Olvi's Matlab Algorithms in CMAT©

Wolfgang M. Hartmann

March 2007

Contents

1 Introduction

This paper shows some Matlab code listings by Olvi Mangasarian and his students D. Musicant, G. Fung, and Y.-J. Lee which were taken from Olvi's website `www.csi.wisc.edu/~olvi`. For better readability some line breaks have been included into the Matlab code without the `...` token which is needed from Matlab to understand that the line breaks does not indicate the end of the statement. People who do want to run these Matlab programs are therefore recommended to download the original scripts from Olvi's website and not to type in the printed version shown in this paper.

In addition to the Matlab codes by Olvi and his students we also show some Matlab code for SCAD SVM from Helen Zhang's website `www4.stat.ncsu.edu/~hzhang`. and a slightly improved version of CMAT code which contains a modification for running the code when there are many variables. The original Zhang code will run out of core memory with many PCs when there are more than 10,000 variables, since storing matrix $\mathbf{Q}$ with 10,000 rows and columns would require $10,000 * 10,000 * 8 = 800,000,000$ bytes in double precision. Of course a similar algorithm is available with the `scad` function inside of CMAT.

For better understanding it should be mentioned that the parameter $\nu > 0$ in Olvi's language refers to the in SVM literature commonly used parameter $C > 0$, where the name ν is used for a different parameter used in SVM ν regression and classifcation (see Schoelkopf and Smola) and which is bounded in the interval $(0, 1]$. Also, most papers and software of SVM use the opposite sign for the intercept parameter γ of the linear SVM model.

Note, that the PSVM and LSVM code listings here refer to a copyright by Musicant and Mangasarian but not to the fact that the patent for LSVM was sold to SAS Institute. Since the algorithm of PSVM is only the first part of the LSVM algorithm, this patent by SAS Institute may also indirectly cover the algorithm of PSVM.

For each of the Matlab implementations (except that of the nonlinear kernel version of LSVM) an implementation in CMAT language was written and tested. In some respects, like the Armijo step size algorithm, the CMAT code slightly differs from the Matlab code. In addition to the CMAT language code listings also some input for the `svm` function is provided which shows that similar results can be obtained with this computationally more efficient function provided by CMAT. It should be mentioned here that also `PROC SVM` in *SAS Enterprise Miner* software includes most of these algorithms.

2 PSVM: Proximal SVM

2.1 Olvi's Matlab Code

The following Matlab code can be found at Olvi Mangasarian's website. In addition to the PSVM implementation it contains a number of other features, like two ways for estimating the regularization parameter ν (which is usually known as C) and some fit evaluation.

```
function [w,gamma, trainCorr, testCorr, cpu_time, nu] =
          psvm(A,d,k,nu,output,bal);
% version 1.1
% last revision: 01/24/03
%=====================================================================
```

```
% Usage:    [w,gamma,trainCorr, testCorr,cpu_time,nu]=
%                psvm(A,d,k,nu,output,bal)
%
% A and d are both required, everything else has a default
% An example: [w gamma train test time nu] = psvm(A,d,10);
%
% Input:
% A is a matrix containing m data in n dimensions each.
% d is a m dimensional vector of 1's or -1's containing
% the corresponding labels for each example in A.
% k is k-fold for correctness purpose
% nu - the weighting factor.
%                     -1  - easy estimation
%                      0  - hard estimation
%                      any other value - used as nu by the algorithm
%                      default - 0
% output - indicates whether you want output
%
% If the input parameter bal is 1
% the algorithm weighs the classes depending on the
% number of points in each class and balance them.
% It is useful when  the number of point in each class
% is very unbalanced.
%
% Output:
% w,gamma are the values defining the separating
% Hyperplane w'x-gamma=0 such that:
%
% w'x-gamma>0 => x belongs to A+
% w'x-gamma<0  => x belongs to A-
% w'x-gamma=0 => x can belongs to both classes
% nu - the estimated or specified value of nu
%
% For details refer to the paper:
% "Proximal Support Vector Machine Classifiers"
% available at: www.cs.wisc.edu/~gfung
% For questions or suggestions, please email:
% Glenn Fung, gfung@cs.wisc.edu
% Sept 2001.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;
[m,n]=size(A);
r=randperm(size(d,1));d=d(r,:);A=A(r,:);    % random permutation

%move one point in A a little if perfectly balanced
AA=A;dd=d;
```

```
ma=A(find(d==1),:); mb=A(find(d==-1),:);
[s1 s2]=size(ma);
     c1=sum(ma)/s1;
[s1 s2]=size(mb);
     c2=sum(mb)/s1;
if (c1==c2)
     nu=1;
     A(3,:)=A(3,:)+0.01*norm(A(3,:)-c1,inf)*ones(1,n);
end

% default values for input parameters
if nargin<6
   bal=0;
end
if nargin<5
   output=0;
end
if (nargin<4)
    nu=0;  % default is hard estimation
end
if (nargin<3)
     k=0;
end

    [H,v]=HV(A,d,bal);  % calculate H and v

trainCorr = 0;
testCorr = 0;

if (nu==0)
  nu = EstNuLong(H,d,m);
elseif nu==-1  % easy estimation
  nu = EstNuShort(H,d);
end

% if k=0 no correctness is calculated, just run the algorithm
if k==0
  [w, gamma] = core(H,v,nu,n);
cpu_time = toc;
  return
end

%if k==1 only training set correctness is calculated
if k==1
  [w, gamma] = core(H,v,nu,n);
  trainCorr = correctness(AA,dd,w,gamma);
```

```
  cpu_time = toc;
  if output == 1
  fprintf(1,'\nTraining set correctness: %3.2f%% \n',trainCorr);
  fprintf(1,'\nElapse time: %10.2f\n',toc);
  end
  return
end

  [sm sn]=size(A);
accuIter = 0;
lastToc=0;    % used for calculating time
indx = [0:k];
indx = floor(sm*indx/k);    %last row numbers for all 'segments'
% split trainining set from test set
tic;
for i = 1:k
Ctest = []; dtest = [];Ctrain = []; dtrain = [];

Ctest = A((indx(i)+1:indx(i+1)),:);
dtest = d(indx(i)+1:indx(i+1));

Ctrain = A(1:indx(i),:);
Ctrain = [Ctrain;A(indx(i+1)+1:sm,:)];
dtrain = [d(1:indx(i));d(indx(i+1)+1:sm,:)];

   [H, v] = HV(Ctrain,dtrain,bal);
   [w, gamma] = core(H,v,nu,n);

   tmpTrainCorr = correctness(Ctrain,dtrain,w,gamma);
   trainCorr = trainCorr + tmpTrainCorr;
   tmpTestCorr = correctness(Ctest,dtest,w,gamma);
   testCorr = testCorr + tmpTestCorr;

 if output==1
 fprintf(1,'_____________________________________________\n');
 fprintf(1,'Fold %d\n',i);
 fprintf(1,'Training set correctness: %3.2f%%\n',tmpTrainCorr);
 fprintf(1,'Testing set correctness: %3.2f%%\n',tmpTestCorr);
 fprintf(1,'Elapse time: %10.2f\n',toc);
 end

 end %

 trainCorr = trainCorr/k;
 testCorr = testCorr/k;
 cpu_time = toc/k;
```

```
if output == 1
  fprintf(1,'_____________________________________________________\n');
  fprintf(1,'\nTraining set correctness: %3.2f%% \n',trainCorr);
  fprintf(1,'\nTesting set correctness: %3.2f%% \n',testCorr);
  fprintf(1,'\nAverage CPU time is: %3.2f \n',cpu_time);
end
%y=spdiags(d,0,m,m)*((A*w-gamma)-ones(m,1));
return

%%%%%%%%%%%%%%% core function to calcuate w and gamma %%%%%%%%
function [w, gamma]=core(H,v,nu,n)

v=(speye(n+1)/nu+H'*H)\v;
w=v(1:n);gamma=v(n+1);

return

%%%%%%%%%%%%%%% correctness calculation %%%%%%%%%%%%%%%%%%%%
function corr = correctness(AA,dd,w,gamma)

p=sign(AA*w-gamma);
corr=length(find(p==dd))/size(AA,1)*100;

return

%%%%%%%%%%%%       EstNuLong     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% use to estimate nu
function lamda=EstNuLong(H,d,m)

if m<201
H2=H;d2=d;
else
r=rand(m,1);
 [s1,s2]=sort(r);
 H2=H(s2(1:200),:);
 d2=d(s2(1:200));
end

lamda=1;
[vu,u]=eig(H2*H2');u=diag(u);p=length(u);
yt=d2'*vu;
lamda0=lamda+1;

cnt=0;
while (abs(lamda0-lamda)>10e-4)&(cnt<100)
```

```
   cnt=cnt+1;
   nu1=0;pr=0;ee=0;waw=0;
   lamda0=lamda;
   for i=1:p
     nu1= nu1 + lamda/(u(i)+lamda);
     pr= pr + u(i)/(u(i)+lamda)^2;
     ee= ee + u(i)*yt(i)^2/(u(i)+lamda)^3;
     waw= waw + lamda^2*yt(i)^2/(u(i)+lamda)^2;
   end
   lamda=nu1*ee/(pr*waw);
end

value=lamda;
if cnt==100
    value=1;
end

%%%%%%%%%%%%%%%%%EstNuShort%%%%%%%%%%%%%%%%%%%%%%%

% easy way to estimate nu if not specified by the user
function value = EstNuShort(C,d)

value = 1/(sum(sum(C.^2))/size(C,2));
return

%%% function to calculate H and v %%%%%%%%%%%%%
function [H,v]=HV(A,d,bal);

[m,n]=size(A);e=ones(m,1);

if (bal==0)
     H=[A -e];
     v=(d'*H)';
else
     H=[A -e];
     mm=e;
     m1=find(d==-1);
     mm(m1)=(1/length(m1));
     m2=find(d==1);
     mm(m2)=(1/length(m2));
     mm=sqrt(mm);
     N=spdiags(mm,0,m,m);
     H=N*H;
    %keyboard
    v=(d'*N*H)';
end
```

2.2 CMAT Code for PSVM

The following is a very short CMAT module illustrating the essential algorithm of PSVM:

```
/* PSVM: linear and nonlinear Classification */
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("..\\tdata\\heart_60.dat");
x = data[,2:14]; y = data[,1];
d = diag(y); nu = 1.;
< w, gamma > = psvm(x,d,nu);
print "PSVM Results: Weights=", w;
print "PSVM Results: Gamma", gamma;
```

```
 PSVM Results: weights=
   |         1
--------------
 1 |  -0.38162
 2 |   0.23916
 3 |   0.03728
 4 |   0.51970
 5 |  -0.22208
 6 |   0.00553
 7 |   0.10994
 8 |  -0.38270
 9 |  -0.04526
10 |   0.22988
11 |   0.35865
12 |   0.60961
```

```
13 |   0.19337


PSVM Results: gamma=-0.5940
```

Similar results are obtained by the following `svm` call:

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

optn = [ "print"           2,
         "pplan"            ,
         "meth"       "prox",
         "c"             1. ,
         "kern"       "line" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);


         Linear Separating Plane (w*x = 0.59404)
         ***************************************

               Dense Row Vector (ncol=14)

 R |           1          2          3          4          5
      -0.3816233  0.2391562  0.0372758  0.5196974 -0.2220820

 R |           6          7          8          9         10
       0.0055331  0.1099441 -0.3827023 -0.0452577  0.2298814

 R |          11         12         13         14
       0.3586522  0.6096060  0.1933694  0.5940395
```

3 LSVM: Lagrangean SVM

3.1 LSVM for Linear Kernel

3.1.1 Olvi's Matlab Code

The following Matlab code by Musicant and Olvi can be found at Olvi Mangasarian's website. In addition to the LSVM implementation it contains a number

of other features, like two ways for estimating the regularization parameter ν (which is usually known as C) and some fit evaluation.

