}$ to the (for most applications much smaller) $(m + 1) \times (m + 1)$ matrix, where m is the number of variables, needs to be inverted in each iteration. The papers show that data sets with several million observations can be processed (with all data in core). The algorithms can easily be modified to have only larger data chunks in core. For linear kernels, the "LSVM" and "ASVM" algorithms perform very similar. However, the simple implementable "LSVM" algorithm is patented. The "PSVM" method is also available in svm() as a method for computing starting values which can be used with other estimation methods. "ASVM" is probably the fastest algorithm for linear kernels. "ASVM" is at this time not available for nonlinear kernels. If specified, it will be changed to SSVM.
6. "NSVM" ($L2$ Feature Selection Method):
   "FSM" ($L1$ Feature Selection Method):
   These methods are only suitable for C classification and should only be used for $N < n$. They are designed for feature selection using linear kernel. Its extension for nonlinear kernel seems is probably of less interest. Both methods are developed by Fung & Mangasarian and are mainly designed for problems with $N \ll n$. The L_2 Newton method

"NSVM" is fast converging and similar to "FSM" but creates dense coefficient vectors and is not suitable for feature selection.
The convergence of then L_1 method "FSM" depends on additional method parameters and normally needs tuning for the $\alpha > 0$ and $\delta > 0$ (see "ctun" input option).
Armijo line search is default, but maybe suppressed by option "nolin". But it may create very sparse coefficient vectors suitable for *feature selection*, i.e. the selection of a few highly predictive variables from a very large set of predictors.

7. "SMO": Sequential Minimal Optimization: Originally developed 1997 by J. Platt, now at Microsoft. This is the extreme of the decomposition method since in each sequence only two parameters α_1 and α_2 are changed. Starting from a feasible solution two parameters can be easily changed so that the nonnegativity constraints $\alpha_i \geq 0$ and the equality constraint $y^T\alpha = 0$ can be maintained. There are some modifications of this algorithm varying in the strategy of selecting the two parameters. The two parameters are selected so that their change promises a significant reduction of the value of the objective function. The modifications have to do with the fact that the original SMO algorithm may be patented by Microsoft.
A method similar to that of LIBSVM, by Hsu and Lin (1999) who developed the BSVM method (LIBSVM is available in R) which permits shrinking of the parameter space, is available for both, C and ν classification. In addition, the original SMO algorithm by Platt can be run by specifying "modif" 1, the two Keerthi modifications can be run by specifying "modif" 2 and "modif" 3.
8. "NPA": Nearest Point Algorithm: Keerthi et al (1999) developed an iterative algorithm for the binary SVM classification problem. For inseperable data sets the violations are quadratically penalized, what means that the objective function of the primal problem is slightly changed to

$$\min \frac{1}{2} w^T w + \frac{C}{2} \sum_{i=1}^{N} \xi_i^2$$

which has the advantage that the nonnegativity constraints can be dropped. Similar modifications are used by some other very fast algorithms. With a number of test examples the authors illustrate that NPA is slightly faster than SMO (measured by the number of kernel evaluations). Since

this method is not significant faster than SMO it was implemented here.

SVM Regression 1. "FQP": Fully Dense QP (Quadratic Programming): Only small applications up to $N = 1000$ observations can be solved due to storing the large dense matrix $\mathbf{Q}$.

L2 SVR lower and upper bounded QP with $N \times N$ matrix $\mathbf{Q}$ and one linear constraint

ϵ **SVR** lower and upper bounded QP with $2N \times 2N$ matrix $\mathbf{Q}$ and one linear constraint

ν **SVR** lower and upper bounded QP with $2N \times 2N$ matrix $\mathbf{Q}$ and two linear constraints

The dual problem can be solved by one of the following QP optimization methods:

(a) null space version (default when linear constraint is involved, i.e. "mmm" is not specified)

(b) range space version (sometimes faster than null space method)

The "FQP" method takes advantage of good starting values when multiple C values are specified.

2. "SMO": Sequential Minimal Optimization: A method similar to that of LIBSVM (available in R) which permits shrinking of the parameter space, is implemented for both, ϵ and ν regression. Using "modif" 1 Platt's original method and using "modif" 2 and "modif" 3 Keerthi's modified methods can be used.

3. "FLP": Fully Dense LP (Linear Programming): This method is currently only used for ν regression for linear kernel. It is especially useful for feature selection when $N \ll n$, see Bi et.al. (2002). For obtaining sparse coefficient vectors this method requires the specification of two tuning parameters $C > 0.$ using the first row of input argument ctun and $\nu \in (0., 1.]$ using the second row of input argument ctun. The "modif" option can be used for specifying different LP methods. Three algorithms can be used currently for solving the very large and dense LP problem,

   - the *pcx* algorithm by Czyzyk, Mehrotra, Wagner, and Wright (1997) and Wright (1997) (method "FLP"),
   - the continuation algorithm LPasL1 by Madsen, Nielsen, and Pinar (1996) (method "FLP" and "modif"= 1),
   - the CLP algorithm by J. J. Forrest (method "FLP" and "modif"= 2).

   For very large problems the CLP algorithm is the fastest and numerically the most stable, and the default interior point algorithm *pcx* seems to be numerically more stable

compared to the continuation algorithm, which could be faster for well behaved problems.

4. "LSQU": Least Squares SVM (Suykens et. al.) This algorithm can be used for both, C classification and $L2$ regression. For regression the "LSQU" method is very similar to *ridge regression*. It only requires to solve a (ridged, i.e. positive definite) large $N \times N$ linear system. Two solution methods are implemented: direct (dense or sparse) Cholesky and iterative conjugate gradient method.

Selection of Estimation Algorithm

- C Classification with Linear Kernel:

  The following is just a first iterate, it needs more computational experience in detail:

Size	small n	large n
small N	"FQP"	"FQP"
medium N	"LSQU"	"DQP","SMO","LSQU"
large N	"DQP","PSVM","LCH","ASVM","LSVM"	"LCH","DQP","SMO"

- C Classification with NonLinear Kernel:

Size	small n	large n
small N	"FQP"	"FQP"
medium N	"PSVM","LSVM","LSQU"	"PSVM","LSVM","LSQU"
large N	"DQP","SMO","RSVM"	"LCH","DQP","SMO","RSVM"

- $L2$ Regression:

Size	small n	large n
small N	"FQP"	"FQP"
medium N	"LSQU"	"LSQU"
large N	"DQP"	"DQP","LSQU"

- Nu and Eps Regression:

Size	small n	large n
small N	"FQP"	"FQP"
large N	"SMO"	"SMO"

Some Important Kernel Definitions Assuming u and v are two observations of the data set and p_1, p_2, and p_3 are specified scalar values.

linear

$$K(u, v) = u^T v$$

polynomial

$$K(u, v) = (p\_3 u^T v + p\_2)^{p\_1}$$

with polynomial degree p_1; where defaults are: $p\_1 = 2, p\_2 = p\_3 = 1$.

"rbf"

$$K(u,v) = \exp(-\frac{(u-v)^2}{2\sigma^2})$$

Gaussian radial basis function, where default $\sigma = p\_1 = \sqrt{(}1./n)$

"rbf2"

$$\mathbf{K}(u,v) = exp(-p\_1 * (u-v)^2)$$

default: $p\_1 = 1./n$

rbfc

$$\mathbf{K}(u,v) = \max(0, (1 - \frac{(u-v)^2}{\theta})^\kappa) exp(-\frac{(u-v)^2}{\sigma^2})$$

the *compactly supported Gaussian RBF* kernel (Hamers, Suykens, & deMoor, 2002), where $\theta = 3. * p\_1, \kappa = even(n, n+1), \sigma = p\_1$, default $\sigma = p\_1 = \sqrt{(}1./n)$

erbf

$$K(u,v) = \exp(-\frac{|u-v|}{2\sigma^2})$$

Exponential radial basis function, produces piecewise linear solutions, where default $\sigma = p\_1 = \sqrt{(}1./n)$

sigmoid (also called *tanh*)

$$K(u,v) = \tanh(p\_1 u^T v - p\_2)$$

where p_1 is scale and p_2 is location (*multi layer perceptron with single hidden layer*), default: $p\_1 = 1./n, p\_2 = 1$,

fourier

$$K(u,v) = \frac{\sin(N + .5)(u-v)}{\sin(\frac{1}{2}(u-v))}$$

The data are scaled by default in $[-\pi/2, \pi/2]$.

spline

$$K(u,v) = \sum_{j=1}^{p} u^j v^j + \sum_{i=1}^{K} (u - \tau_i)_+^p (v - \tau_i)_+^p$$

where p is the order and τ_i are the knots; e.g.

$$K(u,v) = \prod_{j=1}^{m} z_j$$

with

$$z = 1 + u \odot v + u \odot v \min(u,v) - \frac{1}{2}(u+v)\min(u,v)^2 + \frac{1}{3}\min(u,v)^3$$

The data are scaled by default in $[0,1]$.

anova (also called *curvespline*)

$$K(u,v) = \prod_{j=1}^{m} z_j$$

with

$$z = 1 + u \odot v + \frac{1}{2} u \odot v \min(u,v) - \frac{1}{6}(u+v)\min(u,v)^3$$

where the $\odot$ operator denotes elementwise multiplication

bspline

$$K(u,v) = \prod_{j=1}^{m} z_j$$

with

$$z = \sum_{k=1}^{2p+1} (-1)^k \left(\begin{array}{c} 2(p+1) \\ k \end{array} \right) \max(0, u - v + p + 1 - k)^{(2p+1)}$$

The data are scaled by default in $[0,1]$.

anova spline

$$K(u,v) = \prod_{j=1}^{m} z_j$$

with

$$z = 1 + u \odot v + u \odot v \min(u,v) - \frac{1}{2}(u+v)\min(u,v)^2 + \frac{1}{3}(u+v)\min(u,v)^3$$

anova bspline

$$K(u,v) = \prod_{j=1}^{m} (1 + z_j)$$

with

$$z = \sum_{k=1}^{2p+1} (-1)^k \left(\begin{array}{c} 2(p+1) \\ k \end{array} \right) \max(0, u - v + p + 1 - k)^{(2p+1)}$$

i.e. is the same as **bspline** except for the $1+$ term in the product. The data are scaled by default in $[0,1]$.

Other kernel functions can be user specified using the fifth input argument which is a string naming a function kernel(u,v) defined byt the user. Kernel functions must satisfy Mercer's condition to result in positive (semi)definite Hessian matrices $\mathbf{Q}$ of the dual problem. What does a Kernel function? It applies a nonlinear transformation on the location of the points x_i in the m dimensional

space. After that transformation, simply the linear model is applied.

Some of the kernel functions, especially those based on the exp function, like all rbf functions, can create kernel matrices which are diagonal or even identity matrices for extrem parameter settings. using the identity matrix for the kernel matrix produces useless results.

Examples: The heart_scale data set contains 13 predictor variables and 270 observations creating a QP problem with 270×270 positive semidefinite (rank=13) Kernel Hessian matrix. Here we specify a *radial basis function* Kernel definition.

The first set of results shows a comparison of the four QP techniques (QPNUSP, QPRASP, QPMANP, and QPTRON) on the full 270×270 only bound constrained optimization problem. This is the modification of the objective function proposed by Mangasarian and Musicant (1999) which does not impose the original linear constraint

$$\sum_{i=0}^{N} \alpha_i y_i = 0$$

We show the complete printed output only for the first run (null space technique). For the other optimization techniques we only show the output which differs and which is related to the computer resources needed by each technique.

To save space we only show the specification of some examples here. For more information, especially the printed output you are referred to the *Details* chapter, see page ??.

1. Binary SVM Classification: Heart Data RBF2 Kernel
   (a) SVM Classification: with LC
       i. FQP: C is estimated

```
data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           2 ,
         "ppred"             ,
         "popt"            1 ,
         "meth"       "fqp" ,
         "kern"      "rbf2" ];
tun = [ "K1"    .076923076 ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

       ii. FQP: C is specified
           Now we specify c as a vector.

```
data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;
tun = [ "C"  .01 .1  1. 10. 100. ,
        "K1"  .076923076 ];
optn = [ "print"           2 ,
         "popt"            1 ,
         "meth"        "fqp" ,
         "kern"       "rbf2" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

There are 5 optimizations for different C values. Predicted values and results are returned only for the run which shows the smallest misclassification.

iii. DQP: C is specified:

```
data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;
tun = [ "C"  1. 10. 100. ,
        "K1"  .076923076 ];
optn = [ "print"           2 ,
         "meth"        "dqp" ,
         "kern"       "rbf2" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

iv. SMO: C is set: always QPsize=2

To illustrate SMO a smaller data set of only 60 cases is selected.

```
data = rspfile("heart_60.dat");
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           2 ,
         "meth"        "smo" ,
         "kern"       "rbf2" ];
tun = [ "C"   100. ,
        "K1"  .076923076 ];
< alfa,sres,vres,yptr,yptt > = svm(data,modl,optn,class,tun);
```

v. SMO, modif=2 (Keerthi): C is set: always QPsize=2

```
data = rspfile("heart_60.dat");
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           2 ,
         "meth"       "smo"  ,
```

```
                "modif"          2  ,
                "kern"       "rbf2" ];
        tun = [ "C"   100. ,
                "K1"  .076923076 ];
        < alfa,sres,vres,yptr,yptt > = svm(data,modl,optn,class,tun);
```

(b) SVM Classification: w/o LC

i. FQP: C is estimated

```
        data = rspfile("heart_scale.dat");
        modl = "1 = 2 : 14";
        class = 1;
        optn = [ "print"           2 ,
                 "ppred"             ,
                 "popt"            1 ,
                 "mmm"               ,
                 "meth"        "fqp" ,
                 "kern"       "rbf2" ];
        tun = [ "K1"    .076923076 ];
        < alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

ii. FQP: C is specified

```
        data = rspfile("heart_scale.dat");
        modl = "1 = 2 : 14";
        class = 1;
        optn = [ "print"           2 ,
                 "ppred"             ,
                 "popt"            1 ,
                 "mmm"               ,
                 "meth"        "fqp" ,
                 "kern"       "rbf2" ];
        tun = [ "C"  .01 .1  1. 10. 100. ,
                "K1"  .076923076 ];
        < alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

iii. DQP: C is specified

```
        data = rspfile("heart_scale.dat");
        modl = "1 = 2 : 14";
        class = 1;
        tun = [ "C"  1. 10. 100. ,
                "K1"  .076923076 ];
        optn = [ "print"           2 ,
                 "mmm"               ,
                 "meth"        "dqp" ,
                 "kern"       "rbf2" ];
```

```
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

iv. PSVM: C is specified

```
data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;
tun = [ "C"  1. 10. 100. ,
        "K1"  .076923076 ];
optn = [ "print"           2 ,
         "mmm"               ,
         "meth"       "psvm" ,
         "kern"       "rbf2" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

v. LSVM: C is estimated

```
data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;
tun = [ "C"  1. 10. 100. ,
        "K1"  .076923076 ];
optn = [ "print"           2 ,
         "mmm"               ,
         "meth"       "lsvm" ,
         "kern"       "rbf2" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

(c) SVM with LC: FQP: Comparison of Techniques

Null Space Technique

```
data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           2 ,
         "popt"            1 ,
         "kern"       "rbf2" ,
         "meth"        "fqp" ,
         "tech"     "qpnusp" ];
tun = [ "K1"    .076923076 ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

Madsen-Nielsen-Pinar Technique

```
data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           2 ,
```

```
          "popt"            1 ,
          "kern"       "rbf2" ,
          "meth"        "fqp" ,
          "tech"     "qpmanp" ];
  tun = [ "K1"    .076923076 ];
  < alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

Lin-Moré TRON Technique

```
  data = rspfile("heart_scale.dat");
  modl = "1 = 2 : 14";
  class = 1;
  optn = [ "print"           2 ,
           "popt"            1 ,
           "kern"       "rbf2" ,
           "meth"        "fqp" ,
           "tech"     "qptron" ];
  tun = [ "K1"    .076923076 ];
  < alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

(d) SVM with LC: FQP: User Specified Kernel Function

The computation time increases by about a factor 10 for a specified kernel function. Therefore, only the first 60 observations of the heart_scale.dat are being used in heart_60.dat.

```
  /* radial basis kernel function */
  function krbf(vi,vj) {
    w = vi - vj;
    t = exp(-.076923076 * w * w‘);
    return(t);
  }

  data = rspfile("heart_60.dat");
  modl = "1 = 2 : 14";
  class = 1;
  optn = [ "print"           1,
           "meth"        "fqp" ];
  < alfa,sres,vres,yptr > = svm(data,modl,optn,class,.,krbf);

  data = rspfile("heart_60.dat");
  modl = "1 = 2 : 14";
  class = 1;
  optn = [ "print"           2,
           "meth"       "dqp" ];
  < alfa,sres,vres,yptr > = svm(data,modl,optn,class,.,krbf);
```

2. Multinomial SVM Classification: Fisher's Iris Data; RBF2 Kernel

(a) SVM Classification: with LC

```
#include "iris.txt"
   nr = nrow(iris); irs5 = iris[,5];
   count = cons(3);
   for (i = 1; i <= nr; i++) count[irs5[i]] += 1;
   print "Member Count=", count;
```

i. FQP: C is specified

```
  class = 5;
  modl = "5 = 1 : 4";
  optn = [ "print"         3 ,
           "ppred"           ,
           "popt"          1 ,
           "meth"      "fqp" ,
           "kern"     "rbf2" ];
  tun = [ "C"          1. ,
          "K1"         .1 ];
  alfa = svm(iris,modl,optn,class,tun);
```

ii. DQP: C is specified

```
  #include "iris.txt"
  class = 5;
  modl = "5 = 1 : 4";
  optn = [ "print"           3 ,
           "meth"        "dqp" ,
           "kern"       "rbf2" ];
  tun = [ "C"          1. ,
          "K1"         .1 ];
  alfa = svm(iris,modl,optn,class,tun);
```

(b) SVM Classification: without the linear constraint:

i. FQP: C is specified

```
  class = 5;
  modl = "5 = 1 : 4";
  optn = [ "print"         3 ,
           "ppred"           ,
           "popt"          1 ,
           "mmm"             ,
           "meth"      "fqp" ,
           "kern"     "rbf2" ];
  tun = [ "C"          1. ,
          "K1"         .1 ];
```

```
alfa = svm(iris,modl,optn,class,tun);
```

ii. DQP: C is specified

```
#include "iris.txt"
  class = 5;
  modl = "5 = 1 : 4";
  optn = [ "print"           3 ,
           "mmm"               ,
           "meth"        "dqp" ,
           "kern"       "rbf2" ];
  tun = [ "C"         1. ,
          "K1"        .1 ];
  alfa = svm(iris,modl,optn,class,tun);
```

iii. PSVM: C is specified

```
#include "iris.txt"
  class = 5;
  modl = "5 = 1 : 4";
  optn = [ "print"           3 ,
           "mmm"               ,
           "meth"       "psvm" ,
           "kern"       "rbf2" ];
  tun = [ "C"         1. ,
          "K1"        .1 ];
  alfa = svm(iris,modl,optn,class,tun);
```

iv. LSVM: C is specified

```
#include "iris.txt"
  class = 5;
  modl = "5 = 1 : 4";
  optn = [ "print"           3 ,
           "mmm"               ,
           "meth"       "lsvm" ,
           "kern"       "rbf2" ];
  tun = [ "C"         1. ,
          "K1"        .1 ];
  alfa = svm(iris,modl,optn,class,tun);
```

3. XiAlpha Estimates, Loo, and k-fold Cross Validation

To illustrate the approach we use the first 60 observations of the Heart data set. The following input specifies XiA estimates of depth 0 and Loo (Leave-one-out, jackknife) estimation:

```
data = rspfile("heart_60.dat");
```

```
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           2 ,
         "cloo"              ,
         "cxia"              ,
         "xiad"            0 ,
         "meth"        "fqp" ,
         "kern"       "rbf2" ];
tun = [ "C"           1. ,
        "K1"  .076923076 ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

The following input specifies XiA estimates of depth 0 and 4-fold cross validation:

```
/* FQP: CV with fold=4 */
data = rspfile("heart_60.dat");
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           2 ,
         "cloo"              ,
         "fold"            4 ,
         "cxia"              ,
         "xiad"            0 ,
         "meth"        "fqp" ,
         "kern"       "rbf2" ];
tun = [ "C"           1. ,
        "K1"  .076923076 ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

The following Leave-one-out (Loo) specification also requests a sensitivity analysis which leads to a table output of the ten largest outlier observations;

```
data = rspfile("..\\tdata\\heart_60.dat");
modl = "1 = 2 : 14";
class = 1;
optn = [ "print"           3 ,
         "cloo"              ,
         "best"           10 ,
         "kern"       "line" ,
         "meth"        "fqp" ];
 < alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

Now we show the specifications of three cross validation methods, block, split, and random sampling, with a fold of ten:

```
data = rspfile("..\\tdata\\heart_60.dat");
modl = "1 = 2 : 14";
class = 1;
tun = [ "C"   .001 .1 1. 10. 100. ];

optn = [ "print"           3 ,
         "block"          10 ,
         "kern"       "line" ,
         "meth"        "fqp" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,tun);
```

Usually split sampling will find better results than block sampling which is sensitive to ordered training data:

```
optn = [ "print"           3 ,
         "split"          10 ,
         "kern"       "line" ,
         "meth"        "fqp" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

The random sampling will find different results by either not specifying the "seed" option (time of day is default) or by specifying different "seed" values:

```
optn = [ "print"           3 ,
         "random"         10 ,
         "kern"       "line" ,
         "meth"        "fqp" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class,c);
```

For more examples see tsvm2.inp and tsvm10.inp.

4. Parameter Tuning by Grid and Pattern Search

   Remember, when specifying "tun" by default the fit is evaluated using the (left out) validation data. If that is changed with the "tunsel" option, usually the best solution corresponds to the largest C value. As an example, we use the 60 first observations of the Heart data set and the mean misclassiction error of 4-fold cross validation:

```
data = rspfile("..\\tdata\\heart_60.dat");
modl = "1 = 2 : 14";
class = 1;
```

For grid search tuning we are specifying 5 grid values for C and 13 grid values for the parameter of the RBF kernel function we should have to evaluate 65 cross validation, each of them with $fold$ optimizations:

```
  /* tuning by grid search: specify values */
  ctun = [ "C"   .001 .1 1. 10. 100. ,
           "K1"  .2 : .05 : .8 ];
```

We first apply a grid search tuning with Loo. This is time consuming, since $N*65$ optimizations are needed, but each with good starting values.