```
function [iter, optCond, time, w, gamma] = ...
       lsvm(A,D,nu,tol,maxIter,alpha,perturb,normalize);
% LSVM Langrangian Support Vector Machine algorithm
%   LSVM solves a support vector machine problem using an iterative
%   algorithm inspired by an augmented Lagrangian formulation.
%
%   iters = lsvm(A,D)
%
%   where A is the data matrix, D is a diagonal matrix
%     of 1s and -1s indicating which class the points
%     are in, and 'iters' is the number of iterations
%     the algorithm used.
%
%   All the following additional arguments are optional:
%
%   [iters, optCond, time, w, gamma] = ...
%     lsvm(A,D,nu,tol,maxIter,alpha,perturb,normalize)
%
%   optCond is the value of the optimality condition at termination.
%   time is the amount of time the algorithm took to terminate.
%   w is the vector of coefficients for the separating hyperplane.
%   gamma is the threshold scalar for the separating hyperplane.
%
%   On the right hand side, A and D are required. If the rest are not
%   specified, the following defaults will be used:
%     nu = 1/size(A,1), tol = 1e-5, maxIter = 100, alpha = 1.9/nu,
%       perturb = 0, normalize = 0
%
%   perturb indicates that all the data should be perturbed
%     by a random amount between 0 and the value given.
%     perturb is recommended only for highly degenerate
%     cases such as the exclusive or.
%
%   normalize should be set to 1 if the data should
%     be normalized before training.
%
%   The value -1 can be used for any of the entries
%   (except A and D) to specify that default values should be used.
%
%   Copyright (C) 2000 Olvi L. Mangasarian and David R. Musicant.
%   Version 1.0 Beta 1
%   This software is free for academic and research use only.
```

```
%   For commercial use, contact musicant@cs.wisc.edu.

  % If D is a vector, convert it to a diagonal matrix.
  [k,n] = size(D);
  if k==1 | n==1
    D=diag(D);
  end;

  % Check all components of D and verify that they are +-1
  checkall = diag(D)==1 | diag(D)==-1;
  if any(checkall==0)
    error('Error in D: classes must be all 1 or -1.');
  end;

  m = size(A,1);

  if ~exist('nu') | nu==-1
    nu = 1/m;
  end;
  if ~exist('tol') | tol==-1
    tol = 1e-5;
  end;
  if ~exist('maxIter') | maxIter==-1
    maxIter = 100;
  end;
  if ~exist('alpha') | alpha==-1
    alpha = 1.9/nu;
  end;
  if ~exist('normalize') | normalize==-1
    normalize = 0;
  end;
  if ~exist('perturb') | perturb==-1
    perturb = 0;
  end;

  % Do a sanity check on alpha
  if alpha > 2/nu,
    disp(sprintf('Alpha is larger than 2/nu.
    Algorithm may not converge.'));
  end;

  % Perturb if appropriate
  rand('seed',22);
  if perturb,
    A = A + rand(size(A))*perturb;
  end;
```

```
% Normalize if appropriate
if normalize,
  avg = mean(A);
  dev = std(A);
  if (isempty(find(dev==0)))
    A = (A - avg(ones(m,1),:))./dev(ones(m,1),:);
  else
    warning('Could not normalize matrix:
             at least one column is constant.');
  end;
end;

% Create matrix H
[m,n] = size(A);
e = ones(m,1);
H = D*[A -e];
iter = 0;
time = cputime;

% "K" is an intermediate matrix used often in SMW calclulations
K = H*inv((speye(n+1)/nu+H'*H));

% Choose initial value for x
x = nu*(1-K*(H'*e));

% y is the old value for x, used to check for termination
y = x + 1;

while iter < maxIter & norm(y-x)>tol
  % Intermediate calculation which is used twice
  z = (1+pl(((x/nu+H*(H'*x))-alpha*x)-1));
  y = x;
  % Calculate new value of x
  x=nu*(z-K*(H'*z));
  iter = iter + 1;
end;

% Determine outputs
time = cputime - time;
optCond = norm(x-y);
w = A'*D*x;
gamma = -e'*D*x;
disp(sprintf('Running time (CPU secs) = %g',time));
disp(sprintf('Number of iterations = %d',iter));
disp(sprintf('Training accuracy = %g',sum(D*(A*w-gamma)>0)/m));
```

```
  return;

function pl = pl(x);
  %PLUS function : max{x,0}
  pl = (x+abs(x))/2;
  return;
```

3.1.2 CMAT Code for LSVM

The following is a very short CMAT module illustrating the essential algorithm of LSVM:

```
  /* CMAT Formulation of LSVM Algorithm */
  function lsvm(A,y,nu,alf) {
    m = nrow(A); n = ncol(A); np = n + 1;
    rnu = 1. / nu;
    D = diag(y);
    A = A -> 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 = 400; tol = 1.e-4;
    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 = A‘ * D * u; v = D * u;
    gamma = -v[+];
    return(w,gamma);
  }

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

```
  x = data[,2:14]; y = data[,1];
  nu = 1.; alf = 1.9 / nu;

  < w, gamma > = lsvm(x,y,nu,alf);
  print "LSVM Result: gamma=", gamma;
  print "LSVM Result: weights=", w;


Iteration 0 Residual= 1.0000
Iteration 1 Residual= 0.6761
Iteration 2 Residual= 0.4336
Iteration 3 Residual= 0.3460
Iteration 4 Residual= 0.2602
Iteration 5 Residual= 0.2198
.............................
Iteration 65 Residual= 0.0001448
Iteration 66 Residual= 0.0001303
Iteration 67 Residual= 0.0001173
Iteration 68 Residual= 0.0001055
Iteration 69 Residual= 0.00009496


LSVM Result: gamma=-0.7617


LSVM Result: weights=
   |         1
--------------
 1 |  -0.40586
 2 |   0.32313
 3 |   0.02611
 4 |   0.69966
 5 |  -0.10727
 6 |  -0.02071
 7 |   0.11850
 8 |  -0.36877
 9 |  -0.04292
10 |   0.13782
11 |   0.44317
12 |   0.83634
13 |   0.21017
14 |  -0.76168


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

```
modl = "1 = 2 : 14";
class = 1;

optn = [ "print"           2,
         "pplan"            ,
         "meth"       "lagr",
         "c"             1. ,
         "kern"       "line" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

LSVM Iteration History

iter	absxdif	relxdif	xnorm
1	0.67613668	0.42991714	4.50315979
2	0.43355739	0.27293178	4.47579015
3	0.34602820	0.22113435	4.45469115
4	0.26024845	0.16355259	4.44794995
5	0.21978286	0.13883460	4.44184157
6	0.17744518	0.11123565	4.43980688
7	0.15167592	0.09542207	4.43771627
8	0.12666362	0.07935424	4.43702304
9	0.10878276	0.06832289	4.43623427
10	0.09247061	0.05793341	4.43596349
...	...	...	...
80	2.975e-005	1.865e-005	4.43508018
81	2.677e-005	1.678e-005	4.43508019
82	2.409e-005	1.510e-005	4.43508018
83	2.168e-005	1.359e-005	4.43508019
84	1.950e-005	1.223e-005	4.43508019
85	1.755e-005	1.100e-005	4.43508019
86	1.579e-005	9.902e-006	4.43508019

Linear Separating Plane (w*x = 0.761674)

Dense Row Vector (ncol=14)

R |	1	2	3	4	5
	-0.4058607	0.3231554	0.0261167	0.6996711	-0.1072626

R |	6	7	8	9	10
	-0.0207192	0.1185179	-0.3687780	-0.0428979	0.1378322

```
 R |          11         12         13         14
       0.4431936  0.8363591  0.2102033  0.7616736
```

3.2 LSVM for Nonlinear Kernel

3.2.1 Olvi's Matlab Code

The following is a short version of the core algorithm of LSVM for nonlinear kernel:

```
function [iter, optCond, time, u] = ...
    lsvmk(KM,D,nu,tol,maxIter,alpha);

% LSVMK Langrangian Support Vector Machine algorithm
%   LSVMK solves a support vector machine problem using an iterative
%   algorithm inspired by an augmented Lagrangian formulation.
%
%   iters = lsvmk(KM,D)
%
%   where KM is the kernel matrix, D is a diagonal matrix
%     of 1s and -1s indicating which class the points are in,
%     and 'iters' is the number of iterations the algorithm used.
%
%   All the following additional arguments are optional:
%
%   [iters, optCond, time, u] = ...
%     lsvmk(KM,D,nu,tol,maxIter,alpha)
%
%   optCond is the value of the optimality condition at termination.
%   time is the amount of time the algorithm took to terminate.
%   u is the vector of dual variables.
%
%   On the right hand side, KM and D are required. If the rest
%     are not specified, the following defaults will be used:
%     nu = 1/size(KM,1), tol = 1e-5, maxIter = 100, alpha = 1.9/nu
%
%   The value -1 can be used for any of the entries
%    (except A and D) to specify that default values should be used.
%
%   Copyright (C) 2000 Olvi L. Mangasarian and David R. Musicant.
%   Version 1.0 Beta 1
%   This software is free for academic and research use only.
%   For commercial use, contact musicant@cs.wisc.edu.
```

```
% If D is a vector, convert it to a diagonal matrix.
[k,n] = size(D);
if k==1 | n==1
  D=diag(D);
end;

% Check all components of D and verify that they are +-1
checkall = diag(D)==1 | diag(D)==-1;
if any(checkall==0)
  error('Error in D: classes must be all 1 or -1.');
end;

m = size(KM,1);

if ~exist('nu') | nu==-1
  nu = 1/m;
end;
if ~exist('tol') | tol==-1
  tol = 1e-5;
end;
if ~exist('maxIter') | maxIter==-1
  maxIter = 100;
end;
if ~exist('alpha') | alpha==-1
  alpha = 1.9/nu;
end;

% Do a sanity check on alpha
if alpha > 2/nu,
  disp(sprintf('Alpha is larger than 2/nu.
                Algorithm may not converge.'));
end;

% Create matrix H
[m,n] = size(KM);
e = ones(m,1);
I = speye(m);
iter = 0;
time = cputime;

% Generate Q matrix and inverse
Q = I/nu + D*KM*D;
P = inv(Q);

% Choose initial value for x
```

```
  u = P*e;

  % uold is the old value for u, used to check for termination
  oldu = u + 1;
  while iter < maxIter & norm(oldu-u)>tol
    oldu = u;
    u = P*(1+pl(Q*u-1-alpha*u));
    iter = iter + 1;
  end;

  % Determine outputs
  time = cputime - time;
  optCond = norm(u-oldu);
  disp(sprintf('Running time (CPU secs) = %g',time));
  disp(sprintf('Number of iterations = %d',iter));
  disp(sprintf('Training accuracy = %g',sum(D*KM*D*u>0)/m));

  return;

function pl = pl(x);
  %PLUS function : max{x,0}
  pl = (x+abs(x))/2;
  return;
```

3.3 Essential CMAT Code

4 SSVM: Smooth SVM

4.1 Olvi's Matlab Code

The following Matlab code for SSVM by Lee and Mangasarian can be found at Olvi Mangasarian's website. In addition to the SSVM implementation it contains a number of other features, like two ways for estimating the regularization parameter ν (which is usually known as C) and some fit evaluation.

```
function [w, gamma, trainCorr, testCorr, cpu_time, nu] = ssvm(C,d,k,nu,output,step_size,tol
% version 1.1
```

```
% last revision: 01/24/03
%=====================================================================
%Usage: [w gamma trainCorr testCorr nu] =
%            ssvm (C,d,k,nu,output,step_size,tol,maxIter,w0,gamma0)
%
%
%A and d are both required, everything else has a default
%An example: [w gamma train test nu] = ssvm(A, d, 10);
%
%=====================================================================
%Input parameters:
%
%  A:  Represent data points (mxn)
%  d:  d is a m dimensional vector of 1's or -1's
%       containing the corresponding labels for
% each data point in A.
%  k:  way to divide the data set into test and training set
%       if k = 0: simply run the algorithm without
%                 any correctness calculation
%       if k = 1: run the algorithm and calculate
%                 correctness on the whole data set
%       if k = any value less than the # of rows in the data set:
%           divide up the data set into test and training
%           using k-fold method
%       if k = # of rows in the data set:
%           use the 'leave 1' method
%
% output: 0 - no output, 1 - produce output, default is 0
% nu: weighted parameter
%               -1 - easy estimation
%                0  - hard estimation
%               any other value - used as nu by the algorithm
%               default - 0
% [w0; gamma0]:  Initial point
% step_size: 1 indicates Armijo stepsize,
%                  0 indicates Newton stepsize
%
%=====================================================================
%Output parameters:
%
%  w:  the normal vector of the classifier
%  gamma: the threshold
%  trainCorr: training set correctness
%  testCorr: test set correctness
%  cpu_time: time elapsed
%  nu:          estimated value (or specified value) of nu
```

```
%=====================================================================
%Technical Notes:
%
%    1. In order to handle a massive dataset this code
%       takes the advantage of sparsity of the Hessian
%       matrix.
%
%    2. We used the limit values of the sigmoid function
%       and p-function as the smoothing parameter \alpha
%       goes to infinity when we computer the Hessian
%       matrix and the gradient of objective function.
%
%    3. Decrease nu when the classifier is overfitting
%       the training data
%

if nargin<10
gamma0=0;
end
if nargin<9
s=size(C,2);
w0=zeros(s,1);
end
if nargin<8
maxIter=1000;
end
if nargin<7
tol=10e-8;
end
if nargin<6
step_size=1;
end
if nargin<5
output = 0;
end

if ((nargin<4)|(nu==0))
     nu = EstNuLong(C,d);  % default is hard estimation
elseif nu==-1  % easy estimation
  nu = EstNuShort(C,d);
end

if nargin<3
k = 0;
end
```

```
r=randperm(size(d,1));d=d(r,:);C=C(r,:);    % random permutation

tic;

%move one point in A a little if perfectly balanced
Cback=C;dback=d;   %  backup C and d
[sm sn]=size(C);
ma=C(find(d==1),:); mb=C(find(d==-1),:);
[s1 s2]=size(ma);
c1=sum(ma)/s1;
[s1 s2]=size(mb);
c2=sum(mb)/s1;
if c1==c2
   nu = 1;  % use 1 for perfectly balanced situation
   C(3,:)=C(3,:)+.001*norm(C(3,:)-c1,inf)*ones(1,sn);

end

trainCorr = 0;
testCorr = 0;

   % if k=0 no correctness is calculated, just run the algorithm
if k==0
  [w, gamma, iter] = core(C,d,nu,w0,gamma0,step_size,tol,maxIter,output);
  cpu_time = toc;
  if output==1
  fprintf(1,'\nNumber of Iterations: %d',iter);
   fprintf(1,'\nElapse time: %10.2f\n',cpu_time);
  end
  return
end

%if k==1 only training set correctness is calculated
if k==1
  [w, gamma, iter] = core(C,d,nu,w0,gamma0,step_size,tol,maxIter,output);
  trainCorr = correctness(C,d,w,gamma);
  cpu_time = toc;
  if output == 1
    fprintf(1,'\nTraining set correctness: %3.2f%% \n',trainCorr);
fprintf(1,'\nNumber of Iterations: %d',iter);
    fprintf(1,'\nElapse time: %10.2f\n',cpu_time);
  end
  return
end

    accuIter = 0;
```

```
indx = [0:k];
indx = floor(sm*indx/k);    %last row numbers for all 'segments'
% split trainining set from test set
for i = 1:k
 Ctest = []; dtest = [];Ctrain = []; dtrain = [];

Ctest = C((indx(i)+1:indx(i+1)),:);
dtest = d(indx(i)+1:indx(i+1));

Ctrain = C(1:indx(i),:);
Ctrain = [Ctrain;C(indx(i+1)+1:sm,:)];
dtrain = [d(1:indx(i));d(indx(i+1)+1:sm,:)];

 [w, gamma, iter] = core(Ctrain,dtrain,nu,w0,gamma0,step_size,tol,maxIter,output);
tmpTrainCorr = correctness(Ctrain,dtrain,w,gamma);
tmpTestCorr = correctness(Ctest,dtest,w,gamma);

 if output==1
   fprintf(1,'_____________________________________________\n');
fprintf(1,'Fold %d\n',i);
fprintf(1,'Training set correctness: %3.2f%%\n',tmpTrainCorr);
fprintf(1,'Testing set correctness: %3.2f%%\n',tmpTestCorr);
fprintf(1,'Number of iterations: %d\n',iter);
fprintf(1,'Elapse time: %10.2f\n',toc);
end

trainCorr = trainCorr + tmpTrainCorr;
testCorr = testCorr + tmpTestCorr;
accuIter = accuIter + iter; % accumulative iterations

end % end of for (looping through test sets)

trainCorr = trainCorr/k;
testCorr = testCorr/k;
cpu_time=toc/k;

if output == 1
fprintf(1,'=============================================');
  fprintf(1,'\nTraining set correctness: %3.2f%%',trainCorr);
  fprintf(1,'\nTesting set correctness: %3.2f%%',testCorr);
fprintf(1,'\nAverage number of iterations: %d',accuIter/k);
fprintf(1,'\nAverage cpu_time: %10.2f\n',cpu_time);
end

%%%%%%%%%%%%%%%%%%%% Core SSVM function %%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [w, gamma, iteration] = core(C,d,nu,w0,gamma0,step_size,tol,maxIter,output)

%separating the classes
iteration=0;
[ma, n] = size(C(find(d==1),:)); mb = size(C(find(d==-1),:),1);
C=[C(find(d==1),:);-C(find(d==-1),:)]; % equals "DA" in SSVM paper
d = [ones(ma,1); -ones(mb,1)]; % equal "De" is the paper

flag = 1;
H = zeros(ma+mb,1);
rv = zeros(ma+mb,1); e = ones(ma+mb,1);
while flag > tol & iteration< maxIter   %SZ
%tol 5flag is the optimality condition (smaller the better)
% put a counter to count iterations.
iteration=iteration+1;    %SZ
% Find a search direction!

  temp = C*w0 - gamma0*d; % D(Aw0 -e \gamma0)
  rv =  e -temp; % e - D(Aw0 - e \gamma0)
  % Compute the Hessian matrix
  H = (e + sign(rv))/2;
  % We only consider the nonzero part
  Ih= find(H ~= 0); ih = length(Ih);
  Hs = H(Ih); T = speye(ih);
  SH = C(Ih,:)'*spdiags(Hs, 0, T);
  P = SH*C(Ih,:);  q = SH*d(Ih);
  clear SH;
  oneh = norm(Hs,1);
  % Q is the Hessian matrix
  Q = speye(n+1) +nu*[P,(-q); (-q'), oneh];

  % Compute the gradient
  prv = max(rv,0); % (e- D(Aw0 - e \gamma0))_+
  gradz = [(w0 - nu*C'*prv); gamma0+nu*d'*prv];

  if  gradz'*gradz > tol  % Check the First Order Opt. condition
      b =  - gradz;
      z = Q\b;  % z is the Newton direction

      % Compute the gap!
      % (Only when you want to use the Armijo 's rule!)

      gap = z'*gradz;

      % Find the step size & Update to the new point !
       if step_size~=1
```

```
            w0 = w0+z(1:n);
            gamma0 = gamma0+z(n+1);
        else
            stepsize = armijo(C,d,w0,gamma0,nu,z, gap);
            w0 = w0 +stepsize*z(1:n);
            gamma0= gamma0 +stepsize*z(n+1);
        end
       flag = z'*z;
  else
    flag = tol;     %SZ
  end;

% if output==1
%    if (((iteration/10)==floor(iteration/10))|(iteration==1))
%     fprintf(1,'_____________________________________________\n');
%     fprintf(1,'Iteration        Optimality         Elapse Time\n');
%    end
%     fprintf(1,'%d           %12.5f      %10.2f      \n',iteration,flag,toc);
% end

end;  %while

w = w0; gamma = gamma0;
return  % end of core function

%%%%%%%%%%%%%%% correctness calculation %%%%%%%%%%%%%%%

function corr = correctness(AA,dd,w,gamma)

p=sign(AA*w-gamma);
corr=length(find(p==dd))/size(AA,1)*100;
return

%%%%%%%%%%%%Armijo stepsize function%%%%%%%%%%%%%%%%%%%%%%%

function stepsize = armijo(C,d,w,gamma,nu,zd, gap)

% Input
%   C = [A; -B]; equals "DA" in SSVM paper
%   d: equals "De" in SSVM paper (i.e. the diagonal of "D")
%   w, gamma: Current point
%   nu: weight parameter
%   gap: defined in ssvm code
% Note:
%   You will need objf.m to evaluate the objective function value.
%
```

```
temp =1; n = length(w);
obj1 =  objf(C,d, w,gamma,nu);
w2 = w+temp*zd(1:n);
gamma2 = gamma +temp*zd(n+1);
obj2 = objf(C,d,w2,gamma2,nu);
diff = obj1 - obj2;
while diff  < -0.05*temp*gap

      temp = 0.5*temp;
      w2 = w+temp*zd(1:n);
      gamma2 = gamma +temp*zd(n+1);
      obj2 = objf(C,d, w2,gamma2,nu);
      diff = obj1 - obj2;

end;
stepsize = temp;
return
%%%%%%%%%%%%%%%objf.m%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function value = objf(C,d,w,gamma,nu)
%
% Evaluate the function value
%

temp = abs(d)-(C*w - gamma*d);
v = max(temp,0);
value = 0.5*(nu*v'*v + w'*w + gamma^2);
return

%%%%%%%%%%%%%%%EstNuLong%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% hard way to estimate nu if not specified by the user
function value = EstNuLong(C,d)

[m,n]=size(C);e=ones(m,1);
H=[C -e];
if m<201
H2=H;d2=d;
else
r=rand(m,1);
[s1,s2]=sort(r);
H2=H(s2(1:200),:);
d2=d(s2(1:200));
end

lamda=1;
[vu,u]=eig(H2*H2');u=diag(u);p=length(u);
```

```
yt=d2'*vu;
lamda0=lamda+1;

cnt=0;
while (abs(lamda0-lamda)>10e-4)&(cnt<100)
     cnt=cnt+1;
     nu1=0;pr=0;ee=0;waw=0;
     lamda0=lamda;
     for i=1:p
     nu1= nu1 + lamda/(u(i)+lamda);
pr= pr + u(i)/(u(i)+lamda)^2;
ee= ee + u(i)*yt(i)^2/(u(i)+lamda)^3;
waw= waw + lamda^2*yt(i)^2/(u(i)+lamda)^2;
   end
lamda=nu1*ee/(pr*waw);
end

value = lamda;
if cnt==100
    value=1;
end
return

%%%%%%%%%%%%%%%%%EstNuShort%%%%%%%%%%%%%%%%%%%%%%%

% easy way to estimate nu if not specified by the user
function value = EstNuShort(C,d)

value = 1/(sum(sum(C.^2))/size(C,2));
return
```