```
  optn = [ "print"           3 ,
           "popt"            1 ,
           "tun"        "grid" ,
           "cloo"              ,
           "kern"       "rbf2" ,
           "meth"        "fqp" ];
   < alfa,sres,vres,yptr > = svm(data,modl,optn,class,ctun);
```

We can verify the result by running a single optimization specifying the best parameters.

Now we specify random k fold cross validation. Now, only k * 65 optimizations are performed:

```
  optn = [ "print"           3 ,
           "popt"            1 ,
           "tun"        "grid" ,
           "random"         10 ,
           "kern"       "rbf2" ,
           "meth"        "fqp" ];
  < alfa,sres,vres,yptr > = svm(data,modl,optn,class,ctun);
```

The following input specifies bounds for the hyper parameters which can be used for the pattern and optimization search:

```
  /* tuning by automatic NMS: specify bounds */
  ctun = [ "C"    1.e-6   100. ,
           "K1"   1.e-6     .5 ];
```

The following input specifies pattern search:

```
  optn = [ "print"           3 ,
           "popt"            1 ,
```

```
          "tun"        "patt" ,
          "split"         10 ,
          "kern"       "rbf2" ,
          "meth"        "fqp" ];
  < alfa,sres,vres,yptr > = svm(data,modl,optn,class,ctun);
```

The following input specifies the fast "PSVM" method for the time consuming Nelder-Mead bounded tuning:

```
  optn = [ "print"           3 ,
           "popt"            1 ,
           "tun"        "opti" ,
           "split"          10 ,
           "kern"       "rbf2" ,
           "meth"       "psvm" ];
  < alfa,sres,vres,yptr > = svm(data,modl,optn,class,ctun);
```

For more examples see tsvm11.inp.

5. SVM Classification: The case $N > n$

   Two of the methods are specifically designed for the $N > n$ case, the "FSM" and the Newton "NSVM" method by Mangasarian and Fung. Note, that specifically "FSM" method is designed to find sparse parameter vectors for the linear case which can be used for variable selection. "FSM" can be computed also for nonlinear kernel, but does not make much sense since α will be dense most of the time.

   We illustrate the aproach with two aplications:

   (a) NIR Spectra data set: train: Ntrain=21, Ntest: nr=7 and $n = 268$ (use median of y for binary target)

   (b) Caco2 data set by Bennett and Bi: with $N = 27$, $n = 714$ (use median of y for binary target)

   The NIR spectra data set contains a training and a test set. We use the test set with "tunsel" option to find the best C value evaluated at the test data set.

```
  print "NIR Spectra data set: train: nr=21, test: nr=7";
  options NOECHO;
#include "..\\tdata\\nir.dat"
  options ECHO;
  nr = nrow(xtrn); nc = ncol(xtrn);
  print "nrtrn,nctrn=",nr,nc;
```

After reading in the data, we compute some of the moments, including the median and dichotomize the response variable:

```
sopt = [ "ari" "std" "med" ];
mom = univar(ytrn',sopt);
/* print "Moments of y=", mom; */
cutof = mom[3];
y = ytrn .< cutof; cdat = xtrn -> y'; /* attrib(cdat); */
y = ytst .< cutof; test = xtst -> y'; /* attrib(test); */
/* print cdat[ , 200:269 ]; */
/* model: */
xind = [ 1:268 ]; yind = 269;
```

This is a normal run on the training data set and computes the solutions for all C values without using the test data set:

```
modl = "269 = 1 : 268";
class = 269;
optn = [ "print"           2 ,
         "pplan"           1 ,
         "meth"        "fsm" ,
         "kern"       "line" ];
cupr = [ "C"    .001 .001 .01 .1 1. 10. ];
alfa = svm(cdat,modl,optn,class,cupr);
```

To evaluate the fit on test data set we have to use the "tunsel" option. Otheriwse, the solution for best C value would be decided on the traning set goodness-of-fit. (Only, when we specify the "tun" option, the default of "tunsel" would use the test data set.)

```
ctun = [ "C"  .01 .1 1. 10. 100. ];
optn = [ "print"           2 ,
         "pplan"           2 ,
         "meth"        "fsm" ,
         "tunsel"     "test" ,
         "kern"       "line" ];
alfa = svm(cdat,modl,optn,class,ctun,.,.,test);
```

As a second example we use the data set Caco2 from Kristin Bennett's and Jinbo Bi's ([2]) website. It has 27 observations and 714 variables. We read two files: caco_nam.dat and caco.dat with the 714 variable names and all the data.

```
/* Caco: nobs= 27, nvar= 714 (Bi & Bennett) */
fo10 = " %16s %16s %16s %16s %16s %16s %16s %16s %16s %16s";
form = fo10;
```

```
for (j = 2; j <= 71; j++) form = strcat(form,fo10);
fo4 = " %16s %16s %16s %16s";
form = strcat(form,fo4);

fid = fopen("..\\tdata\\caco_nam.txt","r");
/* sscanf() is faster when nr and nc are specified: */
c_nam = fscanf(fid,form,.,1);
/* print c_nam; */


/*--- read data: 715 cols, first column is obs number ---*/
fid = fopen("..\\tdata\\caco.dat","r");
form = fo1 = " %g %g %g %g %g %g %g %g %g %g";
for (j = 2; j <= 71; j++) form = strcat(form,fo1);
fo5 = " %g %g %g %g %g";
form = strcat(form,fo5);
c_dat = fscanf(fid,form,27,.);
nr = nrow(c_dat); nc = ncol(c_dat);
print "Observations of c_dat.dat:",nr;
print "Columns of c_dat.dat:",nc;
cdat = c_dat[,2:715]; /* cut out col 1 */
```

Using the median of the response as cutoff value we create a binary response appropriate for our classification problem:

```
sopt = [ "ari" "std" "med" ];
mom = univar(cdat,sopt);
/* print mom; */
cutof = mom[3,714];
y = cdat[,714]; y = y .< cutof; cdat[,714] = y;
```

We now have 14 observation with zero and 13 observations with unity response. By specifying delt=2 we perform an inner grid search of the method specific parameter δ for the points $\delta = .001, .01, 1., 10., 100$. Unfortunately this is saver for obtaining a convergence region. This executes the algorithm without line search:

```
/* FSM: without line search */
modl = "714 = 1 : 713";
class = 714;
optn = [ "print"          2 ,
         "pplan"          1 ,
         "meth"       "fsm" ,
         "delt"           2 ,
         "nolin"            ,
```

```
          "kern"       "line" ];
  alfa = svm(cdat,modl,optn,class);
```

Without that option we execute the algorithm with Armijo line search:

```
  /* FSM: with line search */
  modl = "714 = 1 : 713";
  class = 714;
  optn = [ "print"           2 ,
           "pplan"           1 ,
           "meth"        "fsm" ,
           "delt"            2 ,
           "kern"       "line" ];
  alfa = svm(cdat,modl,optn,class);
```

For more examples see tsvm8.inp.

6. SVM Regression: MYSVM Data

(a) SVM Regression: FQP with QUAD

```
    #include "mysvm.dat"
      nobs = nrow(data);
      x = data[.,1:11];  y = data[,12];
      /* print nobs,y; */

      modl = "12 = 1 : 11";
      cup = [ 1. 10. 100. 1.e3 1.e4 ];
      optn = [ "print"           3,
               "popt"            1,
               "ppred"            ,
               "c"           "cup",
               "meth"        "fqp",
               "kern"       "line",
               "loss"      "quadr" ];
      alfa = svm(data,modl,optn);
```

(b) SVM Regression: DQP with QUAD

```
    #include "mysvm.dat"
      modl = "12 = 1 : 11";
      cup = [ 1. 10. 100. 1.e3 1.e4 ];
      optn = [ "print"           3,
               "ppred"            ,
               "c"           "cup",
               "meth"        "dqp",
               "kern"       "line",
```

```
            "loss"      "quadr" ];
      alfa = svm(data,modl,optn);
```

7. Some Useful Modules

(a) CMAT Formulation of LSVM Algorithm: Lee and Mangasarian

```
data = rspfile("heart_scale.dat");
x = data[,2:14]; y = data[,1];
m = nrow(x); n = ncol(x); np = n+1;
nu = 1.; rnu = 1. / nu; alf = 1.9 / nu;
D = diag(y);
A = x -> cons(m,1,-1.);
H = D * A; HH = H * H‘; /* HH is a BIG m by m matrix */
Q = H‘ * H + cons(np,np,rnu,’d’);
/* print "Starting Q=", Q; */
S = H * inv(Q);
/* print "Starting S=", S; */
u = nu * (1. - S * H‘ * cons(m,1,1.));
/* print "Starting u:", u; */
ou = u + 1.;
it = 0; itmax = 50; tol = 1.e-6;
while (it < itmax) {
  res = norm(ou - u,"inf");
  print "Iteration", it, " Residual=", res;
  if (res < tol) break;
  w = (rnu - alf) * u + HH * u - 1.;
  w = .5 * (abs(w) + w);
  z = 1. + w; ou = u;
  u = nu * (z - S * H‘ * z);
  it++;
}
w = x‘ * D * u; v = D * u; gamma = -v[+];
print "LAGR: gamma=", gamma;
print "LAGR solution w=", w;
```

(b) CMAT Formulation of PSVM Algorithm: Fung and Mangasarian

```
function psvm(A,D,nu) {
  m = nrow(A); n = ncol(A); np = n + 1;
  e = cons(m,1,-1.);
  H = D * [ A  e ]; r = H[+,]‘;
  rnu = 1. / nu;
  Q = H‘ * H + cons(np,np,rnu,’d’);
  x = Q \ r;
  u = nu * (1. - H * x); s = D * u;
  w = A‘ * s; gamma = -s[+];
  return(w,gamma);
```

```
}

data = rspfile("heart_scale.dat");
modl = "1 = 2 : 14";
class = 1;

x = data[,2:14]; y = data[,1];
d = diag(y); nu = 1.;
< w,gamma > = psvm(x,d,nu);
print "PSVM Results: w=", w;
```

4.3 Extensions of nlkpca() Function

Some specifications of tuning parameters were removed from the optn argument to a newly designed ctun input object. The new input argument ctun is $K \times k$ data object where $K \geq 1$ and $k \geq 1$ are valid choices:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of K1,
- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function
     "K4" fourth parameter of kernel function
  2. remaining columns $k = 2, ...$:
     - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)
     - for "tun" = "patt" or "tun" = "opti": ctun must have only two columns specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

The following options are no longer permitted:

Option Name	Sec. Col.	Meaning
"kfp1"	real	first parameter of kernel function
"kfp2"	real	second parameter of kernel function
"kfp3"	real	third parameter of kernel function

The following is the new doc of function nlkpca():

< gof,pc,eval > = nlkpca(x,optn<,class<,ctun<,kfun>>)

Purpose: The nlkpca function implements a (nonlinear) Hilbert kernel method for principal components analysis.

Input: **x** the $N \times M$ input data set $\mathbf{X}$ for which the $N \times p$ matrix $\mathbf{P}$ of principal components is computed. The number of components (factors) p is either user specified or the rank of $\mathbf{X}$.

optn The option argument is specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. Some of the options are similar to those of the svm() and nlkpca() functions. See below for details.

class : This optional argument should be an integer scalar or vector of integer scalars naming the number of columns which are considered categorical (nominal scaled) variables.

ctun This optional argument specifies a vector or matrix of grid points or ranges for the tuning of kernel function hyper parameters.

kfun This optional argument specifies the module name for a user specified kernel function. Note, the kernel matrix should satisfy the Mercer property of positive definiteness.

Tuning Matrix Argument: Argument ctun is $K \times k$ data object where $K \geq 1$ and $k \geq 1$ are valid choices:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of K1,
- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function
     "K4" fourth parameter of kernel function
  2. remaining columns $k = 2, ...$:
     - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)
     - for "tun" = "patt" or "tun" = "opti": ctun must have only two columns specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

Options Matrix Argument: The option argument is specified in form of a two column matrix:

Option Name	Second Column	Meaning
"fact"	int	number p of factors (components)
"kern"		link function, default is linear
	"line"	linear function
	"poly"	polynomial function
	"rbf"	Gaussian radial basis function (def. scaling $[0, 1]$)
	"rbf2"	mod. Gaussian radial basis function (def. scaling $[0, 1]$)
	"erbf"	Exponential radial basis function
	"tanh"	sigmoid function (same as "sigm")
	"sigm"	sigmoid function (same as "tanh")
	"four"	Fourier function (def. scaling $[-\pi/2, \pi/2]$)
	"spli"	spline function (def. scaling $[0, 1]$)
	"anov"	anova function
	"curv"	curvspline function (def. scaling $[0, 1]$)
	"bspl"	Bspline function (def. scaling $[0, 1]$)
	"anob"	anova spline function (def. scaling $[0, 1]$)
	"ano1"	modified anova function (def. scaling $[0, 1]$)
	"ano2"	modified anova function (def. scaling $[0, 1]$)
	"ano3"	modified anova function (def. scaling $[0, 1]$)
"nopr"		no printed output in ...txt (but warnings or errors are in ...log)
"pall"		is equivalent to "print" 10
"pmat"		print kernel matrix (this can be big)
"print"	int	amount of general printed output (default is 3)
"pshort"		less than default output
"psum"		much less than default output
"scale"	int	if not zero, perform scaling of data; this is default for several kernel functions
"tun"	"gsid"	perform grid search for tuning
	"patt"	perform pattern search for tuning
	"opti"	perform Melder-Mead optimization for tuning

Some of the kernel functions, especially those based on the exp function, like all rbf functions, can create kernel matrices which are diagonal or even identity matrices for extrem parameter settings. Using the identity matrix for the kernel matrix produces useless

results.

Output: **gof** vector of some goodness-of-fit measures.

pc $N \times p$ matrix $\mathbf{P}$ of principal components

eval matrix with eigenvalues of the centered kernel matrix; additional columns contain differences, proportions and cumulated values.

Restrictions: 1. The input data set should not contain missing values, string or complex data.

Relationships: nlkpls(), eig(), svm()

Examples: 1. Australian Institute of Sport (ais) data set (Weisberg, 2002):

```
options NOECHO;
ais = [
%inc "..\\tdata\\ais.dat";
];
options ECHO;
cname = [ "Sex" "Ht" "Wt" "LBM" "RCC" "WCC" "Hc" "Hg" ];
ais = shape(ais,.,8);
nobs = nrow(ais); nvar = ncol(ais);
ais = ais -> log(ais[,5]);
```

Using the "nlkpca" method with linear kernel:

```
optn = [ "kern"    "line" ,
         "fact"         6 ,
         "print"        2 ];
< gof,pc,eval > = nlkpca(ais,optn);
print "gof=" , gof;
```

```
                        *****************
                        Model Information
                        *****************

                  Number Valid Observations  202
                  Number Response Variables    0
                  N Independend Variables      9
                  Kernel Function         Linear
                  Use Unscaled Data

          Memory needed for Kernel matrix: 0.156425 Mb
```

```
              Eigenvalues of Centered Kernel Matrix
              ************************************

          N  Eigenvalue  Difference  Proportion  Cumulative
          1  85027.4890  79274.2033  0.88406286  0.88406286
          2  5753.28568  2136.94956  0.05981908  0.94388194
          3  3616.33611  2445.45001  0.03760041  0.98148235
          4  1170.88610  605.476953  0.01217414  0.99365649
          5  565.409146  536.733574  0.00587877  0.99953526
          6  28.6755719  18.7134747  2.982e-004  0.99983342


   Kernel PCA Number of Factors . . . . . . . . . . . . . . . . . 6
   Variance Explained . . . . . . . . . . . . . . . . . . 0.999833
   Remaining Variance . . . . . . . . . . . . . . . . . 0.000166585
   Kernel Function. . . . . . . . . . . . . . . . . . . . . Linear
   Total Number of Kernel Calls . . . . . . . . . . . . . . 20908
   Total Processing Time. . . . . . . . . . . . . . . . . . . . 0


               Total Number of Kernel Calls: 20908
                    Time for Optimization: 0
                    Total Processing Time: 0
```

The following is the output of the gof return argument:

```
gof=
              |          1
--------------------------
   VarExplain |       1.00
    RemainVar |   1.7e-004
  KernelCalls |   20908.00
    TotalTime |       0.00
   TuningTime |       0.00
```

The following small examples with the RBF2 kernel show that the spectrum of eigenvalues of the kernel matrix become closer to one for increasing values of the parameter kfp1. That means, that more variance is explained by smaller values of the kernel function parameter.

```
    optn = [ "kern"    "rbf2" ,
             "kfp1"      .001 ,
             "fact"         6 ,
             "print"        2 ];
    < gof,pc,eval > = nlkpca(ais,optn);
```

```
print "gof=" , gof;
```

Eigenvalues of Centered Kernel Matrix

N	Eigenvalue	Difference	Proportion	Cumulative
1	0.56787328	0.49693513	0.74938665	0.74938665
2	0.07093815	0.01246475	0.09361261	0.84299925
3	0.05847340	0.01981441	0.07716367	0.92016292
4	0.03865899	0.02730555	0.05101584	0.97117876
5	0.01135344	0.00486527	0.01498241	0.98616117
6	0.00648817	0.00472845	0.00856203	0.99472320

Kernel PCA Number of Factors	6
Variance Explained	0.994723
Remaining Variance	0.0052768
Kernel Function.	RBF2
Kernel Parameter [1]	0.001
Total Number of Kernel Calls	20908
Total Processing Time.	1

Now, for kfp1=.01 the first 6 eigenvalues explain less of the variance of the kernel matrix:

Eigenvalues of Centered Kernel Matrix

N	Eigenvalue	Difference	Proportion	Cumulative
1	48.7147731	35.4515394	0.31269864	0.31269864
2	13.2632338	1.39564658	0.08513629	0.39783494
3	11.8675872	2.38925071	0.07617768	0.47401262
4	9.47833648	1.69308999	0.06084115	0.53485377
5	7.78524649	0.54806431	0.04997326	0.58482703
6	7.23718218	0.16933449	0.04645525	0.63128228

Kernel PCA Number of Factors	6
Variance Explained	0.631282
Remaining Variance	0.368718
Kernel Function.	RBF2
Kernel Parameter [1]	1
Total Number of Kernel Calls	20908
Total Processing Time.	1

Now, for kfp1=.1 the first 6 eigenvalues explain less of the variance of the kernel matrix:

Eigenvalues of Centered Kernel Matrix

N	Eigenvalue	Difference	Proportion	Cumulative
1	38.4773225	33.9002668	0.67433083	0.67433083
2	4.57705565	0.43882766	0.08021477	0.75454561
3	4.13822799	1.46787901	0.07252414	0.82706975
4	2.67034898	0.61003673	0.04679896	0.87386870
5	2.06031225	0.77429050	0.03610782	0.90997652
6	1.28602175	0.51708816	0.02253806	0.93251458

Kernel PCA Number of Factors	6
Variance Explained	0.932515
Remaining Variance	0.0674854
Kernel Function.	RBF2
Kernel Parameter [1]	0.1
Total Number of Kernel Calls	20908
Total Processing Time.	0

Now, for kfp1=1. the first 6 eigenvalues explain less of the variance of the kernel matrix:

Eigenvalues of Centered Kernel Matrix

N	Eigenvalue	Difference	Proportion	Cumulative
1	48.7147731	35.4515394	0.31269864	0.31269864
2	13.2632338	1.39564658	0.08513629	0.39783494
3	11.8675872	2.38925071	0.07617768	0.47401262
4	9.47833648	1.69308999	0.06084115	0.53485377
5	7.78524649	0.54806431	0.04997326	0.58482703
6	7.23718218	0.16933449	0.04645525	0.63128228

Kernel PCA Number of Factors	6
Variance Explained	0.631282
Remaining Variance	0.368718
Kernel Function.	RBF2
Kernel Parameter [1]	1
Total Number of Kernel Calls	20908
Total Processing Time.	0

Now, for kfp1=10 less than 13 % of the variance of the kernel matrix are explained by the first six components.

Eigenvalues of Centered Kernel Matrix

N	Eigenvalue	Difference	Proportion	Cumulative
1	5.72097004	1.19980556	0.02888783	0.02888783
2	4.52116448	0.23974587	0.02282945	0.05171728
3	4.28141861	0.26090560	0.02161886	0.07333614
4	4.02051301	0.56434211	0.02030143	0.09363758
5	3.45617090	0.04657349	0.01745181	0.11108938
6	3.40959740	0.24042161	0.01721664	0.12830602

```
Kernel PCA Number of Factors . . . . . 6
Variance Explained . . . . . 0.128306
Remaining Variance . . . . . 0.871694
Kernel Function. . . . . RBF2
Kernel Parameter [1] . . . . . 10
Total Number of Kernel Calls . . . . . 20908
Total Processing Time. . . . . 1
```

2. NIR Spectra data set: train: nr=21:

First, read in the data:

```
  options NOECHO;
#include "..\\tdata\\nir.dat"
  options ECHO;
  nrtrn = nrow(xtrn); nctrn = ncol(xtrn);
  print "nrtrn,nctrn=",nrtrn,nctrn;
  xytrn = xtrn -> ytrn‘; /* attrib(xytrn); */
```

Linear kernel:

```
    optn = [ "kern"    "line" ,
             "fact"         3 ,
             "pall"           ];
    < gof,pc,eval > = nlkpca(xtrn,optn);
```

```
                    *****************
                    Model Information
                    *****************

                Number Valid Observations   21
```

```
                Number Response Variables    0
                N Independend Variables    268
                Kernel Function         Linear
                Use Unscaled Data

          Memory needed for Kernel matrix: 0.00176239 Mb


              Eigenvalues of Centered Kernel Matrix
              *************************************