4.2 CMAT Code for SSVM

The following CMAT code is not written in form of a subroutine but it shows with an example the essential steps of the SSVM algorithm:

```
/*--- Radial Base (RBF) Kernel ---*/
data = rspfile("..\\tdata\\heart_60.dat",":");
nobs = nrow(data); nvar = ncol(data);
nu = 1.;
print "nobs,nvar=", nobs,nvar;

srand(123);
```

```
y = data[,1]; x = data[,2:14];
y = replace(y,0.,-1.);
/* print y; */
nsmp = 20;
/* two methods of selecting random index vector,
   ind1: use ~ permutation operator
   ind2: use randperm() function */
ind = [1 : nobs]; ind1 = ind[~];
srand(123);
ind2 = randperm(nobs);
ind = ind2[1:nsmp];
xs = x[ind,]; ys = y[ind];

/* Compute rectangular kernel matrix rec[nobs,nsmp] */
lamb = 1.;
rec = cons(nobs,nsmp,0.);
for (j = 1; j <= nobs; j++)
for (k = 1; k <= nsmp; k++) {
  v = x[j,] - xs[k,];
  w = -lamb * v * v‘;
  rec[j,k] = exp(w);
}
print "dim(rec)=", nrow(rec), ncol(rec);
/* print "Rectangular Matrix=",rec; */

/* Initialize for Iteration: get sign of C */
dy = diag(y); ds = diag(ys);
C = dy * rec * ds;

/* print "Input Matrix C=", C; */
w0 = cons(nsmp,1,0.); gam0 = 0.;
e  = cons(nobs,1,1.);
rv = e - (C * w0 - gam0 * y);
prv = max(rv,0.);
crit = .5*(nu*prv‘*prv + w0‘*w0 + gam0*gam0);
print "Starting crit=",crit;
/* print "RV=", rv; */

/*--- Script for SSVM ---*/
iter = 0; snrm = 1.;
while (snrm > 1.e-8) {
  /* Compute grad and Hessian */
  iter++;
  hv = (e + sign(rv)) / 2;
  oneh = nu * hv[+];
  Ch = C‘ * diag(hv);
```

```
  P = nu * Ch * C; q = -nu * Ch * y;
  bot = q‘ -> oneh ;
  Hes = [ P  q , bot ];
  Hes += ide(nsmp+1);
  /* print "Hessian", Hes; */

  /* Compute Newton direction */
  g1 = w0 - nu * C‘ * prv;
  g2 = gam0 + nu * y‘ * prv;
  grad = [ g1, g2 ];
  /* print "GRAD=", grad; */

  /* Test convergence */
  gnrm = grad‘ * grad;
  if (gnrm <= 1.e-8) {
    print "Convergence: iter=", iter," crit=",crit," gnrm=",gnrm;
    break;
  }

  b = -grad;
  sdir = Hes \ b;
  /* print "SDIR=", sdir; */
  w0   += sdir[1:nsmp];
  gam0 += sdir[nsmp+1];
  desc = sdir‘ * grad;
  snrm = sdir‘ * sdir;
  /* print "Iterate gam0, w0=", gam0, w0; */
  rv = e - (C * w0 - gam0 * y);
  prv = max(rv,0.);
  crit = .5*(nu*prv‘*prv + w0‘*w0 + gam0*gam0);
  print "Iter=", iter," crit=", crit,
        " gnrm=", gnrm,"  desc=",desc,snrm;
}
print "Result: w0=", w0;
print "Result: gam0=", gam0;

/* Perform scoring */
sco = rec * ds * w0 - gam0;
pre = sign(sco);
res = y - pre;
tab = [ y pre res ];
print "Residuals=", tab;
```

```
Iter= 1 crit= 22.939 gnrm= 72.946  desc=-14.122 5.8432
```

```
Convergence: iter= 2 crit= 22.939 gnrm= 5.891e-030


Result: w0=
   |         1
--------------
 1 |   0.51210
 2 |   0.50155
 3 |   0.52104
 4 |   0.56858
 5 |   0.73719
 6 |   0.50052
 7 |   0.50674
 8 |   0.81510
 9 |   0.62543
10 |   0.50150
11 |   0.55055
12 |   0.56313
13 |   0.49179
14 |   0.57491
15 |   0.54581
16 |   0.15158
17 |   0.53784
18 |   0.29624
19 |   0.51076
20 |   0.47471


Result: gam0=-0.007102
```

5 LPSVM: L1 SVM

5.1 Case: $Nobs > nvar$

5.1.1 Olvi's Matlab Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   lpsvm when m>=n                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [w,gamma,iter]=lpsvm_with_smw(A,d,nu,delta)
%with Sherman SMW without armijo
```

```
%parameters
epsi=10^(-3); alpha=10^(2); tol=10^(-5); maxiter=100;
[m,n]=size(A);
en=ones(n,1);
em=ones(m,1);

% initial u
u=ones(m,1);iter=0;
epsi=epsi*em;nu=nu*em;
diff=1;
DA=spdiags(d,0,m,m)*A;
while (diff>tol) & (iter<maxiter)
    uold=u;
    iter=iter+1;
    du=d.*u;Adu=A'*du;
    pp=max(Adu-en,0);np=max(-Adu-en,0);
    dd=sum(du)*d;unu=max(u-nu,0);uu=max(-u,0);
    %Gradient
    g=-epsi+(d.*(A*pp))-(d.*(A*np))+dd+unu-alpha*uu;
    %Hessian
    E=spdiags(sqrt(sign(np)+sign(pp)),0,n,n);
    H=[DA*E d];
    f=1./(delta+sign(unu)+alpha*sign(uu));
    F=spdiags(f,0,m,m);gg=f.*g;HT=H';
    di=(eye(n+1)+HT*(F*H))\(HT*gg);
    di=H*di;di=f.*di;di=-gg+di;u=u+di;
    diff=norm(g);
end

w=1/epsi(1)*(pp-np);
gamma=-(1/epsi(1))*sum(du);
iter
return
```

5.1.2 CMAT Code

```
  /* [1.1] CMAT Formulation of LP_SVM Algorithm:
           for Nobs > nvar: with SMW */

  function lpsvm1(A,y,nu,delta,alf,eps1,armlin) {
   m = nrow(A); n = ncol(A); np1 = n + 1;
   rnu = 1. / nu; tol = .001; maxi = 50;
   epsm = cons(m,1,eps1); num = cons(m,1,nu);
```

```
/* Starting u[m] */
u = cons(m,1,1.);
D = diag(y); DA = D * A;

/* Prepare F and G */
du = D * u; Adu = A‘ * du;
pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
dsum = du[+]; dd = dsum * y;
unu = max(u-num,0.); uu = max(-u,0.);
/* Function */
t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf*uu‘*uu;
crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);
/* Gradient */
t1 = y .* (A*pp); t2 = y .* (A*np);
grd = -epsm + t1 - t2 + dd + unu - alf * uu;
gnrm = norm(grd);

iter = 0;
print "Iter=", iter, " Crit=", crit, " Gnrm=", gnrm;
while (++iter <= maxi && gnrm > tol) {
  uold = u; cold = crit;
  /* Hessian */
  e = sqrt(sign(np) + sign(pp));
  E = diag(e);
  B = DA * E; H = [ B y ];
  /* that's different */
  f = 1. / (delta + sign(unu) + alf*sign(uu));
  F = diag(f);
  gg = f .* grd; b = H‘ * gg;
  C = H‘ * F * H + ide(np1);
  di = C \ b;
  di = f .* (H * di); di -= gg;
  u = uold + di;

  /* Prepare */
  du = D * u; Adu = A‘ * du;
  pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
  dsum = du[+]; dd = dsum * y;
  unu = max(u-num,0.); uu = max(-u,0.);

  /* Function */
  t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf * uu‘*uu;
  crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);

  /* Armijo line search for beta */
  beta = 1.;
```

```
    if (armlin && crit >= cold) {
      beta = .5;
      while (beta > 1.e-12) {
        u = uold + beta * di;
        du = D * u; Adu = A‘ * du;
        pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
        dsum = du[+]; dd = dsum * y;
        unu = max(u-num,0.); uu = max(-u,0.);
        /* Function */
        t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf * uu‘*uu;
        crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);
        if (crit < cold) break;
        beta *= .5;
    } }

    /* Gradient */
    t1 = y .* (A*pp); t2 = y .* (A*np);
    grd = -epsm + t1 - t2 + dd + unu - alf * uu;
    gnrm = norm(grd);
    print "Iter=", iter, " Crit=", crit,
          " Gnrm=", gnrm, " Beta=", beta;
  }

  /* final weights and intercept gamma */
  w = (pp - np) / eps1;
  gamma = -du[+] / eps1;
  return(w,gamma);
 }



 print "Problem HEART from Statlog collection";
 print "Michie, Spiegelhalter, & Taylor, 1994";

 data = rspfile("..\\tdata\\heart_scale.dat");
 x = data[,2:14]; y = data[,1];
 armlin = 0; eps1 = .001; nu = 1.;
 alf = 1000.; delta = .001; /* default setting */
 < w, gamma > = lpsvm1(x,y,nu,delta,alf,eps1,armlin);

 print "LPSVM1 Result: Gamma=", gamma;
 print "LPSVM1 Result: weights=", w;


Iter= 0 Crit= 31606.04 Gnrm= 6006.67
Iter= 1 Crit= 5935.36 Gnrm= 3425.90 Beta= 1.0000
```

```
Iter= 2 Crit= 8560.04 Gnrm= 4145.50 Beta= 1.0000
Iter= 3 Crit= 4393.91 Gnrm= 2964.41 Beta= 1.0000
Iter= 4 Crit= 1125.30 Gnrm= 1498.83 Beta= 1.0000
Iter= 5 Crit= 508.086 Gnrm= 1007.99 Beta= 1.0000
Iter= 6 Crit= 22.453 Gnrm= 212.114 Beta= 1.0000
Iter= 7 Crit=-0.07733 Gnrm= 4.3015 Beta= 1.0000
Iter= 8 Crit=-0.09334 Gnrm= 0.4881 Beta= 1.0000
Iter= 9 Crit=-0.08718 Gnrm= 4.1509 Beta= 1.0000
Iter= 10 Crit=-0.09674 Gnrm= 0.0001197 Beta= 1.0000


LPSVM1 Result: Gamma=-1.0534


LPSVM1 Result: weights=
   |         1
--------------
 1 |   0.00000
 2 |   0.41704
 3 |   0.66287
 4 |   0.31688
 5 |   0.68604
 6 |  -0.18942
 7 |   0.24561
 8 |  -0.79834
 9 |   0.27687
10 |   0.65571
11 |   0.19385
12 |    1.0419
13 |   0.54345
```

Very similar results are obtained by executing the following `SVM` call:

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

/*--- L1LIN: with direct Cholesky   ---*/
optn = [ "print"           2 ,
         "popt"            2 ,
         "ppred"              ,
         "pplan"              ,
         "meth"      "l1lin" ,
```

```
          "nchunk"          1 ,
          "peneps"       .001 ,
          "dbeg"        0.001 ,
          "dend"        0.001 ,
          "abeg"        1000. ,
          "aend"        1000. ,
          "c"              1. ,
          "kern"       "line" ];
 < alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

The algorithm converges even without specifying Armijo line search. Therefore, the objective function is not reduced in every iteration:

```
          ---Start Training Cycle: Technique=  L1LIN---

         L1FS Iteration History (alpha=1000 delta=0.001)

  iter        crit        cdif       gnorm        xdif       beta
     1  5935.30019  25670.7358  3425.88490  16.2946702  1.0000000
     2  8559.94841 -2624.64821  4145.47507  13.6064260  1.0000000
     3  4393.87939  4166.06902  2964.39722  6.74360896  1.0000000
     4  1125.29640  3268.58299  1498.82350  4.41820496  1.0000000
     5  508.079879  617.216520  1007.97986  2.84943245  1.0000000
     6  22.4523518  485.627527  212.109616  1.48230984  1.0000000
     7 -0.07732813  22.5296799  4.30197207  0.53620107  1.0000000
     8 -0.09334375  0.01601562  0.48811423  0.27979803  1.0000000
     9 -0.08717789 -0.00616586  4.15078126  0.13887130  1.0000000
    10 -0.09673854  0.00956065  1.197e-004  0.11854318  1.0000000


         Linear Separating Plane (w*x = -1.0534)
         ***************************************

               Dense Row Vector (ncol=14)

R |          1          2          3          4          5
     0.0000000  0.4170373  0.6628647  0.3168829  0.6860352

R |          6          7          8          9         10
    -0.1894124  0.2456148 -0.7983388  0.2768692  0.6557127

R |         11         12         13         14
     0.1938456  1.0418935  0.5434474 -1.0533974
```

5.2 Case: $Nobs \leq nvar$

5.2.1 Olvi's Matlab Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   lpsvm when m<n                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [w,gamma,iter]=lpsvm_without_smw(A,d,nu,delta)
%without sherman and without armijo
%parameters
epsi=10^(-1);alpha=10^3;tol=10^(-3);maxiter=50;
[m,n]=size(A);
en=ones(n,1);
em=ones(m,1);
%initial u
u=ones(m,1);iter=0;
epsi=epsi*em;nu=nu*em;
diff=1;
DA=spdiags(d,0,m,m)*A;
while (diff>tol) & (iter<maxiter)
    uold=u;
    iter=iter+1;
    du=d.*u;Adu=A'*du;
    pp=max(Adu-en,0);np=max(-Adu-en,0);
    dd=sum(du)*d;unu=max(u-nu,0);uu=max(-u,0);
    %Gradient
    g=-epsi+(d.*(A*pp))-(d.*(A*np))+dd+unu-alpha*uu;
    %Hessian
    E=spdiags(sqrt(sign(np)+sign(pp)),0,n,n);
    H=[DA*E d];
    F=spdiags(delta+sign(unu)+alpha*sign(uu),0,m,m);
    di=-((H*H'+F)\g);
    u=u+di;
    diff=norm(g);
end
du=d.*u;Adu=A'*du;
pp=max(Adu-en,0);np=max(-Adu-en,0);
w=1/epsi(1)*(pp-np);gamma=-(1/epsi(1))*sum(du);
return
```

5.2.2 CMAT Code

```
/* [1.2] CMAT Formulation of LP_SVM Algorithm:
         for Nobs < nvar: without SMW */

function lpsvm2(A,y,nu,delta,alf,eps1,armlin) {
 m = nrow(A); n = ncol(A); np1 = n + 1;
 rnu = 1. / nu; tol = .001; maxi = 50;
 epsm = cons(m,1,eps1); num = cons(m,1,nu);

 /* starting u[m] */
 u = cons(m,1,1.);
 D = diag(y); DA = D * A;

 /* Prepare F and G */
 du = D * u; Adu = A‘ * du;
 pp = max(Adu-1.,0.); np = max(-Adu-1.,0.);
 dsum = du[+]; dd = dsum * y;
 unu = max(u-num,0.); uu = max(-u,0.);
 /* Function */
 t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf*uu‘*uu;
 crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);
 /* Gradient */
 t1 = y .* (A*pp); t2 = y .* (A*np);
 grd = -epsm + t1 - t2 + dd + unu - alf * uu;
 gnrm = norm(grd);

 iter = 0;
 print "Iter=", iter, " Crit=", crit, " Gnrm=", gnrm;
 while (++iter <= maxi && gnrm > tol) {
   uold = u; cold = crit;

   /* Hessian */
   e = sqrt(sign(np) + sign(pp));
   E = diag(e);
   B = DA * E; H = [ B y ];
   /* that's different */
   f = delta + sign(unu) + alf*sign(uu);
   F = diag(f);
   C = H * H‘ + F;
   di = C \ grd;
   u = uold - di;

   /* Prepare F and G */
   du = D * u; Adu = A‘ * du;
```

```
    pp = max(Adu-1.,0.); np = max(-Adu-1.,0.);
    dsum = du[+]; dd = dsum * y;
    unu = max(u-num,0.); uu = max(-u,0.);
    /* Function */
    t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf*uu‘*uu;
    crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);

    /* Armijo line search for beta */
    beta = 1.;
    if (armlin && crit >= cold) {
      beta = .5;
      while (beta > 1.e-12) {
        u = uold - beta * di;
        du = D * u; Adu = A‘ * du;
        pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
        dsum = du[+]; dd = dsum * y;
        unu = max(u-num,0.); uu = max(-u,0.);
        /* Function */
        t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf * uu‘*uu;
        crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);
        if (crit < cold) break;
        beta *= .5;
    } }

    /* Gradient */
    t1 = y .* (A*pp); t2 = y .* (A*np);
    grd = -epsm + t1 - t2 + dd + unu - alf * uu;
    gnrm = norm(grd);
    print "Iter=", iter, " Crit=", crit,
          " Gnrm=", gnrm, " Beta=", beta;
  }

  /* final weights and intercept gamma */
  w = (pp - np) / eps1;
  gamma = -du[+] / eps1;
  return(w,gamma);
 }
```

Here we enforce convergence by specifying that the Armijo line search algorithm is being used:

```
  print "NIR Spectra data set: train: nr=21, test: nr=7; nvar=268";
  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.)‘;
  x = xtrn;
  armlin = 1; /* armijo line search */
  eps1 = .1; nu = 1.;
  alf = 1000.; delta = .001; /* default setting */
  < w, gamma > = lpsvm2(x,y,nu,delta,alf,eps1,armlin);

  print "LPSVM2 Result: Gamma=", gamma;
  print "LPSVM2 Result: Weights=", w;


Iter= 0 Crit= 1219.77 Gnrm= 4805.16
Iter= 1 Crit= 1148.82 Gnrm= 4655.05 Beta= 0.03125
Iter= 2 Crit= 1110.34 Gnrm= 4319.15 Beta= 0.06250
Iter= 3 Crit= 1052.20 Gnrm= 3265.37 Beta= 0.2500
Iter= 4 Crit= 904.024 Gnrm= 2680.50 Beta= 0.2500
Iter= 5 Crit= 529.249 Gnrm= 1046.25 Beta= 1.0000
.............................................
Iter= 24 Crit=-1.0018 Gnrm= 0.1953 Beta= 0.01563
Iter= 25 Crit=-1.0114 Gnrm= 0.04851 Beta= 1.0000
Iter= 26 Crit=-1.0119 Gnrm= 0.00002976 Beta= 1.0000


LPSVM2 Result: Gamma=-2.3384
```

The solution is sparse, i.e. only 16 of the 268 coefficients are nonzero:

```
LPSVM2 Result: Weights=

                                  w
                                  *

                       Sparse Column Vector w

     C |          18         19         20         52         53
           0.5186556  0.7369969  0.1150145  0.2758565  0.3301452
```

```
    C |          71         72         73         74         84
         -0.7781402 -0.7981064 -0.4437834 -0.0209576 -0.0877437

    C |          85        101        102        103        104
         -0.0607864 -0.0373918 -0.1957998 -0.2900087 -0.2612492

    C |         105
         -0.0712348
```

Very similar results are obtained by executing the following SVM call:

```
  print "NIR Spectra data set: train: nr=21, test: nr=7; nvar=268";
  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.)';
  tnir = xtrn -> y;

  /* L1FS: with line search: peps=.1 */
  modl = "269 = 1:268";
  class = 269;
  optn = [ "print"           3 ,
           "popt"            2 ,
           "ppred"             ,
           "pplan"             ,
           "meth"       "l1fs" ,
           "lines"             ,
           "peneps"         .1 ,
           "dbeg"        0.001 ,
           "dend"        0.001 ,
           "abeg"        1000. ,
           "aend"        1000. ,
           "c"              1. ,
           "kern"       "line" ];
  alfa = svm(tnir,modl,optn,class);


          L1FS Iteration History (alpha=1000 delta=0.001)

   iter        crit        cdif       gnorm        xdif       beta
```

```
 1  1148.82236  70.9520695  4655.05485  3.31300825  0.0312500
 2  1110.34047  38.4818980  4319.15175  2.12693255  0.0625000
 3  1052.19721  58.1432532  3265.37024  1.75777741  0.2500000
 4  904.024058  148.173156  2680.49975  1.32892901  0.2500000
 5  529.248605  374.775453  1046.24679  3.76393101  1.0000000
 .....................................................
24 -1.00175646  5.696e-004  0.19527266  0.05268940  0.0156250
25 -1.01138019  0.00962373  0.04850791  0.19305714  1.0000000
26 -1.01186016  4.800e-004  2.976e-005  0.02976321  1.0000000


      Nonzeros of Separating Plane (w*x = -2.33837)
      ---------------------------------------------

                1  18  X18   0.518655587
                2  19  X19   0.736996896
                3  20  X20   0.115014457
                4  52  X52   0.275856473
                5  53  X53   0.330145196
                6  71  X71  -0.778140242
                7  72  X72  -0.798106377
                8  73  X73  -0.443783378
                9  74  X74  -0.020957637
               10  84  X84  -0.087743668
               11  85  X85  -0.060786361
               12 101  X101 -0.037391837
               13 102  X102 -0.195799766
               14 103  X103 -0.290008691
               15 104  X104 -0.261249225
               16 105  X105 -0.071234816
```

6 NSVM: Newton SVM

6.1 Case: $Nobs > nvar$

6.1.1 Olvi's Matlab Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        NSVM for m>=n                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [w,gamma,iter]=nsvm_with_smw(A,d,nu,arm,wp1);
% with armijo and with SMW
```

```
maxIter=100;
[m,n]=size(A);
iter=0;
u=zeros(m,1);e=ones(m,1);

H=[spdiags(d,0,m,m)*A -d];

% When balance is on this only approximates the norm
alpha=1.1*((1/nu)+(norm(H',2)^2));

if wp1==0.5
   v=u/nu+H*(H'*u)-e;
else
   vt=ones(m,1);
   vt(find(d==1))=(1-wp1)*ones(length(find(d==1)),1);
   vt(find(d==-1))=wp1*ones(length(find(d==-1)),1);
   v=vt.*u/nu+H*(H'*u)-e;
end
hu=-max((v-alpha*u),0)+v;

while norm(hu)>10^(-3) & (iter < maxIter)
    iter=iter+1;
    E=sign(max(v-alpha*u,0));
    if wp1==.5
       temp=(1./((alpha-(1/nu))*E+(1/nu)));
    else
       temp=(1./((alpha*E)-E.*vt*(1/nu)+vt.*(1/nu)));
   end
    G=spdiags(temp.*(1-E),0,m,m);
    FHU=temp.*hu;
    LD=H'*FHU;
    GH=G*H;
    SM=speye(n+1)+H'*GH;
    R=chol(SM);
    sol=R'\LD;
    sol=R\sol;
    delta=FHU-GH*sol;

    if arm==1 %armijo step without the balance yet
        lambda=1;
        v1=v+e;
        v2=v;
        su=0.5*(u'*v1)-sum(u)+(1/(2*alpha))
           *(norm(max(-alpha*u+v2,0))^2-norm(v2)^2);
        unew=u-lambda*delta;
        v1=u/nu+H*(H'*unew);
```

```
        v2=v1-e;
        sunew=0.5*(unew'*v1)-sum(unew)+(1/(2*alpha))
              *(norm(max(-alpha*unew+v2,0))^2-norm(v2)^2);
        while su-sunew < -(0.25)*lambda*hu

            lambda=0.1*lambda;
            unew=u-lambda*delta;
            v1=u/nu+H*(H'*unew);
            v2=v1-e;
            sunew=0.5*(unew'*v1)-sum(unew)+(1/(2*alpha))
                 *(norm(max(-alpha*unew+v2,0))^2-norm(v2)^2);
        end

    else
        unew=u-delta;
    end
    u=unew;

    if wp1==.5
       v=u/nu+H*(H'*u)-e;
   else
       v=vt.*u/nu+H*(H'*u)-e;
   end
    hu=-max((v-alpha*u),0)+v;

end

w=A'*(d.*u);gamma=-sum(d.*u);

return
```

6.1.2 CMAT Code

```
  /* [1.1] CMAT Formulation of N_SVM Algorithm:
           for Nobs > nvar: with SMW */

  function nsvm1(A,y,nu,armlin) {
   m = nrow(A); n = ncol(A); np1 = n + 1;
   rnu = 1. / nu; tol = .001; maxi = 100;

   /* Starting u,e[m] */
   u = cons(m,1,0.);
   e = cons(m,1,1.);
```

```
D = diag(y); DA = D * A;
H = DA -> -y; /* we cannot store large H*H‘ matrix */

hnrm = norm(H); /* largest singular value of H */
alf = 1.1 * (rnu + hnrm*hnrm);
hhu = H*(H‘*u);
v1 = hhu + rnu * u; v2 = v1 - e;
hu = v2 - max(v2-alf*u,0.);
hunrm = norm(hu); /* Euclidean norm: sqrt(SSQ) */
t1 = norm(max(v2-alf*u,0)); t1 *= t1;
t2 = norm(v2);              t2 *= t2;
crit = .5*u‘*v1 - u[+] + .5*(t1 - t2) / alf;

iter = 0;
while (++iter <= maxi) {
  if (hunrm <= tol) break;
  cold = crit; uold = u;
  w = max(v2-alf*u,0.);
  ev = sign(w);
  t1 = 1. / ((alf - rnu) * ev + rnu);
  t2 = t1 .* (1. - ev);
  G = diag(t2); fhu = t1 .* hu;
  ld = H‘ * fhu; GH = G * H;
  SM = ide(np1) + H‘ * GH;
  sol = SM \ ld;
  delta = fhu - GH * sol;
/* print "Delta=", delta; */