            N  Eigenvalue  Difference  Proportion  Cumulative
            1  131.200580  13.4245676  0.52053381  0.52053381
            2  117.776013  115.931634  0.46727229  0.98780610
            3  1.84437903  1.28061141  0.00731751  0.99512361


We only show the first five from the 21 columns:

              Eigenvectors of Centered Kernel Matrix
              *************************************

                      Dense Matrix (3 by 21)

       |          1          2          3          4          5
     ------------------------------------------------------------
   PC1 |  -0.1450285 -0.0029272 -0.1659867  0.1284992 -0.0327733
   PC2 |   0.4004388  0.3123114  0.3172391  0.2075480  0.2023632
   PC3 |   0.5338241  0.2411152  0.0428607  0.0828897 -0.0584783


              Principal Components (Kernel  Linear)
              *************************************

                      Dense Matrix (3 by 21)

       |          1          2          3          4          5
     ------------------------------------------------------------
   PC1 |  -0.1450285 -0.0029272 -0.1659867  0.1284992 -0.0327733
   PC2 |   0.4004388  0.3123114  0.3172391  0.2075480  0.2023632
   PC3 |   0.5338241  0.2411152  0.0428607  0.0828897 -0.0584783


  Kernel PCA Number of Factors . . . . . . . . . . . . . . . . 3
  Variance Explained . . . . . . . . . . . . . . . . . . 0.995124
  Remaining Variance . . . . . . . . . . . . . . . . . . 0.00487639
  Kernel Function. . . . . . . . . . . . . . . . . . . . . Linear
  Total Number of Kernel Calls . . . . . . . . . . . . . . . 274
```

```
   Total Processing Time. . . . . . . . . . . . . . . . . . . . 0
```

We illustrate the eigenvalue distribution of the RBF kernel matrix for only two values of kfp1:

```
    optn = [ "kern"    "rbf2" ,
             "kfp1"       10. ,
             "fact"         3 ,
             "pall"           ];
    < gof,pc,eval > = nlkpca(xtrn,optn);
```

Eigenvalues of Centered Kernel Matrix

N	Eigenvalue	Difference	Proportion	Cumulative
1	1.00000001	6.661e-016	0.05000000	0.05000000
2	1.00000001	2.220e-016	0.05000000	0.10000000
3	1.00000001	2.220e-016	0.05000000	0.15000000

We only show the first five from the 21 columns:

Eigenvectors of Centered Kernel Matrix

Dense Matrix (3 by 21)

	1	2	3	4	5
PC1	0.1385307	0.1385307	0.1504903	-0.0370896	-0.0368392
PC2	-0.1829113	-0.1829113	0.1227363	0.0958478	0.1066739
PC3	0.1326678	0.1326678	-0.2367299	-0.2686845	-0.2649538

Principal Components (Kernel RBF2)

Dense Matrix (3 by 21)

	1	2	3	4	5
PC1	0.1385307	0.1385307	0.1504903	-0.0370896	-0.0368392
PC2	-0.1829113	-0.1829113	0.1227363	0.0958478	0.1066739
PC3	0.1326678	0.1326678	-0.2367299	-0.2686845	-0.2649538

```
Kernel PCA Number of Factors . . . . . 3
Variance Explained . . . . . 0.15
Remaining Variance . . . . . 0.85
Kernel Function. . . . . RBF2
Kernel Parameter [1] . . . . . 10
Total Number of Kernel Calls . . . . . 274
Total Processing Time. . . . . 0
```

```
optn = [ "kern"    "rbf2" ,
         "kfp1"     .0001 ,
         "fact"         3 ,
         "pall"           ];
< gof,pc,eval > = nlkpca(xtrn,optn);
```

Eigenvalues of Centered Kernel Matrix

****

N	Eigenvalue	Difference	Proportion	Cumulative
1	0.17068201	0.08428511	0.54004649	0.54004649
2	0.08639690	0.03368047	0.27336415	0.81341064
3	0.05271643	0.04893955	0.16679743	0.98020807

```
Kernel PCA Number of Factors . . . . . 3
Variance Explained . . . . . 0.980208
Remaining Variance . . . . . 0.0197919
Kernel Function. . . . . RBF2
Kernel Parameter [1] . . . . . 0.0001
Total Number of Kernel Calls . . . . . 274
Total Processing Time. . . . . 0
```

4.4 Extensions of nlkpls() Function

Some specifications of tuning parameters were removed from the optn argument to a newly designed ctun input object. The new input argument ctun is $K \times k$ data object where $K \geq 1$ and $k \geq 1$ are valid choices:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of K1,
- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function
     "K4" fourth parameter of kernel function
  2. remaining columns $k = 2, ...$:
     - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)
     - for "tun" = "patt" or "tun" = "opti": ctun must have only two columns specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

The following options are no longer permitted:

Option Name	Sec.	Col.	Meaning
"kfp1"		real	first parameter of kernel function
"kfp2"		real	second parameter of kernel function
"kfp3"		real	third parameter of kernel function

The following is the new doc of function nlkpls():

< alpha,sres,vres,yptrn,yptnt > = nlkpls(trn,model,optn<,class<,ctun<,kfun<,test>>>>)

Purpose: The nlkpls function implements a method for nonlinear Hilbert kernel partial least squares (PLS). In many practical aplications the RBF (radial base function kernel) has better modeling properties than the common linear function. The nlkpls function permits multiple runs for tuning kernel parameters using cross validation methods. It also permits the specification of a binary or multinomial response (target) using the class input argument. For multinomial response, the *one vs. all other* strategy is used for estimation and scoring.

Input: trn the $N \times M$ input data set containing the $N \times m$ predictor matrix $\mathbf{X}$ and the N response vector $\mathbf{Y}$. Which data columns

are selected for $\mathbf{X}$ and $\mathbf{Y}$ must be specified with the model string.

model : The analysis model is specified in form of a string, e.g. model= "3=1 2", containing column numbers for variables. The syntax of the model string argument is the same as for the glmod() function except for the additional *events / trial* response specification. ????

optn : The option argument is specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. Some of the options are similar to those of the svm() and nlkpca() functions. See table below for content.

class : This optional argument should be an integer scalar or vector of integer scalars naming the number of columns which are considered categorical (nominal scaled) variables.

ctun This optional argument specifies a vector or matrix of grid points or ranges for the tuning of kernel function hyper parameters.

kfun This optional argument specifies the module name for a user specified kernel function. Note, the kernel matrix should satisfy the Mercer property of positive definiteness.

test specifies a matrix of test data which is not used for modeling but for predicted values are computed (scored) that can be returned with the last output argument yptt.

Tuning Matrix Argument: Argument ctun is $K \times k$ data object where $K \geq 1$ and $k \geq 1$ are valid choices:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of K1,
- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function
     "K4" fourth parameter of kernel function
  2. remaining columns $k = 2, ...$:
     - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)

- for "tun" = "patt" or "tun" = "opti": ctun must have only two columns specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

Options Matrix Argument: The option argument is specified in form of a two column matrix:

Option Name	Second Column	Meaning
"best"	int	only valid with Loo estimation: lists the *best* observations which when deleted improve most the goodness-of-fit
"block"	int	fold number for block cross validation
"cloo"		compute Loo (leave-one-out) estimates
"cval"	string	method of cross validation
	"blo"	blockwise sampling (default)
	"loo"	leave-one-out
	"set"	test data set
	"spl"	split sampling
	"ran"	random sampling
"fact"	int	number p of factors (components)
"kern"		link function, default is linear
	"line"	linear function
	"poly"	polynomial function
	"rbf"	Gaussian radial basis function (def. scaling $[0,1]$)
	"rbf2"	mod. Gaussian radial basis function (def. scaling $[0,1]$)
	"erbf"	Exponential radial basis function
	"tanh"	sigmoid function (same as "sigm")
	"sigm"	sigmoid function (same as "tanh")
	"four"	Fourier function (def. scaling $[-\pi/2, \pi/2]$)
	"spli"	spline function (def. scaling $[0,1]$)
	"anov"	anova function
	"curv"	curvspline function (def. scaling $[0,1]$)
	"bspl"	Bspline function (def. scaling $[0,1]$)
	"anob"	anova spline function (def. scaling $[0,1]$)
	"ano1"	modified anova function (def. scaling $[0,1]$)
	"ano2"	modified anova function (def. scaling $[0,1]$)
	"ano3"	modified anova function (def. scaling $[0,1]$)
"meth"	"kpcr"	kernel principal components regression
	"kpls"	kernel partial least squares regression
	"dpls"	direct kernel partial least squares regression
"nobstat"		print observation statistics
"nomis"		do not use observations that contain missing values in one of the variables used in analysis
"nopr"		no printed output in ...txt (but warnings or errors are in ...log)
"outl"	int	same as "best"
"pall"		is equivalent to "print" 10
"pcrv"		print cross validation tables
"pinit"		print initial information
"pmat"		print kernel matrix (this can be big)
"ppar"		print vector of parameter estimates α
"ppred"		print predicted values

"print"	int	amount of general printed output (default is 3)
"pshort"		less than default output
"ptun"	int	amount of printed output in parameter tuning (i.e. in outer loop), default is 0
"psum"		much less than default output
"scale"	int	if not zero, perform scaling of data; this is default for several kernel functions
"split"	int	fold number for split cross validation
"tun"	"gsid"	perform grid search for tuning
	"patt"	perform pattern search for tuning
	"opti"	perform Melder-Mead optimization for tuning
"tunitr"	int	max iterations for tuning
"tuntim"	int	max time in seconds for tuning
"tunftol"	real	f termination for tuning
"tunxtol"	real	x termination for tuning
"tunsel"	"test"	for tuning use validation data evaluating goodness-of-fit (default if "tun" is specified)
	"train"	for tuning use validation data evaluating goodness-of-fit (default if "tun" is not specified)

Some of the kernel functions, especially those based on the exp function, like all rbf functions, can create kernel matrices which are diagonal or even identity matrices for extrem parameter settings. using the identity matrix for the kernel matrix produces useless results.

Output: **alpha** vector or matrix with parameter estimates

sres a vector of scalar values

vres for multinomial response the sres contains in rows information for each response level vs. all other

yptrn vector of predicted values for training data

yptst vector os predicted values of test data

Restrictions: 1. The input data set should not contain missing values, string or complex data.

Relationships: nlkpca(), pls(), svm()

Examples: 1. Australian Institute of Sport (ais) data set (Weisberg, 2002):

```
  options NOECHO;
  ais = [
  %inc "..\\tdata\\ais.dat";
  ];
  options ECHO;
  cname = [ "Sex" "Ht" "Wt" "LBM" "RCC" "WCC" "Hc" "Hg" ];
  ais = shape(ais,.,8);
  nobs = nrow(ais); nvar = ncol(ais);
  ais = ais -> log(ais[,5]);
```

Using the "nlkpls" method with linear kernel:

```
  modl = "4 = 2 3 9 6";
  optn = [ "slic"         8 ,
           "dec"      "svd" ,
           "numdir"       4 ,
           "print"        3 ,
           "nperm"      499 ,
           "seed"       123 ];
  < gof,parm,atst,ptst > = nlkpls(ais,modl,optn);
```

Fact	SSX	Xdif	Ydif
0	2789392633	.	.
1	30949465.7	2758443167	192.058321
2	426646.390	30522819.3	0.03416301
3	2.72599730	426643.664	0.01018565

```
   Kernel PLS Number of Factors . . . . . . . . . . . . . . . . . 4
   Mean Squared Error (MSE, Training Data). . . . . . . . . 13.9221
   Mean Absolute Error (MAE, Training Data) . . . . . . .  2.9882
   Average Positive Loss (Training Data). . . . .  3.42964 (N=88)
   Average Negative Loss (Training Data). . . . . 2.64744 (N=114)
   Kernel Function. . . . . . . . . . . . . . . . . . . . . Linear
   Total Number of Kernel Calls . . . . . . . . . . . . . . 20908
   Total Processing Time. . . . . . . . . . . . . . . . . . . . 1
   Linear Kernel Constant (PCE) . . . . . . . . . . . . . 64.8737
```

```
    y = ais[,yind]; nr = nrow(ais);
    res = y - yptr';
    s1 = ssq(res); s2 = s1 / nr;
    print "sse, mse=", s1, s2;
```

```
sse, mse= 2812.27 13.922
```

This is the traditional PLS call:

```
optn = [ "prin"         1 ,
         "meth"     "pls" ,
         "tech"  "krnpls" ];
yind = 4; xind = [ 2 3 9 6 ];
cmp  = [ 1:4 ];
< rms,bmat,yprd > = pls(ais,yind,xind,cmp,optn);
```

```
                -------------------------
                Analysis of Training Data
                -------------------------

                 Summary Table: Training
                 -----------------------

   Factor    X_VarExp    Y_VarExp         RMS          R2
        1  0.89501676  0.87706842  4.57126303  0.87706842
        2  0.98884248  0.88123227  4.49317899  0.88123227
        3  0.99997186  0.88243891  4.47029602  0.88243891
        4  1.00000000  0.91809760  3.73123672  0.91809760
```

```
y = ais[,yind]; nr = nrow(ais);
res = y - yprd[,4];
s1 = ssq(res); s2 = s1 / nr;
print "sse, mse=", s1, s2;
```

As you can see we obtain the same result, however, much faster with the specific pls call than with the more general nlkpls call:

```
sse, mse= 2812.27 13.922
```

Now, use the RBF kernel:

```
optn = [ "meth"    "kpls" ,
         "kern"    "rbf2" ,
         "fact"         4 ,
         "ppred"          ,
         "pall"           ];
modl = "4 = 2 3 9 6";
tun = [ .01 .1 1. 10. ];
< alpha,sres,vres,yptr > = nlkpls(ais,modl,optn,.,tun);
```

```
                Best 4 Results of Grid Search
                 ----------------------------

              N        KF1      L1err       Error    TrainErr
              4  10.000000 7.67859361  1551.07591  7.67859361
              2  0.1000000 12.7083058  2567.07776  12.7083058
              1  0.0100000 13.6464396  2756.58079  13.6464396
              3  1.0000000 13.9173195  2811.29854  13.9173195


               Fact         SSX        Xdif        Ydif
                  0  763.949504           .           .
                  1  627.177013  136.772490  185.013336
                  2  577.933090  49.2439234  0.67091219
                  3  522.161312  55.7717779  0.58694927


    Kernel PLS Number of Factors . . . . . . . . . . . . . . . . . 4
    Mean Squared Error (MSE, Training Data). . . . . . . . . 7.67859
    Mean Absolute Error (MAE, Training Data) . . . . . . . . 2.10641
    Average Positive Loss (Training Data). . . . .  2.17089 (N=98)
    Average Negative Loss (Training Data). . . . . 2.04565 (N=104)
    Kernel Function. . . . . . . . . . . . . . . . . . . . . . RBF2
    Kernel Parameter [1] . . . . . . . . . . . . . . . . . . . . 10
    Total Number of Kernel Calls . . . . . . . . . . . . . .  102921
    Total Processing Time. . . . . . . . . . . . . . . . . . . . 3
    Linear Kernel Constant (PCE) . . . . . . . . . . . . . . 64.8737


     y = ais[,yind]; nr = nrow(ais);
     res = y - yptr`;
     s1 = ssq(res); s2 = s1 / nr;
     print "sse, mse=", s1, s2;


   sse, mse= 1551.08 7.6786
```

2. Boston Housing: nobs=506, nvar=14:
   First, read in the data:

```
  fid = fopen("..\\tdata\\housing.dat","r");
  form = "%g %g %g %g %g %g %g %g %g %g %g %g %g %g";
  hous = fscanf(fid,form,506,14);
  vnam = [ "crim" "zn" "indus" "chas" "nox" "rm" "age"
           "dis" "rad" "tax" "ptrat" "b" "lstat" "medv" ];
  hous = cname(hous,vnam);
  nr = nrow(hous);
```

```
/* print hous[1:10,; */


 optn = [ "meth"    "kpls" ,
          "kern"    "line" ,
          "fact"       10 ,
          "ppred"         ,
          "pall"         ];
 modl = "14 = 1 : 13";
 < alpha,sres,vres,yptr > = nlkpls(hous,modl,optn);


          Fact         SSX        Xdif        Ydif
              0  2.535e+014           .           .
              1  1.023e+013  2.433e+014  466.876984
              2  7.942e+011  9.437e+012  0.03616631
              3  2.056e+010  7.736e+011  0.07118562
              4  8591407737  1.197e+010  0.30250690
              5   611679109  7979728627  0.19382942
              6   107150897   504528213  0.06141443
              7  64976679.5  42174217.1  0.11101507
              8  28871952.7  36104726.9  0.05344789
              9  918237.801  27953714.9  0.02281388


 Kernel PLS Number of Factors . . . . . .  10
 Mean Squared Error (MSE, Training Data). . . . . 24.4258
 Mean Absolute Error (MAE, Training Data) . . . . . 3.50328
 Average Positive Loss (Training Data). . . . . 2.91556 (N=304)
 Average Negative Loss (Training Data). . . . . 4.38777 (N=202)
 Kernel Function. . . . . .  Linear
 Total Number of Kernel Calls . . . . . .  129284
 Total Processing Time. . . . . .  33
 Linear Kernel Constant (PCE) . . . . . 22.5328


 y = hous[,yind]; nr = nrow(hous);
 res = y - yptr`;
 s1 = ssq(res); s2 = s1 / nr;
 print "sse, mse=", s1, s2;


sse, mse= 12359.45 24.426
```

Compared to that we run the traditional PLS algorithm:

```
                  ****************************
                  Kernel Partial Least Squares
                  ****************************

    X Dimension . . . . . .       13
    Y Dimension . . . . . .        1
    Nobs Training Data. . . . . .      506
    Number Analyses . . . . . .       10
    Maximum Number Factors. . . . . .       10


                   -------------------------
                   Analysis of Training Data
                   -------------------------

                    Summary Table: Training
                    -----------------------

       Factor    X_VarExp    Y_VarExp         RMS          R2
            1  0.80508818  0.24227210  7.99794054  0.24227210
            2  0.94450953  0.26942855  7.85331254  0.26942855
            3  0.98972248  0.32050921  7.57379108  0.32050921
            4  0.99344496  0.51051474  6.42822904  0.51051474
            5  0.99802224  0.60079391  5.80523907  0.60079391
            6  0.99913209  0.62493450  5.62697636  0.62493450
            7  0.99940679  0.66541682  5.31463678  0.66541682
            8  0.99967974  0.68306063  5.17260871  0.68306063
            9  0.99994045  0.69025001  5.11360509  0.69025001
           10  0.99998521  0.71066198  4.94224519  0.71066198


                     Time for Training: 0
                         Total Time: 0


sse, mse= 12359.45 24.426


Now we run kpls using RBF Kernel with different values of
the bandwidth parameter:

    optn = [ "meth"    "kpls" ,
             "kern"    "rbf2" ,
             "fact"        10 ,
             "ppred"          ,
             "pall"           ];
    modl = "14 = 1 : 13";
```

```
tun = [ .01 .1 1. 10. ];
< alpha,sres,vres,yptr > = nlkpls(hous,modl,optn,.,tun);
```

Best 4 Results of Grid Search

N	KF1	L1err	Error	TrainErr
4	10.000000	0.10003232	50.6163528	0.10003232
3	1.0000000	3.59081102	1816.95038	3.59081102
2	0.1000000	8.57670852	4339.81451	8.57670852
1	0.0100000	12.6576099	6404.75059	12.6576099

The best result is obtained for band width 10:

Grid Search Results: L2 Error (Training)

Dense Row Vector (ncol=4)

R	1	2	3	4
	6404.7506	4339.8145	1816.9504	50.616353

Fact	SSX	Xdif	Ydif
0	915.229759	.	.
1	852.813326	62.4164332	477.240560
2	812.757823	40.0555034	0.84709880
3	793.254145	19.5036779	0.91879709
4	771.593443	21.6607018	0.75298552
5	752.876496	18.7169466	0.60005021
6	733.864648	19.0118480	0.35182905
7	721.197304	12.6673442	0.29731449
8	709.227252	11.9700523	0.32471183
9	702.624902	6.60234949	0.36547887

```
Kernel PLS Number of Factors . . . . . 10
Mean Squared Error (MSE, Training Data). . . . . 0.100032
Mean Absolute Error (MAE, Training Data) . . . . . 0.188909
Average Positive Loss (Training Data). . . . 0.187428 (N=255)
Average Negative Loss (Training Data). . . . 0.190415 (N=251)
Kernel Function. . . . . RBF2
Kernel Parameter [1] . . . . . 10
Total Number of Kernel Calls . . . . . 642369
Total Processing Time. . . . . 153
Linear Kernel Constant (PCE) . . . . . 22.5328
```

```
    y = hous[,yind];
    res = y - yptr`; nr = nrow(hous);
    s1 = ssq(res); s2 = s1 / nr;
    print "sse, mse=", s1, s2;


sse, mse= 50.616 0.1000
```

5 New Developments

5.1 The error() Function

```
error(message)
```

Purpose: The error() function prints an error message mess into the log output file.

Input: The input must be a string. The string may be generated by a call of function sprintf().

Output: Generates an error message in the log file.

Restrictions: 1. The type of the only argument must be string.

Relationships: warning()

Examples:
```
if (a == .) error("Variable cannot be missing.");
```

5.2 The fnampid() Function

```
fnam = fnampid(filpath<,ext>)
```

Purpose: The fnampid() function creates a string result by concatenating the specified prefix string filpath with the process ID of the current run of CMAT and eventually followed by a file extension string ext as suffix.

Input: 1. The first input must be a string specifying the prefix.
2. The second input argument is optional, specifying a suffix which could be a file extension.

Output: An integer indicating whether the specified file has been successfully removed (== 0) or not (!= 0).

Restrictions: 1. The type of each of the two input arguments must be string.

Relationships: pid()

Examples:
```
fnam = fnampid("_pcx_mps_");
id =  pid();
print "Fname with PID=", fnam, " Current Process ID=",id;
```

Fname with PID=_pcx_mps_4132 Current Process ID= 4132

5.3 The fremove() Function