  /* Armijo line search for beta */
  beta = 1.;
  u = uold - delta;
  hhu = H*(H‘*u);
  v1 = hhu + rnu * u; v2 = v1 - e;
  t1 = norm(max(v2-alf*u,0)); t1 *= t1;
  t2 = norm(v2);              t2 *= t2;
  crit = .5*u‘*v1 - u[+] + .5*(t1 - t2) / alf;
  if (armlin) {
    at = 0;
    while (crit > cold && beta > 1.e-12) {
      at++; beta *= .5;
      u = uold - beta * delta;
      hhu = H*(H‘*u);

      v1 = hhu + rnu * u; v2 = v1 - e;
      t1 = norm(max(v2-alf*u,0)); t1 *= t1;
      t2 = norm(v2);              t2 *= t2;
```

```
        crit = .5*u'*v1 - u[+] + .5*(t1 - t2) / alf;
    } }
    hu = v2 - max(v2-alf*u,0.);
    hunrm = norm(hu); /* Euclidean norm: sqrt(SSQ) */
    print "Iter=", iter, " Crit=", crit,
          " HUNorm=", hunrm, " Beta=", beta;
  }

  /* final weights and intercept gamma */
  du = D * u;
  w = A' * du; gamma = -du[+];
  return(w,gamma);
 }

 print "Problem HEART from Statlog collection";
 print "Michie, Spiegelhalter, & Taylor, 1994";

 data = rspfile("..\\tdata\\heart_scale.dat");
 x = data[,2:14]; y = data[,1];
 armlin = 0; nu = 1.;
 < w, gamma > = nsvm1(x,y,nu,armlin);

 print "NSVM1 Result: Gamma=", gamma;
 print "NSVM1 Result: weights=", w;
```

```
Iter= 1 Crit= 1523.08 HUNorm= 1839.49 Beta= 1.0000
Iter= 2 Crit= 159.300 HUNorm= 681.732 Beta= 1.0000
Iter= 3 Crit=-51.555 HUNorm= 117.124 Beta= 1.0000
Iter= 4 Crit=-57.963 HUNorm= 4.1071 Beta= 1.0000
Iter= 5 Crit=-57.971 HUNorm= 1.928e-013 Beta= 1.0000


NSVM1 Result: Gamma=-0.6070


NSVM1 Result: weights=
   |         1
--------------
 1 |  -0.10436
 2 |   0.21681
 3 |   0.35047
 4 |   0.35256
 5 |   0.38100
 6 |  -0.11840
```

```
 7 |   0.10668
 8 |  -0.41184
 9 |   0.13739
10 |   0.33916
11 |   0.12675
12 |   0.55031
13 |   0.25316
```

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

/*--- L2FS: with direct Cholesky   ---*/
optn = [ "print"           2 ,
         "popt"            2 ,
         "ppred"             ,
         "pplan"             ,
         "meth"       "l2fs" ,
         "lines"             ,
         "c"              1. ,
         "kern"       "line" ];
< alfa,sres,vres,yptr > = svm(data,modl,optn,class);
```

NSVM Iteration History

iter	crit	cdif	gnorm	xdif
1	1523.08157	13399.3268	1839.49168	12.1291630
2	159.299827	1363.78174	681.731867	2.33518956
3	-51.5554753	210.855302	117.123834	0.77397219
4	-57.9627777	6.40730242	4.10706860	0.11825454
5	-57.9706668	0.00788917	2.193e-013	0.00393814

Linear Separating Plane (w*x = -0.606977)

Dense Row Vector (ncol=14)

R	1	2	3	4	5
	-0.1043631	0.2168092	0.3504698	0.3525627	0.3809957

R	6	7	8	9	10
	-0.1183990	0.1066772	-0.4118439	0.1373944	0.3391576

```
 R |          11         12         13         14
       0.1267524  0.5503142  0.2531636 -0.6069768
```

6.2 Case: $Nobs \leq nvar$

6.2.1 Olvi's Matlab Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        NSVM for m<n                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [w,gamma,iter]=nsvm_without_smw(A,d,nu,arm,wp1);
maxIter=100;
[m,n]=size(A);
iter=0;
u=zeros(m,1);e=ones(m,1);
HH=diag(d)*(A*A'+1)*diag(d);
if wp1==.5
   Q=speye(m)/nu+HH;
else
   vt=ones(m,1);
   vt(find(d==1))=(1-w1)*ones(length(find(d==1)));
   vt(find(d==-1))=w1*ones(length(find(d==-1)));
   Q=spdiags(vt,0,m,m)/nu+HH;
end
alpha=1.1*((1/nu)+norm(HH));
v=Q*u-e;
hu=-max((v-alpha*u),0)+v;

while norm(hu)>10^(-3) & (iter < maxIter)
    iter=iter+1;
    dhu=sign(max(((Q-alpha*eye(m))*u-e),0));% the 1/2 thing
    dhu=sparse((diag(1-dhu))*Q+diag(alpha*dhu));
    delta=dhu\hu;

    if arm==1 %armijo step
        lambda=1;
        v=Q*u;
        ve=v-e;
        su=0.5*(u'*v)-sum(u)+(1/(2*alpha))
          *(norm(max(-alpha*u+ve,0))^2-norm(ve)^2)
        unew=u-lambda*delta;
```

```
        v=Q*unew;
        ve=v-e;
        sunew=0.5*(unew'*v)-sum(unew)+(1/(2*alpha))
             *(norm(max(-alpha*unew+ve,0))^2-norm(ve)^2);
        at=0;
        while (su-sunew < -(0.25)*lambda*hu) & (at<5)
            at=at+1;
            disp('armijo');
            lambda=0.5*lambda;
            unew=u-lambda*delta;
            v=Q*unew;
            ve=v-e;
            sunew=0.5*(unew'*v)-sum(unew)+(1/(2*alpha))
                 *(norm(max(-alpha*unew+ve,0))^2-norm(ve)^2);
        end
        if at==5
            unew=u-delta;
        end
    else
        unew=u-delta;
    end

    u=unew;
    v=Q*unew-e;
    hu=-max((v-alpha*u),0)+v;
end

w=A'*(d.*u);gamma=-sum(d.*u);

return
```

6.2.2 CMAT Code

```
  /* [1.2] CMAT Formulation of N_SVM Algorithm:
           for Nobs < nvar: without SMW */

  function nsvm2(A,y,nu,armlin) {
   m = nrow(A); n = ncol(A); np1 = n + 1;
   rnu = 1. / nu; tol = .001; maxi = 100;

   /* Starting u,e[m] */
   u = cons(m,1,0.);
   e = cons(m,1,1.);
```

```
D = diag(y);
HH = D * (A*A‘ + 1.) * D;
hnrm = norm(HH); /* largest singular value of HH */
Q = rnu * ide(m) + HH;

alf = 1.1 * (rnu + hnrm);
/* print "eval=", hnrm, " alfa=", alf; */

v1 = qu = Q * u; v2 = v1 - e;
hu = v2 - max(v2-alf*u,0.);
hunrm = norm(hu); /* Euclidean norm: sqrt(SSQ) */

t1 = norm(max(v2-alf*u,0)); t1 *= t1;
t2 = norm(v2);              t2 *= t2;
crit = .5*u‘*v1 - u[+] + .5*(t1 - t2) / alf;
/* print "Crit=", crit; */

iter = 0;
while (++iter <= maxi) {
  if (hunrm <= tol) break;
  cold = crit; uold = u;

  v1 = qu - alf * u - e;
  dhu = sign(max(v1,0.));
  DHU = diag(1. - dhu) * Q + diag(alf * dhu);
  delta = DHU \ hu;

  /* Armijo line search for beta */
  beta = 1.;
  u = uold - delta;
  v1 = qu = Q * u; v2 = v1 - e;
  t1 = norm(max(v2-alf*u,0)); t1 *= t1;
  t2 = norm(v2);              t2 *= t2;
  crit = .5*u‘*v1 - u[+] + .5*(t1 - t2) / alf;

  if (armlin) {
    at = 0;
    while (crit > cold && beta > 1.e-12) {
      at++; beta *= .5;
      u = uold - beta * delta;
      v1 = Q * u; v2 = v1 - e;
      t1 = norm(max(v2-alf*u,0)); t1 *= t1;
      t2 = norm(v2);              t2 *= t2;
      crit = .5*u‘*v1 - u[+] + .5*(t1 - t2) / alf;
    }
    if (at >= 10) u = uold - delta;
```

```
      v1 = qu = Q * u; v2 = v1 - e;
    }
    hu = v2 - max(v2-alf*u,0.);
    hunrm = norm(hu); /* Euclidean norm: sqrt(SSQ) */
    print "Iter=", iter, " Crit=", crit,
          " HUNorm=", hunrm, " Beta=", beta;
  }

  /* final weights and intercept gamma */
  du = D * u;
  w = A' * du; gamma = -du[+];
  return(w,gamma);
 }

 print "NIR Spectra data set: train: nr=21, test: nr=7; nvar=268";
 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.)';
 x = xtrn;
 armlin = 0; nu = 1.;
 < w, gamma > = nsvm2(x,y,nu,armlin);

 print "NSVM2 Result: Gamma=", gamma;
 print "NSVM2 Result: Weights=", w;


NIR Spectra data set: train: nr=21, test: nr=7; nvar=268

Iter= 1 Crit= 3776.36 HUNorm= 12293.24 Beta= 1.0000
Iter= 2 Crit= 2201.55 HUNorm= 9386.83 Beta= 1.0000
Iter= 3 Crit= 394.731 HUNorm= 3980.55 Beta= 1.0000
Iter= 4 Crit=-1.4525 HUNorm= 2.726e-013 Beta= 1.0000


NSVM2 Result: Gamma=-0.03584


NSVM2 Result: Weights=
```

```
    |         1
----------------
  1 |   0.12649
  2 |   0.12714
  3 |   0.10498
  4 |   0.06656
  5 |   0.03067
  6 |   0.00827
  7 |   0.00060
  8 |   0.00807
  9 |   0.02733
 10 |   0.05017
 11 |   0.07429
 12 |   0.09992
 13 |   0.12286
 14 |   0.14088
 15 |   0.15658
 ...............
```

```
  print "NIR Spectra data set: train: nr=21, test: nr=7; nvar=268";
  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.)';
  tnir = xtrn -> y;

  /* L2FS: with line search: */
  modl = "269 = 1:268";
  class = 269;
  optn = [ "print"           3 ,
           "popt"            2 ,
           "ppred"             ,
           "pplan"             ,
           "meth"       "l2fs" ,
           "lines"             ,
           "c"              1. ,
           "kern"       "line" ];
  alfa = svm(tnir,modl,optn,class);
```

NSVM Iteration History

iter	crit	cdif	gnorm	xdif
1	3776.35545	-2910.25467	12293.2437	3.85531353
2	2201.55458	1574.80087	9386.82510	1.22402542
3	394.731123	1806.82346	3980.54590	0.75680150
4	-1.45249258	396.183616	3.684e-013	0.25415143

Linear Separating Plane (w*x = -0.0358424)

1 :	0.1265	0.1271	0.105	0.06656	0.03067
6 :	0.008272	0.0006033	0.008073	0.02733	0.05017
11 :	0.07429	0.09992	0.1229	0.1409	0.1566
	.....				
266 :	-0.01762	-0.01754	-0.01751		

Evaluation of Training Data Fit

Index	Value	StdErr
Absolute Classification Error	0	.
Classification Accuracy	100.0000000	.
Concordant Pairs	100.0000000	.
Discordant Pairs	0.000000000	.
Tied Pairs	0.000000000	.
Goodman-Kruskal Gamma	1.000000000	0.000000000
Kendall Tau_a	0.523809524	.
Kendall Tau_b	1.000000000	0.000000000
Stuart Tau_c	0.997732426	0.020759080
Somers D C|R	1.000000000	0.000000000
Somers D R|C	1.000000000	0.000000000

Classification Table

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

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

Index	Value	StdErr
Absolute Classification Error	0	.
Classification Accuracy	100.0000000	.
Concordant Pairs	100.0000000	.
Discordant Pairs	0.000000000	.
Tied Pairs	0.000000000	.
Goodman-Kruskal Gamma	1.000000000	0.000000000
Kendall Tau_a	0.476190476	.
Kendall Tau_b	1.000000000	0.000000000
Stuart Tau_c	0.816326531	0.292709047
Somers D C|R	1.000000000	0.000000000
Somers D R|C	1.000000000	0.000000000

Classification Table

Observed | Predicted	-1	1
-1	5	0
1	0	2

7 Chunking with linear L1 SVM

7.1 Olvi's Matlab Code

The first subroutine `solve_chunk` solves linear L1 SVM for a subset of the data and is called by a driver subroutine listed below. The subroutine `solve_chunk` is very similar to that of `function [w,gamma,iter]=lpsvm_with_smw(A,d,nu,delta)` listed in section *LPSVM: L1 SVM* for the case $Nobs > nvar$. Here, however, chunking requires more precision with `tol=1.e-6` and also a stronger termination criterion for the iteration involving the `min(u)`. To make that possible, Armijo line search was implemented.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%   lpsvm when m>=n                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [w, gamma, y, obj, uold] = solve_chunk(A,d,nu,delta,epsi)

alpha = 1;
tol = 1e-6;
maxiter = 1000;

[m,n]=size(A);
en=ones(n,1);
em=ones(m,1);

% initial u
u=ones(m,1);
iter=0;
epsi=epsi*em;
nu=nu*em;
diff=1; umin = min(u);
DA=spdiags(d,0,m,m)*A;
while (diff>tol) & ((iter<maxiter) | min(u) <= -1e-2)
   uold=u;
   iter=iter+1;
   du=d.*u;Adu=A'*du;
   pp=max(Adu-en,0);np=max(-Adu-en,0);
   dd=sum(du)*d;unu=max(u-nu,0);uu=max(-u,0);
   %Gradient
   g=-epsi+(d.*(A*pp))-(d.*(A*np))+dd+unu-alpha*uu;
   %Hessian
   E=spdiags(sqrt(sign(np)+sign(pp)),0,n,n);
   H=[DA*E d];
   f=1./(delta+sign(unu)+alpha*sign(uu));

   F=spdiags(f,0,m,m);gg=f.*g;HT=H';
   di=(eye(n+1)+HT*(F*H))\(HT*gg);
   di=H*di; di=f.*di; di=-gg+di;

   %Armijo step
   lambda = 1;
   pu1 = max(u-nu,0); pu2 = max(-u,0);
   t1 = norm(pp)^2 + norm(np)^2; t2 = norm(pu1)^2+norm(pu2)^2;
   t3 = (-d'*u)^2;
   fu = -epsi*(ones(1,m)*u) + .5*(t1 + t3 + t2);

   u2 = u + lambda*di;
   du2 = d.*u2; Adu2 = A'*du2;
```

```
   pp2 = max(Adu2 - en,0); np2 = max(-Adu2 - en,0);
   pu1 = max(u2-nu,0); pu2 = max(-u2,0);

   t1 = norm(pp2)^2 + norm(np2)^2; t2 = norm(pu1)^2+norm(pu2)^2;
   t3 = (-d'*u2)^2;
   fu2 = -epsi*(ones(1,m)*u2) + .5*(t1 + t3 + t2);

   while(fu -fu2 < (-lambda/4)*g'*di )
     lambda = lambda/2;
     u2 = u + lambda*di;
     du2 = d.*u2; Adu2 = A'*du2;
     pp2 = max(Adu2 - en,0); np2 = max(-Adu2 - en,0);
     pu1 = max(u2-nu,0); pu2 = max(-u2,0);
     fu2 = -epsi*(ones(1,m)*u2) + .5*(norm(pp2)^2 + norm(np2)^2 + (-d'*u2)^2 + norm(pu1)^2+n
   end
   u = u2;
   umin = min(u);
   %normal step
   %u=u+di;
   diff=norm(g);
   %diff= norm(uold-u);
end

% 'Result LPSVM: lambda and u' lambda u
w = 1/epsi(1)*(pp-np);
gamma = -(1/epsi(1))*sum(du);
y = 1/epsi(1)*max(uold - nu,0);
obj = nu'*y + ones(1,n)*(pp+np);
return;
```

The second subroutine `CHKlpsvm` is the driver implementing the chunking of the observations of the data set including observations of the last chunk which are violating constraints.

```
function [w,gamma]=CHKlpsvm(A,d,nu,delta,epsi,numChunks)

savedAd = [];
[sm sn]=size(A);
Ad = [A d];

nrad = size(Ad,1);
'Size Ad='
nrad

%chunk up A and d
```

```
k = numChunks;  % number of pieces
indx = [0:k];
indx = floor(sm*indx/k);    %last row numbers for all 'segments'

'Block: k,indx'
k
indx

% Cycle with i mod k
eqCount = 0;
oldObj = -inf;
i = 1;
objTrack = [];
while eqCount < 3
   ii = mod(i-1, k) + 1;
   AAdd = Ad(indx(ii)+1:indx(ii+1),:);

   if size(savedAd,1) > 0
       AAdd = union(AAdd,savedAd,'rows');
       s2 = size(AAdd,1);
       'Size AAdd='
       s2
   end

   [w gamma y obj u] = solve_chunk(AAdd(:,1:sn),AAdd(:,sn+1),nu,delta,epsi);

%determines which constraints to keep for next chunk
   ww = AAdd(:,sn+1).*(AAdd(:,1:sn)*w-gamma);
   toSave = find(ww+y-1 < 1e-2 | (y>0));

   'toSave'
   toSave

   savedAd = AAdd(toSave,:);

   pct = abs(oldObj-obj)/max(oldObj,obj);
   'Iter, pct='
   i
   pct
   mypct = (obj-oldObj)/max(oldObj,obj);

   if pct < 0.001
       eqCount = eqCount + 1;
   else
       oldObj = obj;
       eqCount = 0;
```

```
   end
   i = i + 1;
end
return;
```

7.2 CMAT Code

The corresponding CMAT listings are the following:

```
  /* CMAT Formulation of LP_SVM Algorithm:
     similar to tsvm18.inp: for Nobs > nvar: with SMW */

  function lpsvchnk(A,y,nu,delta,alf,eps1,armlin) {
   m = nrow(A); n = ncol(A); np1 = n + 1;
   rnu = 1. / nu; tol = 1.e-6; maxi = 1000;
   ubnd = -.01;
   epsm = cons(m,1,eps1); num = cons(m,1,nu);
   /* print "Size A:", size(A);
      print "Size y:", size(y); */

   /* Starting u[m] */
   u = cons(m,1,1.); umin = 1.;
   D = diag(y); DA = D * A;

   /* Prepare F and G */
   du = D * u; Adu = A‘ * du;
   pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
   dsum = du[+]; dd = dsum * y;
   unu = max(u-num,0.); uu = max(-u,0.);

   /* Function */
   t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf*uu‘*uu;
   t3 = dsum * dsum;
   crit = -eps1*u[+] + .5 * (t1 + t3 + t2);
   /* print "t1,t2,t3,crit=",t1,t2,t3,crit; */

   /* Gradient */
   t1 = y .* (A*pp); t2 = y .* (A*np);
   grd = -epsm + t1 - t2 + dd + unu - alf * uu;
   gnrm = norm(grd);
   /* print "gnrm=",gnrm," grad=",grd; */

   iter = 0;
   while (++iter <= maxi && (gnrm > tol || umin <= ubnd)) {
```

```
uold = u; cold = crit;
/* Hessian */
e = sqrt(sign(np) + sign(pp));
E = diag(e);
B = DA * E; H = [ B y ];
/* that's different */
f = delta + sign(unu) + alf*sign(uu);
f = 1. / f;
F = diag(f);
gg = f .* grd; b = H‘ * gg;
C = H‘ * F * H + ide(np1);

di = C \ b;
di = f .* (H * di); di -= gg;
u = uold + di;

/* Prepare */
du = D * u; Adu = A‘ * du;
pp = max(Adu-1.,0.); np = max(-Adu-1.,0.);
dsum = du[+]; dd = dsum * y;
unu = max(u-num,0.); uu = max(-u,0.);

/* Function */
t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf * uu‘*uu;
t3 =  dsum * dsum;
crit = -eps1*u[+] + .5 * (t1 + t3 + t2);
/* print "t1,t2,t3=",t1,t2,t3;
   print "crit=",crit," di=",di; */

/* Armijo line search for beta */
beta = 1.;
if (armlin && crit >= cold) {
  beta = .5;
  while (beta > 1.e-12) {
    u = uold + beta * di;
    du = D * u; Adu = A‘ * du;
    pp = max(Adu-1,0.); np = max(-Adu-1.,0.);
    dsum = du[+]; dd = dsum * y;
    unu = max(u-num,0.); uu = max(-u,0.);
    /* Function */
    t1 = pp‘*pp + np‘*np; t2 = unu‘*unu + alf * uu‘*uu;
    crit = -eps1*u[+] + .5 * (t1 + dsum*dsum + t2);
    if (crit < cold) break;
    beta *= .5;
} }
```

```
    /* Gradient */
    t1 = y .* (A*pp); t2 = y .* (A*np);
    grd = -epsm + t1 - t2 + dd + unu - alf * uu;
    gnrm = norm(grd); umin = min(u);
    /* print "Iter=",iter," Crit=",crit," Gnrm=",gnrm,
          " umin=",umin," Beta=",beta; */
    /* print "grad=",grd; */
   }

   /* final weights and intercept gamma */
   w = (pp - np) / eps1;
   gamma = -du[+] / eps1;
   yv = max(u-num,0) / eps1;
   vc = pp + np;
   crit = num' * yv + vc[+];
   print "  RESULT LPSVM: gamma=",gamma," Crit=",crit," Beta=",beta;
   print "  Niter=",iter," gnrm=",gnrm," umin=", umin;
   print "Coeff w=",w;
   /* print "Yv=",yv; */
   return(w,gamma,yv,crit);
  }

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

  /* CMAT Formulation of ChunkingSVM Algorithm:
     by Michael Thompson and Olvi Mangasarian
     for Nobs > nvar: with SMW */

  function SVMchunk(A,y,nu,delta,alf,eps1,nchk,prec) {
   armlin = 1;
   m = nrow(A); n = ncol(A); np1 = n + 1;
   rnu = 1. / nu; tol = .001; maxi = 1000;
   epsm = cons(m,1,eps1); num = cons(m,1,nu);

   k = nchk; indx = [ 0:k ]; rm = (real)m;
   indx = floor(rm * indx / k);
   print "INDX=", indx;

   ecnt = 0; cold = macon("mbig");
   iter = 0; ncon = 0;
   free cind,ind2;
   while (ecnt < 3) {
     ic = iter % nchk + 1;
     i1 = indx[ic]+1; i2 = indx[ic+1];
     ind1 = [ i1 : i2 ]';
```

```
    Ach = A[ind1,]; dch = y[ind1];
    if (ncon) {
      free ind2; k = 0;
      for (i = 1; i <= ncon; i++) {
        id = cind[i];
        if (id < i1 || id > i2) ind2[++k] = id;
      }
      if (k) {
        Ach = Ach |> A[ind2,];
        dch = dch |> y[ind2];
        ind1 = ind1 |> ind2;
        /* print "Ind1=",ind1; print "Ind2=", ind2; */
    } }
    mch = nrow(Ach);
    print "Iter=", iter," ncon=",ncon," size=",mch;

    < w,gamma,yv,crit > =
               lpsvchnk(Ach,dch,nu,delta,alf,eps1,armlin);

    ww = diag(dch) * (Ach * w - gamma);
    /* print "ww=",ww; */
    vcon = ww + yv - 1.;
    ncon = 0; free cind;
    for (i = 1; i <= mch; i++)
    if (vcon[i] < 1.e-2 || yv[i] > 0.) cind[++ncon] = ind1[i];

    dm = 1. / max(cold,crit);
    pct = dm * abs(cold - crit);
    print "  End Outer Iter: Pct=",pct," new ncon=",ncon;
    if (pct < prec) ecnt++;
    else { ecnt = 0; cold = crit; }
    iter++; if (iter == maxi) break;
  }
  return(w,gamma);
}

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

print "Problem HEART from Statlog collection";
print "Michie, Spiegelhalter, & Taylor, 1994";
data = rspfile("..\\tdata\\heart_scale.dat");

x = data[,2:14]; y = data[,1];
eps1 = .001; nu = 1.; nchk = 4;
alfa = 1.; delta = .001; /* default setting */
```

```
prec = .001;
print " CALL of SVMchunk: prec=", prec, " nchk=",nchk;
print " delta=", delta," alfa=", alfa;
< w, gamma > = SVMchunk(x,y,nu,delta,alfa,eps1,nchk,prec);

print "SVMchunk Result: Gamma=", gamma;
print "SVMchunk Result: weights=", w;
```