```
irc = fremove(filpath)
```

Purpose: The fremove() function removes an existing file.

Input: The only input must be a string specifying the path to an existing file.

Output: An integer indicating whether the specified file has been successfully removed (== 0) or could not be removed (!= 0).

Restrictions: 1. The type of the only argument must be string.

Relationships: fopen() fclose()

Examples:

5.4 The frename() Function

```
irc = frename(oldfil,newfil)
```

Purpose: The frename() function renames an existing file oldfil.

Input: Both input arguments must be a string specifying an old and a new file name.

Output: An integer indicating whether the specified file has been successfully renamed (== 0) or could not be renamed (!= 0).

Restrictions: 1. The first argument must be the name of an existing file and the second argument must not be the name of an existing file.

Relationships: rename statement

Examples:

5.5 The `lstlabel()` Function

`b=lstlabel(a,lab<,ind>)` `lab=lstlabel(a<,ind>)`

Purpose: There are two versions of this function:

b=lstlabel(a,lab<,ind>) two or three input arguments where the second argument is either a string scalar or a vector of strings: The returned object b is identical with input a except that label strings lab are assigned to the entries of the list a. If the second argument is a missing value, any existing dimension labels of a are removed. If the last input argument ind is specified, it should be either an integer scalar or a vector of ints referring to indices of entries of the list a to which the strings of lab should belong. The length of lab and ind must be the same.

lab=lstlabel(a<,ind>) one or two input arguments where the second input argument (if specified) is either an integer scalar or vector: Returns a missing value if the list a has no entry labels assigned, otherwise it returns a string scalar or vector with labels of the entries of list a. If an integer scalar or vector ind ist specified, only the selected strings are returned, otherwise all labels of a are returned.

We refer to the first of the two forms as "Form A" and to the second as "Form B".

Input: **b=lstlabel(a,lab<,ind>)** The first argument must be a data list a. The second argument is either a string scalar or a vector of strings lab which should be used as labels of the entries of data list a. The third input argument is optional and if it is specified it must have the same size as lab and then specifies the indices of the entries of list a to which the strings of lab should belong.

lab=lstlabel(a<,ind>) The first input must be a data list a.The second input argument is optional and if it is specified it must be either an integer scalar or vector pointing to those entries of list a of which the label strings must be returned.

Output: **b=lstlabel(a,lab<,ind>)** The returned object b is identical with the input object a except that the list of strings lab is assigned to the labels of the entries of list a.

lab=lstlabel(a<,ind>) Returns a missing value if the dimensions of the list a have no labels assigned. Otherwise it returns the specified set of label strings of the object a.

Restrictions: 1. Form A: The second input argument must be a vector of strings that does not contain any numeric or missing values.

2. Form A: The number of strings in the second argument lab must match
   - the number of entries of the data list a if ind is not specified.
   - the number of indices specified with ind.
3. If ind is specified it integers must be in the range of valid indices of entries of a.

Relationships: lstname(), dimname(), rname(), cname(), rlabel(), clabel()

Examples: See the lstname function below.

5.6 The lstname() Function

b=lstname(a,nam<,ind>) | nam=lstname(a<,ind>)

Purpose: There are two versions of this function:

b=lstname(a,nam<,ind>) two or three input arguments where the second argument is either a string scalar or a vector of strings: The returned object b is identical with input a except that name strings lab are assigned to the entries of the list a. If the second argument is a missing value, any existing dimension names of a are removed. If the last input argument ind is specified, it should be either an integer scalar or a vector of ints referring to indices of entries of the list a to which the strings of nam should belong. The length of nam and ind must be the same.

nam=lstname(a<,ind>) one or two input arguments where the second input argument (if specified) is either an integer scalar or vector: Returns a missing value if the list a has no entry labels assigned, otherwise it returns a string scalar or vector with names of the entries of list a. If an integer scalar or vector ind ist specified, only the selected strings are returned, otherwise all names of a are returned.

We refer to the first of the two forms as "Form A" and to the second as "Form B".

Input: b=lstname(a,nam<,ind>) The first argument must be a data list a. The second argument is either a string scalar or a vector of strings nam which should be used as names of the entries of data list a. The third input argument is optional and

if it is specified it must have the same size as nam and then specifies the indices of the entries of list a to which the strings of nam should belong.

nam=lstname(a<,ind>) The first input must be a data list a.The second input argument is optional and if it is specified it must be either an integer scalar or vector pointing to those entries of list a of which the label strings must be returned.

Output: **b=name(a,nam<,ind>)** The returned object b is identical with the input object a except that the list of strings nam is assigned to the labels of the entries of list a.

nam=name(a<,ind>) Returns a missing value if the dimensions of the list a have no names assigned. Otherwise it returns the specified set of name strings of the object a.

Restrictions: 1. Form A: The second input argument must be a vector of strings that does not contain any numeric or missing values.

2. Form A: The number of strings in the second argument nam must match
   - the number of entries of the data list a if ind is not specified.
   - the number of indices specified with ind.
3. If ind is specified it integers must be in the range of valid indices of entries of a.

Relationships: lstlabel(), dimlabel(), rname(), cname(), clabel(), rlabel()

Examples: This easy examples illustrates how to attach a set of names it to the entries of a list, and how to move it from the list back into a set of strings.

```
print "Create a data list with three entries";
list ilst[3];
ilst[1] = [ "time1" "time2" ];
ilst[2] = [ "row1" "row2" "row3" ];
ilst[3] = [ "col1" "col2" "col3" "col4" ];
print "List ilst=",ilst;
```

```
List ilst=
                        *************************
                        ilst (List with 3 Entries)
                        *************************

                              ilst[1]:
```

```
                    *********

                Dense Row Vector (ncol=2)

                   R |        1       2
                          time1   time2

                         ilst[2]:
                         *********

                Dense Row Vector (ncol=3)

                    R |       1      2      3
                          row1   row2   row3

                         ilst[3]:
                         *********

                Dense Row Vector (ncol=4)

              R |       1      2      3      4
                     col1   col2   col3   col4
```

```
nams = [" time  rows  cols "];
alst = lstname(ilst,nams);
print "List alst=", alst;
```

```
                *************************
                alst (List with 3 Entries)
                *************************

                      alst[1]: time
                      *************

                Dense Row Vector (ncol=2)

                   R |        1       2
                          time1   time2

                      alst[2]: rows
                      *************

                Dense Row Vector (ncol=3)
```

```
                    R |       1      2      3
                           row1   row2   row3

                         alst[3]: cols
                         *************

                    Dense Row Vector (ncol=4)

                  R |       1      2      3      4
                         col1   col2   col3   col4
```

```
  anam = lstname(alst);
  print "Return all entry names of List=", anam;
```

```
Return all entry names of List=
  COL |      1
--------------
   1  |   time
   2  |   rows
   3  |   cols
```

```
  nams = "Two_times";
  blst = lstname(ilst,nams,1);
  print "Set name of first entry: Blst=", blst;
```

```
Set name of first entry: Blst=

                    *************************
                    blst (List with 3 Entries)
                    *************************

                        blst[1]: Two_times
                        ******************

                    Dense Row Vector (ncol=2)

                       R |        1       2
                              time1   time2

                            blst[2]:
                            *********
```

```
                    Dense Row Vector (ncol=3)

                     R |       1      2      3
                            row1   row2   row3

                            blst[3]:
                            *********

                    Dense Row Vector (ncol=4)

                   R |       1      2      3      4
                          col1   col2   col3   col4


  clst = lstname(alst,nams,1);
  print "Reset name of first entry: Clst=", clst;


Reset name of first entry: Clst=

                    *************************
                    clst (List with 3 Entries)
                    *************************

                        clst[1]: Two_times
                        ******************

                    Dense Row Vector (ncol=2)

                       R |        1       2
                              time1   time2

                           clst[2]: rows
                           *************

                    Dense Row Vector (ncol=3)

                     R |       1      2      3
                            row1   row2   row3

                           clst[3]: cols
                           *************

                    Dense Row Vector (ncol=4)
```

```
                R |       1      2      3      4
                      col1   col2   col3   col4
```

```
print "Only the name of the first entry is set";
nam1 = lstname(blst,1);
print "First entry name of blst=", nam1;
nam2 = lstname(blst,2);
print "Second entry name of blst=", nam2;
nam3 = lstname(blst,3);
print "Third entry name of blst=", nam3;
```

```
Only the name of the first entry is set

First entry name of blst=Two_times

Second entry name of blst= .

Third entry name of blst= .
```

```
print "All list names are set";
nam1 = lstname(clst,1);
print "First entry name of clst=", nam1;
nam2 = lstname(clst,2);
print "Second entry name of clst=", nam2;
nam3 = lstname(clst,3);
print "Third entry name of clst=", nam3;
```

```
All list names are set

First entry name of clst=Two_times

Second entry name of clst=rows

Third entry name of clst=cols
```

5.7 The outlier() Function

```
<oind,oval,crit> = outlier(data,type,<,optn>)
```

Purpose: The function outlier() implements the following methods for detecting outliers in onedimensional data:

- χ^2 method (see Dixon, 1950), this is also in R
- 3-Sigma rule, 3-MAD rule
- Tukey (1977b) method for testing the interquartile range
- Chauvenet (1863) Criterion
- Grubbs (1969) test, this is also in R (similar to Thompson)
- Thompson (1985) τ test (similar to Grubbs)
- Dixon Q test (Dixon, 1950; Rohrabacher, 1991; McBane, 2006), this is similar of that in R

Dixon's test is also with the outlier() package of R, however, using interpolated table values by Rohrabacher (1991) instead of the numerically more exact computated by McBane (2006). Note, each of those methods determine only whether the smallest or largest value of a set of univariate data is an outlier.

Only a slight modification of Dixon's test could be used for testing data values which are not the largest or smallest, see McBane (2006) for different thresholds. Some easy CMAT code could be used together with some more specific call of the dixonr() function.

The χ^2, Chauvenet, Grubbs, and Thompson methods test whether a mean-variance standardized value exceeds some threshold. Tukey test is slightly more robust in using median and quartiles.

Input: **data** must be a either a n-vector or $n \times m$ matrix of real or integer data. If it is a $n \times m$ matrix, the matrix is processed columnwise, i.e. $n > 3$ is assumed.

type is either a string scalar or a K vector of strings specifying the method(s):

"chisq" χ^2 method
"sig_3" 3-Sigma rule
"mad_3" 3-MAD rule
"tukey" Tukey method
"chauv" Chauvenet method
"grubb" Grubbs (1969) test
"thomp" Thompson τ test
"dixon" Dixon Q test

optn Specifying a vector of options:

1. amount of printed output, integer >= 0, default is 0 for no printed output

2. the maximum number $drpmax$ of outliers to drop; note, there maybe less than that
3. =0: default (look for lower and upper) =1: return only lower outliers =2: return only upper outliers
4. probability $alpha$ only valid for χ^2 method Chauvenet criterion, Grubbs test, Thompson test and Dixon test, default is $\alpha = 0.05$
5. parameter ***noint*** for specifying the variance divisor (noint=0: vardiv=$n-1$, noint=1: vardiv=n), default is noint=0

Some explanation is needed for optn[3]: When only lower (or upper) outliers must be returned, that does not mean that the upper (resp. lower) outliers are not computed and removed from the data. They are just not be shown in the return arguments as outliers.

Output: Depending on the input parameters, the three output objects can be vectors, matrices, or a list of matrices. Here we only refer to the size of oind, since oval has the same size as oind, and crit has just twice the number of rows and columns. The following four different cases determine the form of the output objects:

1. $m = 1$: if the input argument data is a vector the return objects oind is either a vector or matrix of size $K \times drpmax$.
2. $m > 1 \quad \& \quad K = 1$: oind is matrix of size $m \times drpmax$.
3. $m > 1 \quad \& \quad drpmax = 1$: oind is matrix of size $m \times K$.
4. $m > 1 \quad \& \quad K > 1 \quad \& \quad drpmax > 1$: oind is data list containing m matrices of size $K \times drpmax$.

oind an object containing the integer indices of the detected outliers in the rows of data (depending on the runtime option INDBASE that index could be 0 or 1 based). Missing values indicate there are no outliers.

oval an object containing the real values of the detected outliers in the rows of the data. Missing values indicate there are no outliers.

crit an object containing the test values and the thresholds, i.e. the criteria for deciding whether the smallest and largest entry is an outlier. See table below for some more details.

The meaning of the entries in crit are (μ is mean, σ standard deviation, $q_1, q_2 = median, q_3$ quartiles):

Method		Meaning
χ^2 method	TST_LOW	prob of χ^2 CDF for $(\mu - d_1)^2/\sigma^2$
	TST_UPP	prob of χ^2 CDF for $(d_n - \mu)^2/\sigma^2$
	THR_LOW	α
	THR_UPP	α
3-Sigma rule	TST_LOW	d_1
	TST_UPP	d_n
	THR_LOW	$(\mu - 1.5\sigma)$
	THR_UPP	$(\mu + 1.5\sigma)$
3-MAD rule	TST_LOW	d_1
	TST_UPP	d_n
	THR_LOW	$q_2 - 1.5MAD$
	THR_UPP	$q_2 + 1.5MAD$
Tukey method	TST_LOW	d_1
	TST_UPP	d_n
	THR_LOW	$q_2 - 1.5(q_3 - q_1)$
	THR_UPP	$q_2 + 1.5(q_3 - q_1)$
Chauvenet	TST_LOW	$2 * n * (1 - prob)$, prob is from $(0, 1)$ normal CDF for $(\mu - d_1)/\sigma$
	TST_UPP	$2 * n * (1 - prob)$, prob is from $(0, 1)$ normal CDF for $(d_n - \mu)/\sigma$
	THR_LOW	0.5
	THR_UPP	0.5
Grubbs test	TST_LOW	$(\mu - d_1)/\sigma$
	TST_UPP	$(d_n - \mu)/\sigma$
	THR_LOW	Grubbs test criterion
	THR_UPP	Grubbs test criterion
Thompson test	TST_LOW	$(\mu - d_1)/\sigma$
	TST_UPP	$(d_n - \mu)/\sigma$
	THR_LOW	Thompson τ test criterion
	THR_UPP	Thompson τ test criterion
Dixon test	TST_LOW	$(d_2 - d_1)/(d_n - d_1)$
	TST_UPP	$(d_n - d_{n-1})/(d_n - d_1)$
	THR_LOW	Dixon Q test criterion
	THR_UPP	Dixon Q test criterion

Restrictions:

- For all methods the number of rows n must be at least 3. Since Tukey (more robust) method is based on median and the interquartiles, it makes sense that it requires $n \geq 5$.
- The first input argument data must not have any missing, string, or complex values.

Relationships: outlmd(), dixonr()

Examples: 1. Example from the internet (http://www.stat.wmich.edu/s160/book/node8.html):

```
data = [ 126  132  138  140  141  141  142  143  144  144  144
         145  146  147  148  148  149  149  150  150  150  154
         155  158  158  ];
```

```
  type = [" chisq sig_3 mad_3 tukey chauv grubb thomp dixon "];


  optn = [ 2 ,   /* print */
           3 ];  /* drpmx */
  < oind,oval,crit > = outlier(data,type,optn);
  print "OIND=", oind;
  print "OVAL=", oval;
  print "CRIT=", crit;
```

For $m = 1$ the results are $k \times drpmax$ matrices:

```
OIND=
           |     DRP_1     DRP_2     DRP_3
-----------------------------------------
  CHISQUA |   1.00000    2.0000    25.000
  SIGMA_3 |   1.00000    2.0000    25.000
    MAD_3 |   1.00000    2.0000    25.000
    TUKEY |   1.00000     .         .
  CHAUVEN |   1.00000    2.0000     .
   GRUBBS |     .         .         .
   THOMPS |   1.00000    2.0000    25.000
    DIXON |     .         .         .


OVAL=
           |     DRP_1     DRP_2     DRP_3
-----------------------------------------
  CHISQUA |    126.00    132.00    158.00
  SIGMA_3 |    126.00    132.00    158.00
    MAD_3 |    126.00    132.00    158.00
    TUKEY |    126.00      .         .
  CHAUVEN |    126.00    132.00      .
   GRUBBS |      .         .         .
   THOMPS |    126.00    132.00    158.00
    DIXON |      .         .         .


CRIT=
               |   DRP_1_LOW   DRP_1_UPP   DRP_2_LOW
-------------------------------------------------------
   CHISQUA_TST |       0.007       0.093       0.019
   CHISQUA_THR |       0.050       0.050       0.050
   SIGMA_3_TST |      126.00      158.00      132.00
   SIGMA_3_THR |      134.69      156.67      137.19
     MAD_3_TST |      126.00      158.00      132.00
     MAD_3_THR |      140.00      152.00      141.25
```

```
    TUKEY_TST |      126.00      158.00        .
    TUKEY_THR |      129.50      161.50        .
   CHAUVEN_TST |     0.18137      2.3200     0.46751
   CHAUVEN_THR |     0.50000     0.50000     0.50000
    GRUBBS_TST |        .           .           .
    GRUBBS_THR |        .           .           .
    THOMPS_TST |      2.6849      1.6808      2.3362
    THOMPS_THR |      1.9011      1.9011      1.8985
     DIXON_TST |        .           .           .
     DIXON_THR |        .           .           .


                |   DRP_2_UPP   DRP_3_LOW   DRP_3_UPP
-----------------------------------------------------
   CHISQUA_TST |       0.064       0.097       0.048
   CHISQUA_THR |       0.050       0.050       0.050
   SIGMA_3_TST |      158.00      138.00      158.00
   SIGMA_3_THR |      155.81      138.87      155.39
     MAD_3_TST |      158.00      138.00      158.00
     MAD_3_THR |      151.75      142.50      151.50
     TUKEY_TST |        .           .           .
     TUKEY_THR |        .           .           .
   CHAUVEN_TST |      1.5336        .           .
   CHAUVEN_THR |     0.50000        .           .
    GRUBBS_TST |        .           .           .
    GRUBBS_THR |        .           .           .
    THOMPS_TST |      1.8529      1.6587      1.9746
    THOMPS_THR |      1.8985      1.8957      1.8957
     DIXON_TST |        .           .           .
     DIXON_THR |        .           .           .
```

2. Data from Internet: Chauvenetś Criterion

```
data = [ 9 10 10 10 11 50 ];
type = [" chisq sig_3 mad_3 tukey chauv grubb thomp dixon "];
optn = [ 2 ,   /* print */
         3 ];  /* drpmx */
< oind,oval,crit > = outlier(data,type,optn);


                        Outlier Index
                        *************

                    Dense Matrix (8 by 3)

                  |       DRP_1      DRP_2      DRP_3
```

```
          ------------------------------------
 CHISQUA |   6.0000000     .          .
 SIGMA_3 |   6.0000000     .          .
   MAD_3 |   6.0000000  5.0000000  1.0000000
   TUKEY |   6.0000000  5.0000000     .
 CHAUVEN |   6.0000000     .          .
  GRUBBS |   6.0000000     .          .
  THOMPS |   6.0000000     .          .
   DIXON |   6.0000000  5.0000000  1.0000000
```

Now, by means of setting optn[3] we can only report the lower outliers:

```
data = [ 9 10 10 10 11 50 ];
type = [" chisq sig_3 mad_3 tukey chauv grubb thomp dixon "];
optn = [ 2 ,   /* print */
         3 ,   /* drpmx */
         1 ];  /* return only lower */
< oind,oval,crit > = outlier(data,type,optn);
```

```
                        Outlier Index
                        *************

                     Dense Matrix (8 by 3)

                |       DRP_1      DRP_2      DRP_3
          ------------------------------------------
 CHISQUA |      .          .          .
 SIGMA_3 |      .          .          .
   MAD_3 |   1.0000000     .          .
   TUKEY |      .          .          .
 CHAUVEN |      .          .          .
  GRUBBS |      .          .          .
  THOMPS |      .          .          .
   DIXON |   1.0000000     .          .
```

Now, by means of setting optn[3] we can only report the upper outliers:

```
data = [ 9 10 10 10 11 50 ];
type = [" chisq sig_3 mad_3 tukey chauv grubb thomp dixon "];
optn = [ 2 ,   /* print */
         3 ,   /* drpmx */
         2 ];  /* return only upper */
< oind,oval,crit > = outlier(data,type,optn);
```

Outlier Index
****

Dense Matrix (8 by 3)

	DRP_1	DRP_2	DRP_3
CHISQUA	6.0000000	.	.
SIGMA_3	6.0000000	.	.
MAD_3	6.0000000	5.0000000	.
TUKEY	6.0000000	5.0000000	.
CHAUVEN	6.0000000	.	.
GRUBBS	6.0000000	.	.
THOMPS	6.0000000	.	.
DIXON	6.0000000	5.0000000	.

3. Data by Grubbs (1969) : Example 1

```
data = [ 568  570  570  570  572  572  572  578  584  596 ];
type = [" chisq sig_3 mad_3 tukey chauv grubb thomp dixon "];
optn = [ 2 ,   /* print */
         3 ];  /* drpmx */
< oind,oval,crit > = outlier(data,type,optn);
print "OIND=", oind;
print "OVAL=", oval;
print "CRIT=", crit;
```

Interestingly, Dixon's test does not find any outlier, but
all other methods find all three largest entries as outliers:

OIND=

	DRP_1	DRP_2	DRP_3
CHISQUA	10.0000	9.0000	8.0000
SIGMA_3	10.0000	9.0000	8.0000
MAD_3	10.0000	9.0000	8.0000
TUKEY	10.0000	9.0000	8.0000
CHAUVEN	10.0000	9.0000	8.0000
GRUBBS	10.0000	9.0000	8.0000
THOMPS	10.0000	9.0000	8.0000
DIXON	.	.	.

OVAL=

	DRP_1	DRP_2	DRP_3
CHISQUA	596.00	584.00	578.00
SIGMA_3	596.00	584.00	578.00
MAD_3	596.00	584.00	578.00
TUKEY	596.00	584.00	578.00
CHAUVEN	596.00	584.00	578.00
GRUBBS	596.00	584.00	578.00
THOMPS	596.00	584.00	578.00
DIXON	.	.	.

CRIT=

	DRP_1_LOW	DRP_1_UPP	DRP_2_LOW
CHISQUA_TST	0.40804	0.017	0.32926
CHISQUA_THR	0.050	0.050	0.050
SIGMA_3_TST	568.00	596.00	568.00
SIGMA_3_THR	562.15	588.25	565.37
MAD_3_TST	568.00	596.00	568.00
MAD_3_THR	569.00	575.00	569.00
TUKEY_TST	568.00	596.00	568.00
TUKEY_THR	558.00	590.00	567.00
CHAUVEN_TST	4.0804	0.16843	2.9633
CHAUVEN_THR	0.50000	0.50000	0.50000
GRUBBS_TST	0.82735	2.3901	0.97561
GRUBBS_THR	2.2900	2.2900	2.2150
THOMPS_TST	0.82735	2.3901	0.97561
THOMPS_THR	1.7984	1.7984	1.7770
DIXON_TST	.	.	.
DIXON_THR	.	.	.

	DRP_2_UPP	DRP_3_LOW	DRP_3_UPP
CHISQUA_TST	0.027	0.23958	0.029
CHISQUA_THR	0.050	0.050	0.050
SIGMA_3_TST	584.00	568.00	578.00
SIGMA_3_THR	580.41	567.04	575.96
MAD_3_TST	584.00	568.00	578.00
MAD_3_THR	575.00	569.50	572.50
TUKEY_TST	584.00	568.00	578.00
TUKEY_THR	575.00	567.00	575.00
CHAUVEN_TST	0.23942	1.9166	0.23166
CHAUVEN_THR	0.50000	0.50000	0.50000
GRUBBS_TST	2.2173	1.1760	2.1841
GRUBBS_THR	2.2150	2.1266	2.1266

```
THOMPS_TST |      2.2173      1.1760      2.1841
THOMPS_THR |      1.7770      1.7491      1.7491
 DIXON_TST |         .           .           .
 DIXON_THR |         .           .           .
```

4. Difference of critical values for Thompson τ and Grubbs test: Thompson is using for the two-sided test the critical value $t_{\alpha/2}$, whereas Grubbs is using $t_{\alpha/(n2)}$ but both with $df = n-2$.

```
print "Test some t values for Thompson tau test: p=1.-alfa/2";
na = [ 3 4 5 6 7 8 9 10 ];
for (i = 1; i <= ncol(na); i++) {
  n = na[i];
  alfa = .05; prb = 1. - .5 * alfa; df = n - 2;
  ta = tmis(prb,.,df);
  tau = ta * (n - 1.) / (sqrt(n) * sqrt((n - 2 + ta *ta)));
  print "n=", n, " ta=", ta," crt=", tau;
}
```

```
Test some t values for Thompson tau test: p=1.-alfa/2
n= 3 ta= 12.706 crt= 1.1511
n= 4 ta= 4.3027 crt= 1.4250
n= 5 ta= 3.1824 crt= 1.5712
n= 6 ta= 2.7764 crt= 1.6563
n= 7 ta= 2.5706 crt= 1.7110
n= 8 ta= 2.4469 crt= 1.7491
n= 9 ta= 2.3646 crt= 1.7770
n= 10 ta= 2.3060 crt= 1.7984
```

```
print "Test some t values for Grubbs test: p=alfa/2n";
na = [ 3 4 5 6 7 8 9 10 ];
for (i = 1; i <= ncol(na); i++) {
  n = na[i]; df = n - 2;
  alfa = .05; prb = alfa / (2. * n);
  tq = tmis(prb,.,df); rat = (n - 1.) / sqrt(n);
  crt = rat * sqrt(tq * tq / (n - 2. + tq * tq));
  print "n=", n, " tq=", tq," crt=", crt;
}
```

```
Test some t values for Grubbs test: p=alfa/2n
n= 3 tq=-38.188 crt= 1.1543
n= 4 tq=-8.8602 crt= 1.4812
```

```
n= 5 tq=-5.8409 crt= 1.7150
n= 6 tq=-4.8510 crt= 1.8871
n= 7 tq=-4.3818 crt= 2.0200
n= 8 tq=-4.1152 crt= 2.1266
n= 9 tq=-3.9467 crt= 2.2150
n= 10 tq=-3.8325 crt= 2.2900
```

5.8 The pid() Function

```
id = pid()
```

Purpose: The pid() function returns the integer ID of the running process.

Input: The function does not need any input.

Output: The function returns the unique integer process ID. Note, you may get a different process ID at different runs of CMAT.

Restrictions:

Relationships: fnampid()

Examples: The process ID of that run at this time on my PC is:

```
Fname with PID=_pcx_mps_4132 Current Process ID= 4132
```

5.9 The svmfsm() Function

This function was created for the easier use of the two feature selection methods since the two options "meth" and "task" of the svm() function must not be specified. But, if you use the "meth" = "FSM" for classification or "meth" = "FLP" for regression you should get (essentially) the same result.

alfa = svmfsm(train,model,<,optn<,class<,ctun<,x0<,kfun<,test>>>>>)

<alfa,sres,vres,yptr,yptt,plan> = svmfsm(train,model,<,optn<,class<,ctun<,...>>>)

Purpose: The svmfsm() function implements algorithms for feature selection for applications with

- SV C classification (response is nominal scaled class variable): FSM algorithm by Fung & Mangasarian (2003) : in addition of tuning c, this algorithm needs the tuning of two additional parameters α and δ, corresponds to the "meth" = "FSM" option for function svm().
- and SV ν regression (response is interval scaled): algorithm by Bi, Bennett, Embrechts, Breneman, Song (2002) : in addition of tuning c, this algorithm needs the tuning of one additional parameters ν, corresponds to the "meth" = "FLP" option for function svm().

Both algorithms are $L1$ parameter estimation methods which for correct parameter tuning result in sparse parameter estimates.

As in svm() the "modif" option can be specified:

Classification: as with "meth"= "FSM":

"modif"= 0 default Fung & Mangsarian (2003) FSM algorithm : $L1$ estimation may result in sparse coefficient vector. Here an additional tuning paramter can be specified with the "peneps" option, $0. < peneps < 1.$, default is 0.1.

"modif"= 1 Fung & Mangsarian (2003) NSVM algorithm : $L2$ estimation normally does not result in a sparse coefficient vector.

Regression: as with "meth"= "FLP": As in function svm() for obtaining sparse coefficient vectors this method requires the specification of two tuning parameters $C > 0.$ using the first row of input argument ctun and $\nu \in (0., 1.]$ using the second row of input argument ctun. The "modif" option can be used for specifying different LP methods:

"modif"= 0 using PCx algorithm by Czyzyk et al. is default

"modif"= 1 using LP algorithm by Madsen et al.

"modif"= 2 using CLP algorithm by J. J. Forrest

Input: See function svm() for the input arguments.

train specifies a $N \times m$ matrix or vector of training data, i.e. data which are used to obtain parameters that specify a specific model. For solving the binary classification problem the response variable must be a binary CLASS variable.

model : The analysis model is specified in form of a string, e.g. model= "3=1 2", containing column numbers for variables. The syntax of the model string argument is the same as for the glmod() function except for the additional *events / trial* response specification. ????

optn The option argument is specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. Most of the options of function svm() do apply, but the options for "meth" and "task" must not be specified.

class specifies a vector CLASS variables in form of a vector of integers which defining the column numbers of the training data set.

ctun The form of input depends on the "ctun" option:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of C,
- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function
     "K4" fourth parameter of kernel function
     "C" regularization parameter C
     "NU" NU classification or regression ν
     "EP" eps value for epsilon regression
     "AL" parameter α for FSM classification
     "DE" parameter δ for FSM classification
  2. remaining columns $k = 2, ...$:
     - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)

– for "tun" = "patt" or "tun" = "opti": ctun must have only two columns $k = 2$ specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

x0 specifies a vector of nr initial estimates α. If the response (target) is multinomial with K categories, this must be a $K \times nr$ matrix. That means this input argument corresponds in size to the first output argument α.

kfun optional string argument specifying a user defined function for kernel evaluation. The function definition must be of the form: function kernel(vi,vj) where the input arguments vi and vj are observation vectors of the data and the function must be a real scalar. Note, that Mercer's condition must be satisfied by the functions kernel definition, i.e. the resulting matrices $\mathbf{Q}$ of the QP must be positive (semi)definite.

test optional argument specifying a matrix or vector of test data, i.e. data for which predicted model values should be obtained based on the parametric model obtained from the training data. The column structure for the test data set should be the same as that of the training data set since the model applies also at the test data.

Output: See function svm() for the output arguments.

Restrictions: 1. There is no complex version of this function.

2. Observations with missing values in the X data (predictors) are not used.

3. Note, that for interfacing with CLP a temporary _svm....mps file is written into the work directory. The unique process ID is added to the file name to prevent confusion when running multiple processes at the same computer.

Relationships: svm(), svmmat(), svmpred(), svmstw()

Examples: 1. Feature selection with NIR Spectra Data: Classification

The NIR Spectra training data set has 21 observations (rows) and 268 variables (columns). A test data set with 7 observations is used for evaluation. For both data sets the interval response was made to binary using the median as criterion. For the test example see cmat\test\tsvm19.inp.

```
  options NOECHO;
#include "..\\tdata\\nir.dat"
  options ECHO;
  sopt = [ "ari" "std" "med" ];
  mom = univar(ytrn',sopt);
```

```
/* print "Moments of y=", mom; */
cutof = mom[3];
d = ytrn .< cutof; y = replace(d,0.,-1.)‘;
train = xtrn -> y;
d = ytst .< cutof; y = replace(d,0.,-1.)‘;
test = xtst -> y;
```

For binary reponse, svmfsm() is using the FSM method by Fung & Mangasarian (2003) which needs tuning parameters "peneps", α, and δ. Here we set only "peneps" and use the automatic tuning w.r.t. α and δ:

```
print "FSM: with line search: peneps=.001: 3 nonzeros, nmis=0";
modl = "269 = 1:268";
class = 269;
optn = [ "print"           2 ,
         "pplan"           1 ,
         "peneps"       .001 ,
         "kern"       "line" ];
c = 1.;
alfa = svmfsm(train,modl,optn,class,c,.,.,test);
```

```
[note] For missing Alpha parameter choose: 0.001 (* 10) 1000

[note] For missing Delta parameter choose: 1e-005 (* 10) 0.1
```

The following shows the initial output concerning the model specification and the input data as with the svm() function:

```
                        *****************
                        Model Information
                        *****************

                  Number Valid Observations   21
                  Observations Test Data       7
                  Response Variable        Y[269]
                  N Independent Variables    268
                  Feature Selection Classification
                  Model Without Linear Constraint
                  Estimation Method          FSM
                  Kernel Function         Linear
                  Use Unscaled Predictor
                  Bias Corrected Predicted Values
```

```
                    ************************
                    Class Level Information
                    ************************

  Class  Level    Value

Y[269]      2       -1       1
        Memory needed for Kernel matrix: 0.00176239 Mb
    Traindata stored incore: Mem need: 0.043 Mem spec: 2 Mb


            ***************************************
            Number of Observations for Class Levels
            ***************************************

            Variable       Value    Nobs  Proportion
            Y[269]            -1      11   52.380952
                               1      10   47.619048

            ***************************************
            Number of Observations for Class Levels
            ***************************************

            Variable       Value    Nobs  Proportion
            Y[269]            -1       5   71.428571
                               1       2   28.571429


 [note] Estimated Regularization Parameter C=0.00114993
```

The following output is a short version of the optimization results for all grid points of the α and δ parameters. The output shows:

- The grid values for the δ and α parameters.
- The optimal function value of the optimization.
- The number of misclassifications of the predicted values for training and test data set.
- The number of nonzeros in the vector of linear plane coefficients.
- The number of iterations needed for each optimization.

Using the "popt" option and/or a higher value of the "print" option the detail iteration history would be printed.

```
           ---Start Training Cycle: Technique=  FSM---

       Delta      Alpha  Criterion    MisTrn    MisTst Nzero Niter
```

1.00e-005	0.0010000	-0.0097368	0	0	3	25
1.00e-004	0.0010000	-0.0097368	0	0	3	22
0.0010000	0.0010000	-0.0097368	0	0	3	46
0.0100000	0.0010000	-0.0097317	0	0	3	100
0.1000000	0.0010000	-0.0082891	1	0	4	100
1.00e-005	0.0100000	-0.0090846	0	0	4	28
1.00e-004	0.0100000	-0.0090846	0	0	4	20
0.0010000	0.0100000	-0.0090846	0	0	4	22
0.0100000	0.0100000	-0.0090846	0	0	4	89
0.1000000	0.0100000	-0.0088541	0	0	4	100
1.00e-005	0.1000000	-0.0087802	0	0	5	26
1.00e-004	0.1000000	-0.0087802	0	0	5	20
0.0010000	0.1000000	-0.0087802	0	0	5	21
0.0100000	0.1000000	-0.0087802	0	0	5	76
0.1000000	0.1000000	-0.0086923	0	0	5	100
1.00e-005	1.0000000	-0.0087223	0	0	4	45
1.00e-004	1.0000000	-0.0087223	0	0	4	31
0.0010000	1.0000000	-0.0087223	0	0	4	18
0.0100000	1.0000000	-0.0087223	0	0	4	73
0.1000000	1.0000000	-0.0085907	1	0	4	100
1.00e-005	10.000000	-0.0087164	0	0	4	64
1.00e-004	10.000000	-0.0087164	0	0	4	33
0.0010000	10.000000	-0.0087164	0	0	4	31
0.0100000	10.000000	-0.0087164	0	0	4	88
0.1000000	10.000000	-0.0084675	1	0	4	100
1.00e-005	100.00000	-0.0079714	0	0	3	100
1.00e-004	100.00000	-0.0087158	0	0	4	65
0.0010000	100.00000	-0.0087158	0	0	4	50
0.0100000	100.00000	-0.0087149	0	0	4	100
0.1000000	100.00000	-0.0084353	1	0	4	100
1.00e-005	1000.0000	-0.0011049	5	2	2	100
1.00e-004	1000.0000	-0.0087158	0	0	4	73
0.0010000	1000.0000	-0.0087158	0	0	4	76
0.0100000	1000.0000	-0.0087158	0	0	4	79
0.1000000	1000.0000	-0.0078315	0	0	3	100

There are 11 optimizations which take the upper limit of 100 iterations, from those are 3 with large gradient and eight without further progress as indicated below:

```
[warning] There are 3 solutions with large gradient.
[warning] There are 8 solutions which terminated due to no
          progress.
```

For illustrating the grid search w.r.t. parameters (α, δ) the

follwing information is pribnted in table form:

- the number of misclassifications for the test data set
- the SSQ error of predicted values for the test data set
- the number of misclassifications for the training data set
- the SSQ error of predicted values for the training data set
- the number of nonzeros in the vector of linear plane coefficients.

Misclassification (Test) for (alpha,delta) (C=1)
****

Dense Matrix (7 by 5)

	d1e-005	d0.0001	d0.001	d0.01	d0.1
a0.001	0	0	0	0	0
a0.01	0	0	0	0	0
a0.1	0	0	0	0	0
a1	0	0	0	0	0
a10	0	0	0	0	0
a100	0	0	0	0	0
a1000	2.0000000	0	0	0	0

SSE (Test) for (alpha,delta) (C=1)
****

Dense Matrix (7 by 5)

	d1e-005	d0.0001	d0.001	d0.01	d0.1
a0.001	3.0471542	3.0464499	3.0483839	3.0758676	3.4347574
a0.01	2.7762821	2.7761835	2.7760582	2.7766734	3.0054326
a0.1	2.5193449	2.5193549	2.5192169	2.5190787	2.6482143
a1	2.4595360	2.4598707	2.4592203	2.4602486	2.4321525
a10	2.4594458	2.4593337	2.4588433	2.4600570	2.4614542
a100	3.4981126	2.4595623	2.4599499	2.4646633	2.4734352
a1000	115.67037	2.4595604	2.4599562	2.4600092	2.7808243

Misclassification (Training) for (alpha,delta) (C=1)
****

Dense Matrix (7 by 5)

	d1e-005	d0.0001	d0.001	d0.01	d0.1
a0.001	0	0	0	0	1.0000000
a0.01	0	0	0	0	0
a0.1	0	0	0	0	0
a1	0	0	0	0	1.0000000
a10	0	0	0	0	1.0000000
a100	0	0	0	0	1.0000000
a1000	5.0000000	0	0	0	0

SSE (Training) for (alpha,delta) (C=1)

Dense Matrix (7 by 5)

	d1e-005	d0.0001	d0.001	d0.01	d0.1
a0.001	5.0275834	5.0280935	5.0266946	5.0085637	5.4258258
a0.01	6.4341486	6.4335277	6.4334823	6.4355241	6.2161979
a0.1	15.630160	15.629344	15.640175	15.651511	11.533176
a1	16.287160	16.292280	16.281955	16.298308	17.009708
a10	16.292711	16.290961	16.283298	16.302262	18.439951
a100	64.868723	16.295258	16.301286	17.323952	18.287235
a1000	1913.5867	16.295299	16.301453	16.302279	7.3579442

Number Nonzeros for (alpha,delta) (C=1)

Dense Matrix (7 by 5)

	d1e-005	d0.0001	d0.001	d0.01	d0.1
a0.001	3.0000000	3.0000000	3.0000000	3.0000000	4.0000000
a0.01	4.0000000	4.0000000	4.0000000	4.0000000	4.0000000
a0.1	5.0000000	5.0000000	5.0000000	5.0000000	5.0000000
a1	4.0000000	4.0000000	4.0000000	4.0000000	4.0000000
a10	4.0000000	4.0000000	4.0000000	4.0000000	4.0000000
a100	3.0000000	4.0000000	4.0000000	4.0000000	4.0000000
a1000	2.0000000	4.0000000	4.0000000	4.0000000	3.0000000

C=1 : Solution for delta=1e-005 alpha=0.001 selected based on Test fit.

The following output shows the nonzero values of linear plane coefficients:

```
           Nonzeros of Separating Plane (w*x = 6.04377)
           -------------------------------------------

                    1  20  X20_   0.682529495
                    2  98  X98_  -2.016126691
                    3  99  X99_  -0.606268743

             Largest 3 Plane Coefficients (Sorted)
             *************************************

                    1  98  X98_ -2.016126691
                    2  20  X20_  0.682529495
                    3  99  X99_ -0.606268743
             Sparsity: 3 Nonzeros 265 Zeros (C=1)
```

For the best solution, there are no misclassifications, neither for scoring with the training nor the test data set:

```
                *******************************
                Evaluation of Training Data Fit
                *******************************

                     Classification Table
                     --------------------

                               |    Predicted
                     Observed  |       -1       1
                     ----------|-----------------
                           -1  |       11       0
                            1  |        0      10


               ***********************************
               Evaluation of Test Data Set Data Fit
               ***********************************

                     Classification Table
                     --------------------

                               |    Predicted
                     Observed  |       -1       1
                     ----------|-----------------
                           -1  |        5       0
```

```
                                1 |        0       2

  Regularization Parameter C . . . . . . . . . . . . . . . . . . 1
  Kernel Function. . . . . . . . . . . . . . . . . .          Linear
  Norm of Longest Vector . . . . . . . . . . . . . . . . .  31.955
  Number Misclassifactions (Training Data) . . . . . . . . . . . 0
  Number Misclassifactions (Test Data) . . . . . . . . . . . . . 0
  Number Fast Runs through TrainData . . . . . . . . . . . . 12591
  Number Slow Runs through TrainData . . . . . . . . . . . . 10411
  Total Number of Kernel Calls . . . . . . . . . . . . . . . . 330
  Time for Optimization. . . . . . . . . . . . . . . . . . . . . 1
  Total Processing Time. . . . . . . . . . . . . . . . . . . . . 1
  Optimization Criterion . . . . . . . . . . . . . . . -0.00973677
  Infinity Norm of Gradient. . . . . . . . . . . .  1.99735e-008
  Geometric Margin . . . . . . . . . . 0.903676 (|w|^2= 4.89818)
  Number Support Vectors . . . . . . . . . . . . . 21 (100.00 %)
  Number Support Vectors on Margin . . . . . . . . . . . . .  11
  Bias . . . . . . . . . . . . . . . . . . . . . . . . 0.0570279
  Radius of Sphere Around SV . . . . . . . . . . . . . . .  31.955
  Estimated VCdim of Classifier. . . . . . . . . . . . . . 5002.62
  Linear Kernel Constant (Fit) . . . . . . . . . . . . . . 6.04377
  Linear Kernel Constant (PCE) . . . . . . . . . . . . . .   .
```

The algorithm is able to find those three columns (features, variables) from the data sets of 268 columns (features, variables) which perfectly predict the values of the binary response (of training and test data) of the classification model.

2. Feature selection with NIR Spectra Data: Regression

   The training and test data are the same as above, However, the response now consists of the original interval scaled data (and is not made to binary). For this test example see cmat\test\tsvm19.inp.

```
  options NOECHO;
#include "..\\tdata\\nir.dat"
  options ECHO;


  train = xtrn -> ytrn‘;
  test  = xtst -> ytst‘;

  print "FLP: nonzeros, nmis=0";
  modl = "269 = 1:268";
  tun = [ "C"   1. 10. 30. 50. 100. ,
          "NU"  .01  .1  .2  .5 .8 ];
```

```
print "modif=0: PCX algorithm";
optn = [ "print"           2 ,
         "pplan"           2 ,
         "tun"        "grid" ,
         "kern"       "line" ];
alfa = svmfsm(train,modl,optn,.,tun,.,.,test);
```

The following is the best result obtained for $C = 100$ and $\nu = 0.5$:

```
*******************************************************
**********- Evaluation for Best C=100 Nu=0.5-**********
*******************************************************

Regularization Parameter C . . . . . 100
Parameter NU . . . . . 0.5
Kernel Function. . . . . Linear
Norm of Longest Vector . . . . . 31.955
Mean Squared Error (MSE, Training Data). . . . . 0.0803305
Mean Squared Error (MSE, Test Data). . . . . 0.0398418
Mean Absolute Error (MAE, Training Data) . . . . . 0.22297
Average Loss (Training Data) . . . . . 0.22297
Average Positive Loss (Training Data). . . . . 0.195098 (N=12)
Average Negative Loss (Training Data). . . . . 0.260131 (N=9)
Mean Absolute Error (MAE, Test Data) . . . . . 0.160299
Average Loss (Test Data) . . . . . 0.160299
Average Positive Loss (Test Data). . . . . 0.160299 (N=7)
Average Negative Loss (Test Data). . . . . 0 (N=0)
Number Fast Runs through TrainData . . . . . 106
Number Slow Runs through TrainData . . . . . 136
Total Number of Kernel Calls . . . . . 1380
Time for Optimization. . . . . 11
Total Processing Time. . . . . 11


          Grid Search Results: L1 Error (Training)
          ****************************************

                   Dense Matrix (5 by 5)

     |      Nu0.01      Nu0.1      Nu0.2      Nu0.5      Nu0.8
   ------------------------------------------------------------
  c1 |   889.25207  889.25207  889.25207  889.25207  889.25207
 c10 |   889.25207  889.25207  889.25207  7.0508036  5.9004114
```

c30	889.25207	5.5289306	4.8580753	3.8183665	3.5656172
c50	889.25207	6.4758037	3.6822417	0.1948526	0.1756181
c100	889.25206	4.2347019	0.2128654	0.0803305	0.0934748

Grid Search Results: L2 Error (Training)

Dense Matrix (5 by 5)

	Nu0.01	Nu0.1	Nu0.2	Nu0.5	Nu0.8
c1	18674.293	18674.293	18674.293	18674.293	18674.293
c10	18674.293	18674.293	18674.293	148.06688	123.90864
c30	18674.293	116.10754	102.01958	80.185697	74.877962
c50	18674.293	135.99188	77.327076	4.0919053	3.6879799
c100	18674.293	88.928740	4.4701741	1.6869412	1.9629702

Grid Search Results: L1 Error (Test)

Dense Matrix (5 by 5)

	Nu0.01	Nu0.1	Nu0.2	Nu0.5	Nu0.8
c1	136.28951	136.28951	136.28951	136.28951	136.28951
c10	136.28951	136.28951	136.28951	0.6047261	0.8289635
c30	136.28951	0.3441866	0.3524950	1.4137789	1.8186339
c50	136.28951	0.3854443	0.9808013	0.0796735	0.0709915
c100	136.28951	0.2501561	0.0707189	0.0398418	0.0831037

Grid Search Results: L2 Error (Test)

Dense Matrix (5 by 5)

	Nu0.01	Nu0.1	Nu0.2	Nu0.5	Nu0.8
c1	954.02660	954.02660	954.02660	954.02660	954.02660
c10	954.02660	954.02660	954.02660	4.2330825	5.8027443
c30	954.02660	2.4093061	2.4674647	9.8964524	12.730437
c50	954.02660	2.6981103	6.8656093	0.5577142	0.4969405
c100	954.02659	1.7510930	0.4950323	0.2788924	0.5817260

FSM Grid Search: N Nonzero Coefficients

```
************************************

        Dense Matrix (5 by 5)

     |  Nu0.01  Nu0.1  Nu0.2  Nu0.5  Nu0.8
  ------------------------------------------
  c1 |     125    230      0    252      0
 c10 |       0      0     52      3      2
 c30 |       0      2      2      3      2
 c50 |     268      3      3      5      5
c100 |     267      3      5      6      6

Duality Gap. . . . . . . . . . . . . . . . . . . . . . 2.049e-006
Infeasibility. . . . . . . . . . . . . . . . . . . . . 7.3474e-010
Geometric Margin . . . . . . . . . 0.0234826 (|w|^2= 7253.85)
Number Support Vectors . . . . . . . . . . . . . . 21 (100.00 %)
Number Support Vectors on Margin . . . . . . . . . . . . . . . 8
Bias . . . . . . . . . . . . . . . . . . . . . . . . -0.00166365
Radius of Sphere Around SV . . . . . . . . . . . . . . 31.955
Estimated VCdim of Classifier. . . . . . . . . . 7.40705e+006
Linear Kernel Constant (Fit) . . . . . . . . . . . . . . 52.5292
Linear Kernel Constant (PCE) . . . . . . . . . . . . . .   .

      Nonzeros of Separating Plane (w*x = 52.5292)
      --------------------------------------------

          1  16  X16_   0.891033591
          2  23  X23_  -21.71266072
          3  52  X52_  -49.06926973
          4  72  X72_   65.84941206
          5  98  X98_  -0.458933259
          6 130  X130_  6.121376607


       Largest 6 Plane Coefficients (Sorted)
       *************************************

          1  72  X72_   65.84941206
          2  52  X52_  -49.06926973
          3  23  X23_  -21.71266072
          4 130  X130_  6.121376607
          5  16  X16_   0.891033591
          6  98  X98_  -0.458933259
    Sparsity: 6 Nonzeros 262 Zeros (C=100)
```

5.10 The svmstw() Function

Part of what this function is doing has been part of the old implementation of the svm() function but has been removed. Splitting this function off made it easier to include more specific properties to the stepwise approach.

```
alfa = svmstw(train,model,<,optn<,class<,ctun<,test>>>>)
```

```
<alfa,sres,vres,yptr,yptt,plan> = svmstw(train,model,<,optn<,class<,ctun<,test>>>)
```

Purpose: The svmstw() function implements a set of algorithms for forward and backward stepwise variable selection. This stepwise feature selection is implemented only for the linear kernel function. The stepwise approach is using stepwise svm solutions for adding or dropping a model effect to or from the model generating a solution with a specified number of nonzero coefficients.

- The forward stepwise algorithm adds in each step that effect which is expected to improve the model fit most.
- The backward stepwise algorithm drops in each step that effect which is expected to reduce the model fit least.

There are essentially two different approaches to feature selection:

- the *all-relevant* problem: related to backward stepwise selection,
- the *minimal-optimal* problem: related to forward stepwise selection.

backward stepwise method is a modified algorithm similar to the stepwise feature selection method by Guyon & Elisseeff(2003) for applications with

- SV C classification (response is nominal scaled class variable):
- and SV ν regression (response is interval scaled).

The algorithm is a backward elimination where in each step the SVM parametes are estimated for a specific number of variables. The number of variables is dropped off the model based on one of the two criteria:

1. computationally expensive: the smallest fit increase when one of the parameters of the linear model is set to zero, this needs one more run through the data set
2. computationally inexpensive: the variable corresponding to the smallest parameter of the linear model is dropped.

The computationally more expensive method is default since it seems to obtain the better results.

forward stepwise method is a newly invented algorithm which tries to improve the goodness of the fit with each newly added variable. This method is preferred for applications where there are many more variables than observations as in genetic data.

Not all SVM methods of function svm() can be used here for each step. See table below for valid methods.

Input: See function svm() for the input arguments:

train specifies a $N \times m$ matrix or vector of training data, i.e. data which are used to obtain parameters that specify a specific model. For solving the binary classification problem the response variable must be a binary CLASS variable.

model : The analysis model is specified in form of a string, e.g. model= "3=1 2", containing column numbers for variables. The syntax of the model string argument is the same as for the glmod() function except for the additional *events / trial* response specification. ????

optn The option argument is specified in form of a two column matrix where the first column defines the option as string value (in quotes) and the second column can be used for a numeric or string specification of the option. See below for details.

class specifies a vector CLASS variables in form of a vector of integers which defining the column numbers of the training data set.

ctun The form of input depends on the "ctun" option:

- ctun is $K = k = 1$, missing value: no input tuning parameter, default values are being used,
- ctun is $K = k = 1$, real scalar and specifies a scalar choice of C,
- ctun is $K \geq 1$ and $k \geq 2$:
  1. first column must contain one of the following string values:
     "K1" first parameter of kernel function
     "K2" second parameter of kernel function
     "K3" third parameter of kernel function
     "K4" fourth parameter of kernel function
     "C" regularization parameter C
     "NU" NU classification or regression ν
     "EP" eps value for epsilon regression
     "AL" parameter α for FSM classification

"DE" parameter δ for FSM classification

2. remaining columns $k = 2, ...$:
   - for "tun" = "grid": columns $k = 2, ...$ contain one or more real values for grid search (missing values are neglected)
   - for "tun" = "patt" or "tun" = "opti": ctun must have only two columns $k = 2$ specifying the lower and upper bound (missing values are not permitted), $lb_k \leq ub_k$ for pattern search.

test optional argument specifying a matrix or vector of test data, i.e. data for which predicted model values should be obtained based on the parametric model obtained from the training data. The column structure for the test data set should be the same as that of the training data set since the model applies also at the test data.

Most of the options of function svm() do apply, but the "task" option must not be specified. The following are additional or modified options that could be specified for svmstw():

Option Name	Second Column	Meaning
"algv"	int	
"fscrit"	string	lower bound of var number for stepwise var selection
	"xfit"	expected change of fit criterion
	"csiz"	magnitude of linear coefficient
	"mixd"	"xfit" for $n < N$ and "csiz" for $n > N$
	"cold"	undocumented version
"meth"		method of estimation, default depends on problem
	"asvm"	for classification: linear $L2$ SVM similar to Mangasarian & Musicant
	"fsm"	for classification: $L1$ sparse feature selection, similar to Fung & Mangsarian
	"flp"	currently only for ν regression: Bi et.al (2002)
	"flp"	"modif" =0 : using PCx algorithm (Czyzyk et al.)
	"flp"	"modif" =1 : using LP algorithm by Madsen et al.
	"flp"	"modif" =2 : using CLP algorithm by J. J. Forrest
	"fqp"	for classification and regression: full QP may be modified with "mmm" for Mangasarian & Musicant modification; null and rage space technique
	"fqp"	"modif" =1 : for classification and regression: full QP: modified version: use Penalty method
	"lsqu"	for classification and regression: Suykens & Vandevalle least squares SVM: similar to proximal SVM and ridge regression; solution by direct (dense or sparse) Cholesky or iterative conjugate gradient
	"lsvm"	for classification: Lagrangean $L2$ SVM similar to Mangasarian & Musicant
	"nsvm"	for classification: $L2$ feature selection Newton SVM, similar to Fung & Mangasarian
	"psvm"	for classification: Proximal $L2$ SVM similar to Fung & Mangasarian; direct (dense or sparse) Cholesky or iterative conjugate gradient
	"smo"	for classification and regression: similar to Lin & Chen
	"smo"	"modif" =1 : SMO (sequential minimal opt.) by Platt
	"smo"	"modif" =2 : SMO 1 by Keerthi et.al.
	"smo"	"modif" =3 : SMO 2 by Keerthi et.al.
"nvarend"	int	forward stepwise: upper bound of var number backward stepwise: lower bound of var number
"selec"	string	kind of stepwise selection
	'f'	forward
	'b'	backward
"vardrop"	real	drop rate for backward stepwise variable selection

Output: See function svm() for the output arguments.

Restrictions: 1. There is no complex version of this function.

2. Observations with missing values in the X data (predictors) are not used.

3. Note, that for interfacing with CLP a temporary _svm....mps file is written into the work directory. The unique process ID is added to the file name to prevent confusion when running multiple processes at the same computer.

Relationships: svm(), svmmat(), svmpred(), svmfsm()

Example: 1. Feature selection with Heart data: Classification

There is a train data set with 210 observations (cases), and a test data set with 60 observations, and each data set has 14 effects (features) with a binary response, see test tsvm7.inp.

```
print "Heart data: train: Nobs=210, test: Nobs=60; nvar=14";
data = rspfile("..\\tdata\\heart_210.dat");
test = rspfile("..\\tdata\\heart_60.dat");
modl = "1 = 2:14";
clas = 1;
```

- Forward stepwise: Using FQP method:

```
print "---SVMSTW: forward stepwise with FQP, C=1 ---";
print "Found [4,13,14] is 1. best";
clas = 1;
modl = "1 = 2:14";
optn = [ "print"           2 ,
         "popt"            0 ,
         "pplan"           1 ,
         "sel"           'f' ,
         "meth"        "fqp" ,
         "nvarend"         3 ,
         "peneps"       .001 ];
c = 1.;
< alfa,sres,vres,yptr,yptt,plan > = svmstw(data,modl,optn,clas,c,test);
```

```
                        *****************
                        Model Information
                        *****************

                  Number Valid Observations  210
                  Observations Test Data      60
```

```
                Response Variable          Y[1]
                N Independent Variables     13
                Stepwise Feature Sel. Classification
                Model Without Linear Constraint
                Estimation Method          FQP
                Kernel Function         Linear
                Use Unscaled Predictor
                Bias Corrected Predicted Values

                   ***********************
                   Class Level Information
                   ***********************

Class  Level    Value

Y[ 1]      2       -1       1
        Memory needed for Kernel matrix: 0.169029 Mb
   Traindata stored incore: Mem need: 0.021 Mem spec: 2 Mb

    ****************************************************
    Forward Variable Selection (Ntrn=210 Nvar=13 Ntst=60)
    ****************************************************

 Iter   VAdd   NVin     Err_Trn     Err_Tst  MiscTrn  MiscTst
    1     13      1  209.749321  59.9375447       93       27
    2      3      2  187.812972  59.4774832       49       15
    3     12      3  153.200400  69.5874591       45       16

              *******************************
              Evaluation of Training Data Fit
              *******************************

   Index                              Value       StdErr
   Absolute Classification Error         40            .
   Classification Accuracy        80.95238095         .
   Percent Concordant Pairs       64.59884202         .
   Percent Discordant Pairs       3.639371381         .
   Percent Tied Pairs             31.76178660         .
   c                              0.804797353         .
   Goodman-Kruskal Gamma          0.893333333  0.035728960
   Kendall Tau_a                  0.302255639         .
   Kendall Tau_b                  0.612756467  0.054966692
   Stuart Tau_c                   0.601632653  0.055376198
   Somers D C|R                   0.609594706  0.055268003
   Somers D R|C                   0.615934627  0.055156776
```

Classification Table

Observed	Predicted -1	Predicted 1
-1	99	18
1	22	71

****
Evaluation of Test Data Set Data Fit
****

Index	Value	StdErr
Absolute Classification Error	12	.
Classification Accuracy	80.00000000	.
Percent Concordant Pairs	63.63636364	.
Percent Discordant Pairs	4.040404040	.
Percent Tied Pairs	32.32323232	.
c	0.797979798	.
Goodman-Kruskal Gamma	0.880597015	0.072587790
Kendall Tau_a	0.300000000	.
Kendall Tau_b	0.595959596	0.104216288
Stuart Tau_c	0.590000000	0.104530697
Somers D C|R	0.595959596	0.104447843
Somers D R|C	0.595959596	0.104447843

Classification Table

Observed	Predicted -1	Predicted 1
-1	27	6
1	6	21

Regularization Parameter C	1
Kernel Function	Linear
Norm of Longest Vector	3.28753
Number Misclassifactions (Training Data)	40
Number Misclassifactions (Test Data)	12
Number Fast Runs through TrainData	23
Number Slow Runs through TrainData	39

```
Total Number of Kernel Calls . . . . . 112586
Time for Optimization. . . . . 2
Total Processing Time. . . . . 49166
Optimization Criterion . . . . . -84.6667
Infinity Norm of Gradient. . . . . 1.55585e-006
Geometric Margin . . . . . 1.1547 (|w|^2= 3)
Number Support Vectors . . . . . 88 ( 41.90 %)
Number Support Vectors on Margin . . . . . 84
Bias . . . . . 1.13546e-016
Radius of Sphere Around SV . . . . . 1.73205
Estimated VCdim of Classifier. . . . . 9.99999
Linear Kernel Constant (Fit) . . . . . 0.160984
Linear Kernel Constant (PCE) . . . . . .
```

As we can see below, the selection of $(X4, X13, X14)$ is also the best of all 3-feature linear regression models w.r.t. R^2:

```
          Nonzeros of Separating Plane (w*x = 0.160984)
          ---------------------------------------------

                    1  3  X4_   1.000000458
                    2 12  X13_  0.999998739
                    3 13  X14_  0.999998741

               Largest 3 Plane Coefficients (Sorted)
               *************************************

                    1  3  X4_   1.000000458
                    2 13  X14_  0.999998741
                    3 12  X13_  0.999998739
                Sparsity: 3 Nonzeros 10 Zeros (C=1)
```

For 13 features it is computationally feasible to compute the 286 linear regression analyses of all combinations of 3 features:

```
  print "Linear Regression : 20 best solutions for all three vars";
  print "[ 4,13,14 ] is 1. best, and [4,10,14] is 5. best";
  modl = "1 = 2:14";
  optn = [ "print"         3 ,
           "best"         20 ,
           "selc"       "r2" ,
           "sing"     1.e-12 ,
           "seed"        123 ,
           "pobs"            ];
  < gof,best,parm,yptr > = lrallv(3,data,modl,optn,.,.,.,test);
```

The following shows the 20 "best" (w.r.t. R^2 value) 3-feature solutions of the linear regression model. The first solution is that we obtained with the forward stepwise FQP method of SVM classification:

There are 286 regression computations with 3 coefficients.

```
 1 X14  X4 X13  0.46442333
 2 X14  X4 X11  0.45916444
 3 X14  X4  X9  0.45814106
 4 X14 X10 X13  0.45483522
 5 X14  X4 X10  0.44981004
 6 X14  X9 X13  0.44659556
 7 X14  X4  X2  0.43622667
 8 X14  X4  X8  0.43613067
 9 X14  X4 X12  0.43523758
10 X14 X10  X9  0.43189788
11 X14  X4  X6  0.42862049
12 X14  X4  X5  0.42181286
13 X14  X4  X3  0.42169732
14 X14 X10 X11  0.41584017
15 X14 X13 X11  0.41504188
16 X14  X4  X7  0.41280897
17 X14  X9  X6  0.41125473
18 X14  X9  X8  0.41105434
19 X14  X9 X11  0.40921821
20 X14 X10  X2  0.40719558
```

The forward stepwise linear regression analysis also selects features X14, X4, and X13 first (based on best R^2 value):

```
print "Linear Regression : Forward first 5 vars: [4,10,11,13,14]";
modl = "1 = 2:14";
optn = [ "print"     3 ,
         "maxst"     5 ];
< gof,parm,yprd > = lrforw(data,modl,optn,.,.,.,test);
```

Forward stepwise linear regression based on the R^2 value also shows the same set of three features as the best:

```
                ************************************
                History of Forward Variable Selection
                ************************************

         (first row incremental, second row current status)

 Step  Effect  DF Rsquared    F_value    pval        SSE        BIC
```

```
1     X14   1   0.2878  84.044322  0.0000  59.644324
            2   0.2878  42.022161  0.0000  147.61282 -67.989825
2      X4   1   0.1250  44.069252  0.0000  25.909929
            3   0.4128  48.505368  0.0000  121.70289 -106.47367
3     X13   1   0.0516  19.858768  0.0000  10.700800
            4   0.4644  44.658035  0.0000  111.00209 -123.73270
4     X10   1   0.0347  14.220718  0.0002  7.2006398
            5   0.4992  40.863431  0.0000  103.80145 -135.72908
5     X11   1   0.0215  9.1619734  0.0028  4.4615187
            6   0.5207  36.935652  0.0000  99.339932 -142.84637
```

- Forward stepwise: Using FSM method:

```
print "---SVMSTW: forward stepwise with FSM, C=1 ---";
print "Found [4,10,14] is 5. best";
clas = 1;
modl = "1 = 2:14";
optn = [ "print"           2 ,
         "popt"            2 ,
         "pplan"           1 ,
         "sel"           'f' ,
         "meth"       "fsm" ,
         "nvarend"         3 ,
         "peneps"       .001 ];
c = 1.;
< alfa,sres,vres,yptr,yptt,plan > = svmstw(data,modl,optn,clas,c,test);
```

This results in a different solution which is that listed on rank 5 of the linear regressions of all combinations of 3 features.
The following shows the history of adding variables X14, X10, and X4 to the model (response y is the first of the data set):

```
       ****************************************************
       Forward Variable Selection (Ntrn=210 Nvar=13 Ntst=60)
       ****************************************************

   Iter   VAdd   NVin     Err_Trn     Err_Tst  MiscTrn  MiscTst
      1     13      1  207.257143  59.4122449       93       27
      2      9      2  147.614310  45.5662501       49       15
      3      3      3  127.937157  47.1108510       47       15
```

- Backward stepwise: Using the fast NSVM method:

```
print "---SVMSTW: backward stepwise with NSVM, C=1 ---";
print "Found [11,13,14] is 15. best";
```

```
clas = 1;
modl = "1 = 2:14";
optn = [ "print"           2 ,
         "popt"            2 ,
         "pplan"           1 ,
         "sel"           'b' ,
         "meth"       "nsvm" ,
         "nvarend"         3 ,
         "peneps"       .001 ];
c = 1.;
< alfa,sres,vres,yptr,yptt,plan > = svmstw(data,modl,optn,clas,c,test);
```

The following shows the history of dropping variables from
the model:

```
      ******************************************************
      Backward Variable Selection (Ntrn=210 Nvar=13 Ntst=60)
      ******************************************************

   Iter  NDrop   NRem       Train        Test  MiscTrn  MiscTst
      1      0     13  22.0000000  56.0000000       28       11
                      Drop:  11 X12_  27.00000000
      2      1     12  22.0000000  56.0000000       28       11
                      Drop:   4 X5_   28.00000000
      3      1     11  24.0000000  60.0000000       30       12
                      Drop:   7 X8_   29.00000000
      4      1     10  26.0000000  56.0000000       28       13
                      Drop:   1 X2_   26.00000000
      5      1      9  26.0000000  56.0000000       28       13
                      Drop:   6 X7_   27.00000000
      6      1      8  26.0000000  54.0000000       27       13
                      Drop:   2 X3_   31.00000000
      7      1      7  26.0000000  62.0000000       31       13
                      Drop:   5 X6_   32.00000000
      8      1      6  22.0000000  50.0000000       25       11
                      Drop:   8 X9_   29.00000000
      9      1      5  24.0000000  54.0000000       27       12
                      Drop:   3 X4_   36.00000000
     10      1      4  18.0000000  66.0000000       33        9
                      Drop:   9 X10_  44.00000000
     11      1      3  20.0000000  82.0000000       41       10

          Nonzeros of Separating Plane (w*x = 0.655823)
          ---------------------------------------------

                     1 10  X11_  0.630882603
```

```
                    2 12  X13_  0.489935142
                    3 13  X14_  0.408281412

                Largest 3 Plane Coefficients (Sorted)
                **************************************

                    1 10  X11_  0.630882603
                    2 12  X13_  0.489935142
                    3 13  X14_  0.408281412
             Sparsity: 3 Nonzeros 10 Zeros (C=1)
```

2. Compare svm and regression results with scad() and boruta():
   Lets have a look at the results of two competing models for
   variable selection, SCAD and BORUTA:

```
print "*** Choice of LOGLAM: [-4,0] ******************************";
d = 4. / 9.; loglam = [ -2. : d : 1. ];
lamb = 10 ** loglam; print "Lambda", lamb;
frac = [ 1.e-6 : 4.e-6 : 1.e-5 ]; print "Error Fractions=", frac;
```

   The following specifies a standard run of the "svm" version
   of the scad() function:

```
print "Solution ( X1,X4,X8,X10,X11,X14 } contains both: ";
print "( X4,X13,X14 ) is 1. best, and ( X4,X10,X14) is 5. best";
clas = 1;
optn = [ "aparm"     3.7 ,
         "initer"    200 ,
         "cdelta"         , /* constant delta */
         "minsel"      5 ,
         "xtol"    1.e-4 ,
         "ridg"       1. ,
         "techsc"  "ori" ,
         "pinit"       1 ,
         "ptab"        2 ,
         "phist"       1 ,
         "print"       2 ];
< gof,parm,gcvtab,cov,yprd > = scad("svm",data,lamb,frac,modl,optn,clas,test);
```

   With the specified λ anmd γ parameters, SCAD obtains six features:
   X1, X4, X8, X10, X13, and X14:

```
                        Parameter Estimates
                        *******************
```

```
        C |      Var_01     Var_04     Var_08     Var_10     Var_13
Estimate  |   2.52e-005  3.77e-005  9.89e-006  4.91e-005  3.97e-005

        C |      Var_14
Estimate  |   0.9999865
```

The following specifies a standard run of the boruta() function:

```
optn = [ "conf"          .999 ,
         "maxrun"          20 ,
         "rough"      "final" ,
         "print"            2 ,
         /* ranfor(): -------*/
         "seed"         12345 ,
         "ntree"          200 ,
         "mtry"            10 ,
         "ndsiz"            1 ];
< gof,bstv,stat,hist > = boruta(data,modl,optn,clas);
```

The first three features found are those which are the variables found with the "best" (w.r.t. R^2) linear regression model:

Sorted Importance Estimates of 9 Confirmed Variables

1	3	X4	1.20890885
2	13	X14	1.12886664
3	12	X13	0.93849819
4	10	X11	0.71981242
5	8	X9	0.57597607
6	9	X10	0.54260696
7	2	X3	0.34651058
8	11	X12	0.30460059
9	1	X2	0.28585032

The order of the first six features is almost the same (with the exception of feature X9 on rank 5) as obtained by the forward stepwise linear regression model based on the size of the R^2 value.

3. Feature selection with Boston Housing Data: SVM Regression
   There is a training data set with 506 observations (cases) and 14 features (effects), see test tsvm16.inp.

```
fid = fopen("..\\tdata\\housing.dat","r");
```

```
form = "%g %g %g %g %g %g %g %g %g %g %g %g %g %g";
hous = fscanf(fid,form,506,14);
vnam = [ "crim" "zn" "indus" "chas" "nox" "rm" "age"
         "dis" "rad" "tax" "ptrat" "b" "lstat" "medv" ];
hous = cname(hous,vnam);
```

- Backward stepwise: Using LSQU method:

```
print "Backward Stepwise FSM: regression: use LSQ method";
print "Uses expected fit increase when setting plan[] to zero";
print "Result: [ 6 11 13 ]=[ rm ptrat lstat ]: this is best of all 3-var solu
modl = "14 = 1:13";
optn = [ "print"           2 ,
         "pplan"           1 ,
         "sel"           'b' ,
         "meth"       "lsqu" ,
         "nvarend"         3 ];
c = 1.;
< alfa,sres,vres,yptr > = svmstw(hous,modl,optn,.,c);
```

```
                        *****************
                        Model Information
                        *****************

                  Number Valid Observations  506
                  Response Variable         Y[14]
                  N Independent Variables     13
                  Stepwise Feature Sel. Regression
                  Model Without Linear Constraint
                  Estimation Method         LSQU
                  Kernel Function         Linear
                  Use Unscaled Predictor
                  Bias Corrected Predicted Values

      Traindata stored incore: Mem need: 0.05 Mem spec: 2 Mb
         Estimated Regularization Parameter C=3.08629e-006

 The following shows the history of dropping variables from
 the model:
           ********************************************
           Backward Variable Selection (Ntrn=506 Nvar=13)
           ********************************************

        Iter  NDrop   NRem        Cold        Crit     Err_Trn
           1      0     13           .     1651.87443  22.0444523
                    Drop:   4 chas_   1640.932138
```

```
 2      1     12  1651.87443  1661.11914  22.4698471
          Drop:   3 indus_  1649.976464
 3      1     11  1661.11914  1661.13881  22.4759122
          Drop:   7 age_    1685.614047
 4      1     10  1661.13881  1661.14977  22.4723243
          Drop:   2 zn_     1702.247623
 5      1      9  1661.14977  1691.21135  22.9983897
          Drop:   1 crim_   1756.307024
 6      1      8  1691.21135  1701.85460  23.4326024
          Drop:   9 rad_    1927.348794
 7      1      7  1701.85460  1710.59633  24.1422748
          Drop:  10 tax_    1768.955537
 8      1      6  1710.59633  1710.99050  24.1324338
          Drop:  12 b_      2269.778952
 9      1      5  1710.99050  1769.91033  24.7776664
          Drop:   8 dis_    2795.161587
10      1      4  1769.91033  1840.46337  27.0834181
          Drop:   5 nox_    1976.679108
11      1      3  1840.46337  1843.53710  27.1306616
```

```
Regularization Parameter C . . . . . . . . . . . . . . . . . . 1
Kernel Function. . . . . . . . . . . . . . . . . . .        Linear
Norm of Longest Vector . . . . . . . . . . . . . . . . . 819.609
Mean Squared Error (MSE, Training Data). . . . . . . . . 27.1307
Mean Absolute Error (MAE, Training Data) . . . . . . . . 3.64335
Average Loss (Training Data) . . . . . . . . . . . . . . 3.64335
Average Positive Loss (Training Data). . . . . 3.14597 (N=293)
Average Negative Loss (Training Data). . . . . 4.32755 (N=213)
Number Fast Runs through TrainData . . . . . . . . . . . . 58
Number Slow Runs through TrainData . . . . . . . . . . . . 59
Total Number of Kernel Calls . . . . . . . . . . 1.55241e+006
Time for Optimization. . . . . . . . . . . . . . . . . . . . 1
Total Processing Time. . . . . . . . . . . . . . . . . . 37907
Optimization Criterion . . . . . . . . . . . . . . . . .    .
Geometric Margin . . . . . . . . . . 0.433122 (|w|^2= 21.3226)
Number Support Vectors . . . . . . . . . . . . . 506 (100.00 %)
Number Support Vectors on Margin . . . . . . . . . . . . . 409
Bias . . . . . . . . . . . . . . . . . . . . . . -1.8887e-015
Radius of Sphere Around SV . . . . . . . . . . . . . . . 43.2075
Estimated VCdim of Classifier. . . . . . . . . . . . . . 39807.8
Linear Kernel Constant (Fit) . . . . . . . . . . . . . . 18.7894
Linear Kernel Constant (PCE) . . . . . . . . . . . . . .    .
```

```
We end up with the three features which are also selected
for the linear regression model as the most important w.r.t.
```

R^2 :

```
            Nonzeros of Separating Plane (w*x = 18.7894)
            ----------------------------------------

                     1  6  rm_     4.486170584
                     2 11  ptrat_ -0.931686419
                     3 13  lstat_ -0.573440630

                  Largest 3 Plane Coefficients (Sorted)
                  *************************************

                     1  6  rm_     4.486170584
                     2 11  ptrat_ -0.931686419
                     3 13  lstat_ -0.573440630
                 Sparsity: 3 Nonzeros 10 Zeros (C=1)

            Total Number of Kernel Calls: 1.55241e+006
                    Time for Optimization: 1
                    Total Processing Time: 1
```

The following shows the 20 "best" (w.r.t. R^2 value) 3-feature solutions of the linear regression model. The first solution is that which we obtained with the backward stepwise LSQU method of SVM regression:

```
  print "Linear Regression : 20 best solutions for all three vars";
  print "[6,11,13] is [ rm ptrat lstat] best reg() solution of all 3-var sets";
  modl = "14 = 1:13";
  optn = [ "print"         3 ,
           "fost"             ,
           "best"          20 ,
           "selc"        "r2" ,
           "sing"     1.e-12 ,
           "seed"        123 ,
           "pobs"            ];
  < gof,best,parm,yptr > = lrallv(3,hous,modl,optn);

    There are  286 regression computations with 3 coefficients.

                   1 lstat    rm ptrat  0.67862416
                   2 lstat    rm  chas  0.65140281
                   3 lstat    rm     b  0.65055484
                   4 lstat    rm   tax  0.64851477
                   5 lstat    rm   dis  0.64678214
                   6 lstat    rm  crim  0.64585205
                   7 lstat    rm   rad  0.64278606
```

```
 8 lstat    rm indus  0.63999171
 9 lstat    rm    zn  0.63988560
10 lstat    rm   age  0.63903412
11 lstat    rm   nox  0.63891038
12 lstat ptrat   dis  0.62891180
13 lstat ptrat  chas  0.61852907
14 lstat ptrat   age  0.61601360
15 lstat ptrat   rad  0.61115537
16 lstat ptrat     b  0.60984531
17 lstat ptrat  crim  0.60716959
18 lstat ptrat    zn  0.60674158
19 lstat ptrat indus  0.60668244
20 lstat ptrat   tax  0.60666187
```

The forward stepwise linear regression analysis also selects features 13=lstat, 6=rm, and 11=ptrat first (based on best R^2 value):

```
print "Linear Regression : Forward: [ 13=lstat,6=rm,11=ptrat ], r^2=0.6786";
modl = "14 = 1:13";
optn = [ "print"     3 ,
         "maxst"     5 ];
< gof,parm,yprd > = lrforw(hous,modl,optn);
```

The following shows the history of adding variables lstat, rm, and ptrat to the model:

```
*************************************
History of Forward Variable Selection
*************************************

        (first row incremental, second row current status)

 Step  Effect  DF Rsquared    F_value    pval        SSE        BIC

    1   lstat   1   0.5441  601.61788  0.0000  23243.914
                2   0.5441  300.80894  0.0000  19472.381  1853.0250
    2      rm   1   0.0944  131.39417  0.0000  4033.0720
                3   0.6386  296.22060  0.0000  15439.309  1737.6122
    3   ptrat   1   0.0401  62.579071  0.0000  1711.3239
                4   0.6786  265.00851  0.0000  13727.985  1680.1951
    4     dis   1   0.0117  18.900871  0.0000  499.07760
                5   0.6903  223.34695  0.0000  13228.908  1663.4929
    5     nox   1   0.0178  30.457240  0.0000  759.56363
                6   0.7081  202.14209  0.0000  12469.344  1635.6166
```

- Forward stepwise: Using FQP method:

```
print "Forward Stepwise SVM regression: use LSQ method";
print "Result: [ 10 12 13 ] is [ tax b lstat ]";
modl = "14 = 1:13";
optn = [ "print"           3 ,
         "pplan"           2 ,
         "sel"           'f' ,
         "meth"        "lsq" ,
         "nvarend"         3 ];
c = [ "C"    .1  1. 10. ,
      "NU"   .1 .5 .8 ];
< alfa,sres,vres,yptr > = svmstw(hous,modl,optn,.,c);

           Grid Search Results: L1 Error (Training)
           ****************************************

                   Dense Matrix (3 by 3)

                |      NU_0.1     NU_0.5     NU_0.8
               -----------------------------------
           C_0.1 |   38.653638  40.755711  40.787525
             C_1 |   44.650818  38.280604  38.762125
            C_10 |   38.103069  38.002130  38.653638

           Grid Search Results: L2 Error (Training)
           ****************************************

                   Dense Matrix (3 by 3)

                |      NU_0.1     NU_0.5     NU_0.8
               -----------------------------------
           C_0.1 |   19558.741  20622.390  20638.488
             C_1 |   22593.314  19369.986  19613.635
            C_10 |   19280.153  19229.078  19558.741

   ******************************************************
   **********- Evaluation for Best C=0.1 Nu=0.1-**********
   ******************************************************

 Regularization Parameter C . . . . . . . . . . . . . . . . . 0.1
 Parameter NU . . . . . . . . . . . . . . . . . . . . . . . . 0.1
 Kernel Function. . . . . . . . . . . . . . . . . .        Linear
 Norm of Longest Vector . . . . . . . . . . . . . . . . . 819.609
 Mean Squared Error (MSE, Training Data). . . . . . . . . 38.6536
 Mean Absolute Error (MAE, Training Data) . . . . . . . . 4.38293
 Average Loss (Training Data) . . . . . . . . . . . . . . 4.38293
```

```
  Average Positive Loss (Training Data). . . . . 3.46526 (N=320)
  Average Negative Loss (Training Data). . . . . 5.96173 (N=186)
  Number Fast Runs through TrainData . . . . . . . . . . . . 222
  Number Slow Runs through TrainData . . . . . . . . . . . . 216
  Total Number of Kernel Calls . . . . . . . . . . 1.39926e+007
  Time for Optimization. . . . . . . . . . . . . . . . . . . 602
  Total Processing Time. . . . . . . . . . . . . . . . . . . 609
  Duality Gap. . . . . . . . . . . . . . . . . . . . . . 28.2028
  Infeasibility. . . . . . . . . . . . . . . . . . . . 0.0268819
  Geometric Margin . . . . . . . . . . 2.78711 (|w|^2= 0.514934)
  Number Support Vectors . . . . . . . . . . . . .  506 (100.00 %)
  Number Support Vectors on Margin . . . . . . . . . . . . . . 0
  Bias . . . . . . . . . . . . . . . . . . . . . . 8.92391e-015
  Radius of Sphere Around SV . . . . . . . . . . . . . . 819.609
  Estimated VCdim of Classifier. . . . . . . . . . . . .  345913
  Linear Kernel Constant (Fit) . . . . . . . . . . . . .  32.353
  Linear Kernel Constant (PCE) . . . . . . . . . . . . .   .
```

This solution does not belong to the 20 best (w.r.t R^2) of all 3-feature linear regression models (and reg() would show only R^2=.55):

```
            Nonzeros of Separating Plane (w*x = 32.353)
            -------------------------------------------

                    1 10  tax_   -0.006492927
                    2 12  b_      0.005353669
                    3 13  lstat_ -0.717539927

               Largest 3 Plane Coefficients (Sorted)
               *************************************

                     1 13  lstat_ -0.717539927
                     2 10  tax_   -0.006492927
                     3 12  b_      0.005353669
               Sparsity: 3 Nonzeros 10 Zeros (C=0.1)

            Total Number of Kernel Calls: 1.39926e+007
                   Time for Optimization: 602
                   Total Processing Time: 609
```

```
  print "Forward Stepwise SVM regression: use FLP method: CLP";
  print "Result: [ 10 12 13 ] is [ tax b lstat ]";
  modl = "14 = 1:13";
  optn = [ "print"          3 ,
           "pplan"          2 ,
```

```
        "sel"           'f' ,
        "meth"        "flp" ,
        "modif"         2 ,
        "nvarend"       3 ];
c = [ "C"     .1  1. 10. ,
      "NU"    .1 .5 .8 ];
< alfa,sres,vres,yptr > = svmstw(hous,modl,optn,.,c);

           Grid Search Results: L1 Error (Training)
           ****************************************

                   Dense Matrix (3 by 3)

                |      NU_0.1     NU_0.5     NU_0.8
               -------------------------------------
             C_0.1 |   66.987930  67.071160  66.987930
               C_1 |   77.402248  67.071160  66.987930
              C_10 |   77.402248  67.071160  66.987930

           Grid Search Results: L2 Error (Training)
           ****************************************

                   Dense Matrix (3 by 3)

                |      NU_0.1     NU_0.5     NU_0.8
               -------------------------------------
             C_0.1 |   33895.892  33938.007  33895.892
               C_1 |   39165.537  33938.007  33895.892
              C_10 |   39165.537  33938.007  33895.892


    ********************************************************
    **********- Evaluation for Best C=0.1 Nu=0.1-**********
    ********************************************************

 Regularization Parameter C . . . . . . . . . . . . . . . . 0.1
 Parameter NU . . . . . . . . . . . . . . . . . . . . . . . 0.1
 Kernel Function. . . . . . . . . . . . . . . . . . .       Linear
 Norm of Longest Vector . . . . . . . . . . . . . . . . 819.609
 Mean Squared Error (MSE, Training Data). . . . . . . . 66.9879
 Mean Absolute Error (MAE, Training Data) . . . . . . . 5.98436
 Average Loss (Training Data) . . . . . . . . . . . . . 5.98436
 Average Positive Loss (Training Data). . . . . 4.81865 (N=330)
 Average Negative Loss (Training Data). . . . . 8.17007 (N=176)
 Number Fast Runs through TrainData . . . . . . . . . . . . 237
 Number Slow Runs through TrainData . . . . . . . . . . . . 231
```

```
Total Number of Kernel Calls . . . . . 1.39644e+007
Time for Optimization. . . . . 57
Total Processing Time. . . . . 69
Duality Gap. . . . . 0
Geometric Margin . . . . . 98.9764 (|w|^2= 0.000408316)
Number Support Vectors . . . . . 1 (  0.20 %)
Number Support Vectors on Margin . . . . . 0
Bias . . . . . 0
Radius of Sphere Around SV . . . . . 676.207
Estimated VCdim of Classifier. . . . . 187.705
Linear Kernel Constant (Fit) . . . . . 31.3882
Linear Kernel Constant (PCE) . . . . . .

         Nonzeros of Separating Plane (w*x = 31.3882)
         --------------------------------------------

                1 10  tax_   -0.020161205
                2 12  b_     -0.000871533
                3 13  lstat_ -0.001040451

            Largest 3 Plane Coefficients (Sorted)
            *************************************

                1 10  tax_   -0.020161205
                2 13  lstat_ -0.001040451
                3 12  b_     -0.000871533
            Sparsity: 3 Nonzeros 10 Zeros (C=0.1)
```

- Compare svm and regression results with scad() and boruta():

```
d = 2. / 9.; loglam = [ -1. : d : 1. ];
lamb = 10 ** loglam; print "Lambda", lamb;
frac = [ .01 : .01 : .1 ]; print "Error Fractions=", frac;

print "Order of the largest 5 parameters: [ rm,chas,dis,lstat,ptrat ]: ";
print "[ rm,chas,dis,lstat,ptrat ] = [ 6,4,8,13,11 ] ";
modl = "14 = 1:13";
print "Use Sherman-Morrison-Woodbury";
optn = [ "aparm"     3.7 ,
         "initer"    200 ,
         "xtol"    1.e-4 ,
         "ridg"       1. ,
         "techsc"  "ori" ,
         "pinit"       2 ,
         "ptab"        2 ,
         "phist"       2 ,
```

```
          "print"       3 ];
   < gof,parm,gcvtab,cov,yprd > = scad("lse",hous,lamb,frac,modl,optn);
   print "GOF=",gof;
```

The result is a dense solution with almost all parameters significant (except nox and indus) and the order of the largest parameters is (rm,chas,dis,lstat,ptrat) = (6,4,8,13,11).

Parameter Estimates
****

	Estimate	AsStdErr	LowerCI	UpperCI
crim	-0.0730211	0.0058831	-0.0845518	-0.0614904
zn	0.0422025	0.0024905	0.0373211	0.0470838
indus	-4.81e-006	3.16e-006	-1.10e-005	1.38e-006
chas	2.9727434	0.1788811	2.6221429	3.3233438
nox	0	6.07e-019	-1.19e-018	1.19e-018
rm	5.6252553	0.0473942	5.5323643	5.7181463
age	-0.0054530	0.0019533	-0.0092813	-0.0016246
dis	-0.7364158	0.0259350	-0.7872474	-0.6855842
rad	0.1401021	0.0098916	0.1207149	0.1594894
tax	-0.0094327	5.59e-004	-0.0105284	-0.0083370
ptrat	-0.3679123	0.0161606	-0.3995865	-0.3362382
b	0.0133784	5.21e-004	0.0123581	0.0143987
lstat	-0.4412160	0.0084880	-0.4578521	-0.4245798

The following specifies a run of the boruta() function:

```
   modl = "14 = 1:13";
   optn = [ "conf"          .999 ,
            "maxrun"          20 ,
            "rough"      "final" ,
            "print"             2 ,
            /* ranfor(): -------*/
            "seed"         12345 ,
            "ntree"          200 ,
            "mtry"            10 ,
            "ndsiz"            1 ];
   < gof,bstv,stat,hist > = boruta(hous,modl,optn);
```

Except for nox the four features with largest importance scores agree with those found by other methods:

Sorted Importance Estimates of 12 Confirmed Variables

```
                    1       6       rm  19.6765933
```

2	13	lstat	16.7894387
3	5	nox	7.60697227
4	11	ptrat	6.18957571
5	1	crim	6.15471509
6	8	dis	5.87997675
7	3	indus	5.67405142
8	10	tax	4.64851333
9	7	age	4.17130017
10	9	rad	2.89548779
11	4	chas	2.22165399
12	12	b	1.85223974

But, different models and methods for variable selection may find also different important features especially when they become more and more unimportant.

5.11 The warning() Function

```
warning(message)
```

Purpose: The warning() function prints an warning message mess into the log output file.

Input: The input must be a string. The string may be generated by a call of function sprintf().

Output: Generates an warning message in the log file.

Restrictions: 1. The type of the only argument must be string.

Relationships: error()

Examples:

```
if (a == .) warning("Variable must not be missing.");
```

6 Illustrations

6.1 Three Different Model Specifications with glim()

The following three specifications of a specific logistic regression model yield to an essentially identical result:

```
print "[1.1] Common Logistic without FREQ";
data = [ 1  0  ,  1  0  ,  1  0  ,  1  0  ,  1  0  ,
         0  0  ,  0  0  ,  0  0  ,
         1  1  ,
         0  1  ,  0  1  ,  0  1  ,  0  1  ] ;
cnam = [" cancer treat "];
data = cname(data,cnam);


optn = [ "print"          5 ,
         "link"      "logit" ,
         "order"       "des" ,
         "dist"      "binom" ,
         "tech"     "trureg" ];
clas = 1;
modl = "1 = 2";
< gof,parm,sterr,conf > = glim(data,modl,optn,clas);
```

Practically the same results are obtained by using the a freq variable in the data set:

```
print "[1.2] Common Logistic with FREQ variable";
data = [ 1  0   5 ,
         0  0   3 ,
         1  1   1 ,
         0  1   4 ] ;
cnam = [" cancer treat num "];
data = cname(data,cnam);


optn = [ "print"          5 ,
         "link"      "logit" ,
         "order"       "des" ,
         "dist"      "binom" ,
         "freq"            3 ,
         "tech"     "trureg" ];
clas = 1;
modl = "1 = 2";
< gof,parm,sterr,conf > = glim(data,modl,optn,clas);
```

The shortest specification of the same model has only two observations in the data set but refers to the *events-trial* response columns of the data set. Note, that the old specification "trial" of the column number of a trial variable in a model statement is no longer needed:

```
print "[1.3] Common Logistic Event/Trial: trial spec not needed anymore ";
data = [ 5  8  0  ,
         1  5  1 ];
cnam = [" cancer n treat "];
data = cname(data,cnam);


optn = [ "print"           5 ,
         "link"      "logit" ,
         "order"       "des" ,
         "dist"      "binom" ,
    /*   "trial"           2 , not needed anymore */
         "tech"     "trureg" ];
modl = "1/2 = 3";
< gof,parm,sterr,conf > = glim(data,modl,optn);
```

The results of the three models are basically the same and we list here only those of the last run:

```
                      *****************
                      Model Information
                      *****************

                Number Valid Observations    2
                Events/Trial Resp      [1]/[2]
                N Independent Variables      1
                Error Distribution    BINOMIAL
                Link Function            LOGIT

                Trial Variable Column        2
                Significance Level:  0.0500000
                Design Coding:       Full-Rank
                Hessian Matx. for Optimization
                Hessian Matx. for Covariance M
                No Variable Selection Process


                       *************
                       Model Effects
                       *************
```

Intercept + X3

****
Simple Statistics
****

Column	Nobs	Mean	Std Dev	Skewness	Kurtosis
Y[1]	2	0.4615385	0.5188745	0.1752044	-2.3636364
X[3]	2	0.3846154	0.5063697	0.5385932	-2.0563636

****
Parameter Information
****

Parameter	Meaning
1	Intercept
2	treat

****
Events/Trials Response Variable
****

Value	Nobs	Proportion
Event	6	46.153846
NonEvent	7	53.846154

****
Goodness of Model Fit
****

Log Likelihood	-7.7945180	Degrees of Freedom	11
Deviance	25.950102	Value /df	2.3591002
Pearson ChiSquare	26.383327	Value /df	2.3984842
SSE	2.6750000	MSE = SSE/nobs	0.2057692
AIC (Intercept)	19.944828	AIC (All Param.)	19.589036
SBC (Intercept)	20.509777	SBC (All Param.)	20.718935
-2logL (Intercept)	17.944828	-2logL (All Param.)	15.589036
Likelihood Ratio	2.3557915	Pvalue (df=1)	0.1248185
Score ChiSqu. Test	2.2363095	Pvalue (df=1)	0.1348024
Wald Test	2.0181663	Pvalue (df=1)	0.1554267

```
        ****************************************
        Analysis of Effects and Parameter Estimates
        ****************************************

 Parameter        DF  Estimate Std_Error WaldChiSq Pr>ChiSq

 Intercept         1  0.510826  0.730297 0.4892678 0.484254
 treat             1 -1.897120  1.335415 2.0181663 0.155427
 Scale             0  1.000000         0     .        .

     Covariance Matrix and ASE Based on Hessian Matrix


              ******************************
              Confidence Limits of Parameters
              ******************************

    Parameter          Estimate   LowWaldCL   UppWaldCL

    Intercept         0.5108256 -0.92052969  1.94218094
    treat            -1.8971196 -4.51448470  0.72024560

     Wald Confidence Intervals Based on Hessian Matrix


           ************************************
           Odds Ratio and Standardized Estimates
           ************************************

Parameter        OddsRatio   StndEst     LowerCL     UpperCL

treat            0.1500001 -0.529631  0.01094925  2.05493783


              ******************************
              Evaluation of Training Data Fit
              ******************************

  Index                              Value       StdErr
  Absolute Classification Error          4      .
  Classification Accuracy        69.23076923      .
  Percent Concordant Pairs       47.61904762      .
  Percent Discordant Pairs       7.142857143      .
  Percent Tied Pairs             45.23809524      .
  c                              0.702380952      .
  Goodman-Kruskal Gamma          0.739130435  0.302929688
```

```
Kendall Tau_a                  0.217948718      .
Kendall Tau_b                  0.414757531  0.242250204
Stuart Tau_c                   0.402366864  0.240300132
Somers D (Gini Coef.)          0.404761905      .
Somers D C|R                   0.404761905  0.241109044
Somers D R|C                   0.425000000  0.247582057
```

Classification Table

Observed	Predicted NoEvt	Predicted Event
NoEvt	4	3
Event	1	5

Area under the ROC Curve (Disc.) = 0.702381
Area under the ROC Curve (Cont.) = 0.702381

Nobs	Yobs	Ypred	LowerCL	UpperCL	Diag(H)
1	5	0.62500000	0.28484998	0.87459155	1.00000000
2	1	0.20000007	0.02718311	0.69104560	1.00000000

SSE=2.675 MSE=0.205769 Misclassification=4

Pearson ChiSquare=26.3833 Deviance=25.9501 (df=11)

****
LR Statistics for Type 1 Analysis
****

Effect	Deviance	DF	ChiSquare	Pr>Chi
Intercept	18.25312894	0	.	.
treat	25.95010211	1	2.35579153	0.1248

****
LR Statistics for Type 3 Analysis
****

Effect	DF	ChiSquare	Pr>Chi

```
            treat                1  2.35579153  0.1248
```

This model is extensively applied for analyzing data of my wife's cancer research.

6.2 Some Univariate Outlier Tests

The following is the CMAT code for some functionality of the outlier package in R. Thompson's τ is also computed in Matlab. Compared to the R package the advantage here is that Dixon's Q test is using McBane's (2006) more exact critical values than the table interpolation used in the R package.

```
WT_0h = [ 157.892500 265.978400 102.106300 192.758900 129.999400 157.892500 123.026100 ]‘;
print "Sorted WT_0h:", WT_0h = WT_0h[<];


  set = 1; data = WT_0h; n = nrow(data);
  sopt = [ "ari" "std" "mad" "qua" ];
  print mom = univar(data,sopt);


  print "Chi-Square Test, see Dixon (1950)";
  ari = mom[1]; var = mom[2] * mom[2];
  if (data[n] - ari < ari - data[1]) {
    chi = (ari - data[1])**2 / var;
    prb = chimis(.,chi,1);
    print "Chi Square Test: Lower Outlier:", data[1], " prob=", prb;
  } else {
    chi = (data[n] - ari)**2 / var;
    prb = chimis(.,chi,1);
    print "Chi Square Test: Upper Outlier:", data[n], " prob=", prb;
  }


  print "Common 3-Sigma Rule";
  ari = mom[1]; std = mom[2]; diff = 1.5 * std;
  low = ari - diff; upp = ari + diff;
  print "3-Sigma: Set=", set, " [low,upp]=", low,upp, " diff=", diff;
  if (data[1] <= low)
  print "3-Sigma: Set=", set," Lower Outlier:", data[1], " Diff=", low - data[1];
  if (data[n] >= upp)
  print "3-Sigma: Set=", set," Upper Outlier:", data[n], " Diff=", data[n] - upp;


  print "Robust 3-MAD Rule";
```

```
med = mom[3]; mad = mom[4]; diff = 1.5 * mad;
low = med - diff; upp = med + diff;
print "3-MAD: [low,upp]=", low,upp, " diff=", diff;
if (data[1] <= low)
print "3-MAD: Set=", set," Lower Outlier:", data[1], " Diff=", low - data[1];
if (data[n] >= upp)
print "3-MAD: Set=", set," Upper Outlier:", data[n], " Diff=", data[n] - upp;


print "Tukey Test";
q1 = mom[5]; q3 = mom[6]; diff = 1.5 * (q3 - q1);
low = q1 - diff; upp = q3 + diff;
print "Tukey: Set=", set, " [low,upp]=", low,upp, " diff=", diff;
if (data[1] <= low)
print "Tukey: Set=", set," Lower Outlier:", data[1], " Diff=", low - data[1];
if (data[n] >= upp)
print "Tukey: Set=", set," Upper Outlier:", data[n], " Diff=", data[n] - upp;


print "Chauvenet\'s Criterion";
ari = mom[1]; std = mom[2];
t = abs(ari - data[1]) / std;
prb = 2. * n * (1. - normis(.,t,0.,1.));
if (prb < .5)
print "Chauvenet: Set=", set, " Lower Outlier:", data[1], " Crit=", prb;
t = abs(data[n] - ari) / std;
prb = 2. * n * (1. - normis(.,t,0.,1.));
if (prb < .5)
print "Chauvenet: Set=", set, " Upper Outlier:", data[n], " Crit=", prb;


print "Grubbs Test";
df = n - 2; alfa = .05; prb = alfa / (2. * n);
tq = tmis(prb,.,df); rat = (n - 1.) / sqrt(n);
crt = rat * sqrt(tq * tq / (n - 2. + tq * tq));
ari = mom[1]; std = mom[2];
gmin = (ari - data[1]) / std;
if (gmin > crt)
print "Grubbs: Set=", set, " Lower Outlier:", data[1], " Crit=", gmin;
gmax = (data[n] - ari) / std;
if (gmax > crt)
print "Grubbs: Set=", set, " Upper Outlier:", data[n], " Crit=", gmax;


print "Thompson\'s Tau Test";
ari = mom[1]; std = mom[2];
alpha = .05; a2 = 1. - .5 * alpha; df = n - 2;
```

```
  ta = tmis(a2,.,df);
  tau = ta * (n - 1.) / (sqrt(n) * sqrt((n - 2 + ta *ta)));
  /* print "n, tau=", n, tau; */
  tlow = (ari - data[1]) / std;
  if (tlow > tau)
  print "Thompson: Set=", set," Lower Outlier:", data[1], " Diff=", tlow-tau;
  tupp = (data[n] - ari) / std;
  if (tupp > tau)
  print "Thompson: Set=", set," Upper Outlier:", data[n], " Diff=", tupp-tau;


  print "Dixon Q type=10 : i=j=1 Test";
  /* For two-sided test we must divide alpha by 2 */
  j = i = 1; alpha = .05 / 2.;
  opt = [ 0,    /* ipri*/
          0,    /* minmet */
        200 ];  /* maxf */
  qcrt = dixonr("crit",i,j,n,alpha,opt);
  qlow = (data[2] - data[1]) / (data[n] - data[1]);
  if (qlow > qcrt)
  print "Dixon Set=", set," Lower Outlier:", data[1], " Crit=", qlow;
  qupp = (data[n] - data[n-1]) / (data[n] - data[1]);
  if (qupp > qcrt)
  print "Dixon Set=", set," Upper Outlier:", data[n], " Crit=", qupp;
```

Together with the following 5 data sets

```
KO_0h = [ 133.486000 95.132980 143.946000 133.486000 147.432600 196.245600 136.972700 150.91
print "Sorted KO_0h:", KO_0h = KO_0h[<];
WT_2h = [ 164.865800 98.619620 367.091000 248.545200 335.711200 185.785600 921.466900 ]‘;
print "Sorted WT_2h:", WT_2h = WT_2h[<];
KO_2h = [ 447.283700 265.978400 461.230300 597.209300 684.375300 234.598600 991.199700 255.5
print "Sorted KO_2h:", KO_2h = KO_2h[<];
WT_4h = [ 203.218800 293.871500 771.541300 252.031800 286.898200 597.209300 2176.658000 ]‘;
print "Sorted WT_4h:", WT_4h = WT_4h[<];
KO_4h = [ 740.161600 656.482200 855.220700 1169.018000 1165.532000 210.192100 2072.058000 74
print "Sorted KO_4h:", KO_4h = KO_4h[<];
```

we obtain the following table of outlier occurences:

	WT_0h		KO_0h		WT_2h		KO_2h		WT_4h		K4_0h	
	Low	Upp	Low	Upp	Low	Upp	Low	Upp	Low	Upp	Low	Upp
Chi Square Test		*		*		*		*		*		*
3-Sigma Rule		*	*	*		*		*		*		*
3-MAD Rule	*	*	*	*	*	*		*		*	*	*

Tukey Test	*	*	*	*		*	*
Chauvenet	*		*	*	*	*	*
Grubbs Test				*		*	
Thompson Test	*		*	*	*	*	*
Dixon Q Test				*		*	

6.3 Some Data Sets for svm

Response	Name	Nobs	Nvar	File
interval	CACO	27	714	tsvm7, tsvm9, tsvm23
	NIR	21 (7)	268	tsvm8, tsvm9, tsvm22
	mysvm	100	11	tsvm20, tsvm21
	BostonHousing	506	14	tsvm16, tsvm20, tsvm23
binary	Heart	270	13	tsvm2
	Heart	210 (60)	13	tsvm7
	Diabetes	768	8	tsvm5
	Australian	690	14	tsvm6, tsvm17, tsvm24
	Galaxy	4192	14	tsvm13
	BUPA Liver	345	6	tsvm13
	Ionosphere	351	34	tsvm13
	Pima Diabetes	768	8	tsvm13
	German	1000	20	tsvm13
	Myeloma Affym.	105	7009	tmicro\taffym0
	Myeloma Affym.	105	28033	tmicro\taffym0
	Ulrich Affym.	37	22283	tmicro\taffym2
multinomial	Iris (3 cat)	130	4	tsvm1,tsvm10,tsvm12
	LungCancer (3)	27	56	tsvm8, tsmsvm3
	Roche (18 cat)	719 (190)	29501	tmicro\roche
ordinal	BreastCancer	98	24189	tmicro\vanVeer

For Nobs the number in parenthesis is the number of observations in a test set.

Names of methods implemented in svm() function:

Method	Classification	Regression	Reference
FLP	n	y	Bi, Bennett, Embrechts, Breneman & Song (2002): LP
FQP	y	y	full QP with null or range space methods
FQP	y	y	penalty method for solving QP
DQP	y	y	T. Joachims (1999): decomposition methods for QP
SMO	y	y	J. Platt (1998), Lin & Chen, Keerthi: SMO Algorithm
LCH	y	n	Mangasarian & Thompson (2006): linear chnunking
ASVM	y	n	Mangasarian & Musicant (2000): active set SVM
LSVM	y	n	Mangasarian & Musicant (2000): Lagrangian SVM
PSVM	y	n	Fung & Mangasarian (2000) :
RSVM	y	n	Lee & Mangasarian (1999): smooth and reduced SVM
LSQU	y	y	Suykens & Vandevalle (1999) : least squres SVM
FSM	y	n	Fung & Mangasarian (2003) :
NSVM	y	n	Fung & Mangasarian (2002) :

Method	Classification		Regression		
	C Class.	ν Class.	L2 Regr.	ν Regr.	ϵ Regr.
FLP	bug	fqp	fqp	y	y
FQP	y	y	y	y	y
FQP	y	y	y	y	y
DQP	y	fqp	bug	smo	smo
SMO	y	y	y	y	y
LCH	fqp	fqp	dqp	fqp	fqp
ASVM	y	fqp	fqp	fqp	fqp
LSVM	y	fqp	fqp	fqp	fqp
PSVM	y	fqp	fqp	fqp	fqp
RSVM	bug	fqp	fqp	fqp	fqp
LSQU	y	fqp	y	fqp	fqp
NSVM	fqp	fqp	dqp	fqp	fqp
FSM	fqp	fqp	dqp	fqp	fqp

7 The Bibliography

References

[1] Abebe, A., Daniels, J., McKean, J.W., & Kapenga, J.A. (2001), *Statistics and Data Analysis*, http://www.stat.wmich.edu/s160/book/

[2] Bi, J., Bennett, K.P., Embrechts, M., Breneman, C.M., & Song, M. (2002), ‘‘Dimensionality reduction via sparse support vector machines’’; *Journal of Machine Learning*, **1**, 1-48.

[3] Chauvenet, W. (1863), *A Manual of Spherical and Practical Astronomy*, Dover N.Y. 1960, 474-566.

[4] Christensen, R., Pearson, L.M., & Johnson, W. (1992), ‘‘Case deletion diagnostics for mixed models’’, *Technometrics*, **34**, 38-45.

[5] Czyzyk, J., Mehrotra, S., Wagner, M. & Wright, S.J. (1997) ‘‘PCx User Guide (Version 1.1)’’, Technical Report OTC 96/01, Office of Computational and Technology Research, US Dept. of Energy.

[6] Dixon, W. J. (1950), ‘‘Analysis of extreme values’’, *The Annals of Mathematical Statistics*, **21**, 488-506.

[7] Forrest, J. J. & Lougee-Helmer, R. (2014), *Cbc*, jjforre@us.ibm.com, robinlh@us.ibm.com.

[8] Forrest, J. J., de la Nuez, D., & Lougee-Helmer, R. (2014), *Clp*, http://www.tsp.gatech.edu/concorde.

[9] Fung, G. & Mangasarian, O.L. (2002), ‘‘Finite Newton method for Lagrangian support vector machine classification’’; Technical Report 02-01, Data Mining Institute, Computer Sciences Dep., Univ. of Wisconsin, Madison, Wisconsin.

[10] Fung, G. & Mangasarian, O.L. (2003), ‘‘A Feature Selection Newton Method for Support Vector Machine Classification’’, *Computational Optimization and Applications*, 1-18.

[11] Grubbs, F. E. (1969), ‘‘Procedures for detecting outlying observations in samples’’, *Technometrics*, **11**, 1-21.

[12] Guyon, I., & Elisseeff, A. (2003), ‘‘An introduction to variable and feature selection’’, *Journal of Machine Learning Research*, **3**, 1157-1182.

[13] Kursa, M.B. & Rudnicki, W.R. (2010), ‘‘Feature Selecion with the Boruta Package’’; *JSS*, 2010.

[14] Lee, Y.-Y. & Mangasarian, O. (2000), *RSVM: Reduced support vector machines*, Technical Report 00-07, Computer Sciences Dept., University of Wisconsin, Madison.

[15] Lee, Y., Lin, Y., & Wahba, G. (2003), *Multicategory Support Vector Machines, Theory and Application to the Classification of Microarray Data and Satellite Radiance Data*, Technical Report 1064, Dep. of Statistics, Univ. of Wisconsin, 2003.

[16] ‘‘MPS file format’’, http://lpsolve.sourceforge.net/5.5/mps-format.htm

[17] Madsen, K., Nielsen, H.B., & Pinar, M.C. (1996), ‘‘A new finite continuation algorithm for linear programming’’, *SIAM Journal on Optimization*, **6**, 600-616.

[18] Mangasarian, O.L. & Musicant, D.R (2000a), ‘‘Active Support Vector Machine Classification’’, Technical Report 00-04, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[19] Mangasarian, O.L. & Musicant, D.R (2000b), ‘‘Lagrangian Support Vector Machines’’, Technical Report 00-06, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[20] Mangasarian, O.L. & Thompson, M.E. (2006), ‘‘Massive data classification via unconstrained support vector machines’’, *Journal of Optimization Theory and Applications*. Technical Report 06-07, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[21] Mangasarian, O.L. & Thompson, M.E. (2006), ‘‘Chunking for massive nonlinear kernel classification’’, Technical Report 06-07, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[22] McBane, G.C. (2006), ‘‘Programs to compute distribution functions and critical values for extreme value ratios for outlier detection’’; *JSS*, 2006.

[23] Nilsson, R., Pẽna, J., Björkegren, J., & Tegnér, J. (2007), ‘‘ Consistent feature selection for pattern recognition in polynomial time’’, *Journal of Machine Learning Research*, **8**, 612.

[24] Platt, J. (1999), ‘‘Sequential Minimal Optimization: A fast algorithm for training support vector machines’’, in B. Schölkopf, C.J.C. Burges, and A.J. Smola (eds), *Advances in Kernel Methods: Support Vector Learning*, Cambridge: MIT Press.

[25] Rorabacher, D.B. (1991), ‘‘Statistical treatment for rejection of deviant values: Critical values of Dixon Q parameter and related subrange ratios at the 95 percent confidence level’’, *Analytical Chemistry*, **63**, 139-146.

[26] Stoppiglia, H., Dreyfus, G. Dubois, R.& Oussar, Y. (2003), ‘‘Ranking a random feature for variable and feature selection’’, *Journal of Machine Learning Research*, **3**, 1399-1414.

[27] Suykens, J.A.K. & Vandevalle, J. (1999), ‘‘Least squares support vector classifiers’’, *Neural Processing Letters*, **9**, 293-300.

[28] Thompson, R. (1985), ‘‘A note on restricted maximum likelihood estimation with an alternative outlier model’’; *Journal of the Royal Statistical Society*, Ser. B, **47**, 53-55.

[29] Tukey, J.W. (1977b), *Exploratory Data Analysis*, Reading: Addison-Wesley.