8 SCAD SVM and SCAD LSE

Here, in our CMAT implementation, as in some of Olvi's implementations, the Sherman - Morrison - Woodbury formula (Golub & Van Loan, 1989, p. 51) is used for the solution of linear systems with large coefficient matrices:

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U})\mathbf{V}^T\mathbf{A}^{-1},$$

where $\mathbf{U}, \mathbf{V} \in \mathcal{R}^{n\times p}$ and $\mathbf{A} \in \mathcal{R}^{n\times n}$ and all inverted matrices are nonsingular. In many applications, the formula is applied when $n \gg p$ and where $\mathbf{A}$ is a large nonsingular diagonal matrix and $\mathbf{U}\mathbf{V}^T$ is a very large $n \times n$ nonsingular matrix. The right side of the expression contains the easy inversion of the diagonal matrix $\mathbf{A}$ and the inversion of the "small" $p \times p$ matrix $\mathbf{I} + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U}$.

8.1 Helen Zhang's Matlab Code

```
function [w, b, xind] = scadsvc(xtrain, ytrain, a, lambda, tol)
% this implements SCAD SVM classification
% Input:
%   xtrain : n-by-d data matrix to train
%   ytrain : column vector of target {-1, 1}'s
%   a : tuning parameter in scad function
%       (default: 3.7 or whatever the paper uses)
%   lambda : tuning parameter in scad function
%       (default : 2 or whatever the paper uses)
%   tol: the cut-off value to be taken as 0
% Output:
%   w : direction vector of separating hyperplane
%   b : the bias
%   Ind : Indices of remained variables
% You have to have SVM software of OSU to run this program

if nargin < 2;
    disp('not enough input');
```

```
elseif nargin == 2;
    a = 3.7; lambda = 1; tol= 10^(-6);
elseif nargin == 3;
    lambda = 2; tol= 10^(-4);
elseif nargin == 4;
    tol= 10^(-4);
end;

[AlphaY, SVs, Bias, Parameters, nSV, nLabel] = ...
   LinearSVC(xtrain', ytrain');
d = size(xtrain,2);
w = sum(vec2matSM(AlphaY,d) .* SVs,2) ;
b = Bias;
diff = 1000;
ntrain = size(xtrain,1);
xind = 1:d;
while diff > tol;
    x = [ones(ntrain,1) xtrain];
    y1 = ytrain;
    y0 = y1./abs(y1 - x * [b ; w]);
    sgnres = y1 -x*[b;w];
    res = abs(y1 - x * [b ; w]);
    D = 1/(2*ntrain)*diag(1./res) ;
    aw = abs(w);
    dp = zeros(size(xtrain, 2), 1);
    dp = lambda*(aw<=lambda)+
         (a*lambda-aw)/(a-1).*(aw>lambda&aw<=a*lambda);
    Q1 = diag([0; dp./aw]);
    Q = x'* D * x + Q1;
    P = 0.5*(y1 + y0)' * x /ntrain;
    nwb = pinv(Q) * P';
    nw = nwb(2:end);
    nb = nwb(1);
    diff = norm(nwb - [b;w]);
    ind = abs(nw)>0.001;
    if (sum(ind)>0)
      w = nw(ind);
      xtrain = xtrain(:,ind);
      xind = xind(ind);
      b = nb;
    else
      diff=tol/2;
    end
end;
    ind = abs(nw)>0.001;
    if (sum(ind)>0)
```

```
  w = nw(ind);
 else
  w = zeros(size(xtrain,2),1);
 end
 b = nb;
 f = xtrain*w+b;
 xqx = 0.5*x*pinv(Q)*x';
```

8.2 CMAT Code for SCAD SVM

The following CMAT implementation shows the modification of the algorithm for $nvar >> Nobs$ using the Sherman-Morrison-Woodbury formula.

```
function scadsvm(X,y,a,lamb,tol,bw0,smw) {
/* this implements SCAD SVM classification
 % Input:
 %   xtrain[m,n] : m-by-n data matrix to train
 %   ytrain[m]   : column vector of target {-1, 1}'s
 %   a : tuning parameter in scad function
 %       (default: 3.7 or whatever the paper uses)
 %   lambda : tuning parameter in scad function
 %       (default : 2 or whatever the paper uses)
 %   tol: the cut-off value to be taken as 0
 %   bw0[np1]: initial values
 %   smw : use Sherman-Morrison-Woodbury
 % Output:
 %   b    : the bias
 %   w[n] : direction vector of separating hyperplane
 %   ind  : Indices of remained variables */

 m = nrow(X); n = ncol(X); np1 = n + 1;
 if (a == .) a = 3.7;
 if (lamb == .) lamb = 1.;
 if (tol == .) tol = 1.e-4;
 if (smw == .) smw = (m < n) ? 1 : 0;
 /* delta is threshold for zero weight */
 delta = 1.e-6;

 /* Initial values should be supplied */
 if (bw0 == .) {
   b = 0.; w = cons(n,1,1.);
 } else {
   nr = nrow(bw0); nc = ncol(bw0);
```

```
  nmin = min(nr,nc); nmax = max(nr,nc);
  if (nmin != 1 || nmax != np1)
  print "Error dimension of initial estimates",nmin,nmax;
  b = bw0[1]; w = bw0[2:np1];
}
print "Initial: b=", b, " w=", w;
icpt = cons(m,1,1.);
tf1 = .5 / m;

xind = [ 1:n ];
diff = tol + 1.; iter = 0;
while (diff > tol) {
  iter++; nsel = ncol(xind);
  Xtr = icpt -> X[,xind];
  nc = ncol(Xtr);
  bw = b |> w;
  sres = y - Xtr * bw; ares = abs(sres);
  ss = ssq(sres);
  print "iter=", iter, " sres=", ss, " nsel=", nsel;
  y0 = y ./ ares;
  D = tf1 * diag(1. / ares);
  aw = abs(w);
  c1 = (aw <= lamb);
  c2 = (aw > lamb && aw <= a*lamb);
  dp1 = c1 .* lamb;
  dp2 = c2 .* ((a*lamb-aw) / (a-1.));
  dp = dp1 + dp2;
  q1 = 0. |> (dp ./ aw);
  /* print "Ridge=", q1; */
  P = tf1 * (y + y0)‘ * Xtr;
  /* print "RHS grad=", P; */
  qmin = q1[><]; qmax = q1[<>];
  if (smw && qmax > 0.) {
    /* use Sherman-Morrison-Woodbury */
    thr = 1.e-6 * qmax;
    /* print "qmin=", qmin, " qmax=", qmax," thr=", thr; */
    if (qmin < thr) {
      qloc = loc(q1,"small",thr);
      q1[qloc] = thr;
    }
    Qd = inv(diag(q1));
    Dsq = sqrt(D); DX = Dsq * Xtr;
    /* Coef is small m by m matrix */
    Coef = ide(m) + DX * Qd * DX‘;
    Qinv = Qd - Qd * DX‘ * inv(Coef) * DX * Qd;
  } else {
```

```
    /* original Zhang algorithm:
       Q is np1 by np1 matrix */
    Q = Xtr‘ * D * Xtr + diag(q1);
    Qinv = pinv(Q);
  }
  nbw = Qinv * P‘;
  /* print "iter=", iter, " nbw=", nbw; */

  nw = nbw[2:nc]; nb = nbw[1];
  diff = norm(nbw - bw);
  ind = loc(abs(nw) > delta);
  /* print "New index vector=", ind; */
  nsel = nrow(ind);
  if (nsel <= 1) break;
  w = nw[ind]; b = nb;
  xind = xind[ind];
  print "iter=", iter, " diff=", diff;
}
print "Convergence after", iter, " iterations.";
print "xind=", xind;
b = nb;
if (ind != .) {
  w = nw[ind]; xind = xind[ind];
  Xtr = icpt -> X[,xind];
  bw = b |> w;
  ares = abs(y - Xtr * bw);
  aw = abs(w);
  c1 = (aw <= lamb);
  c2 = (aw > lamb && aw <= a*lamb);
  dp1 = c1 .* lamb;
  dp2 = c2 .* ((a*lamb-aw) / (a-1.));
  dp = dp1 + dp2;
  q1 = 0. |> (dp ./ aw);
  D = tf1 * diag(1. / ares);
  Qinv = inv(Xtr‘ * D * Xtr + diag(q1));
  xpx = .5 * Xtr * Qinv * Xtr‘;
}
print "Return: ind=", ind, " bias=", b, " weights=", w;
return(b,w,xind);
}
```

First an example with binary response variable y where $Nobs < nvar$:

```
print "NIR Spectra data set: train: nr=21, test: nr=7; nvar=268";
```

```
  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.)';
  m = nrow(xtrn); n = ncol(xtrn);


  /* Original Zhang algorithm */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 0;
  bw0 = 0. |> cons(n,1,1.);
  < bias, w, ind > = scadsvm(xtrn,y,a,lamb,tol,bw0,smw);
  print "SCAD Result: bias=", bias;
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;


iter= 1 sres= 3423907.8 nsel= 268
iter= 1 diff= 18.455
iter= 2 sres= 7756.13 nsel= 268
iter= 2 diff= 8.0250
iter= 3 sres= 27.514 nsel= 228
iter= 3 diff= 0.2036
iter= 4 sres= 33.436 nsel= 206
iter= 4 diff= 0.04183
iter= 5 sres= 34.597 nsel= 179
iter= 5 diff= 0.06341
iter= 6 sres= 34.378 nsel= 157
iter= 6 diff= 0.002151
iter= 7 sres= 33.882 nsel= 137
iter= 7 diff= 0.01831
iter= 8 sres= 32.600 nsel= 126
iter= 8 diff= 0.01566
iter= 9 sres= 30.585 nsel= 118
iter= 9 diff= 0.01326
iter= 10 sres= 27.986 nsel= 108
iter= 10 diff= 0.01911
iter= 11 sres= 25.257 nsel= 98
iter= 11 diff= 0.03695
iter= 12 sres= 22.960 nsel= 90
iter= 12 diff= 0.03682
iter= 13 sres= 21.644 nsel= 82
```

```
iter= 13 diff= 0.01581
iter= 14 sres= 21.206 nsel= 77
iter= 14 diff= 0.04390
iter= 15 sres= 20.965 nsel= 69
iter= 15 diff= 0.01754
iter= 16 sres= 20.943 nsel= 62
iter= 16 diff= 0.01096
iter= 17 sres= 20.946 nsel= 55
iter= 17 diff= 0.006947
iter= 18 sres= 20.949 nsel= 44
iter= 18 diff= 0.002086
iter= 19 sres= 20.951 nsel= 32
iter= 19 diff= 0.002498
iter= 20 sres= 20.953 nsel= 25
iter= 20 diff= 0.003360
iter= 21 sres= 20.953 nsel= 4
Convergence after 21 iterations.


xind=
   |    1    2    3    4
------------------------
 1 |   86   87   88   89


Return: ind= .  bias=-0.05083 weights=
   |            1
-----------------
 1 |  -0.00000107
 2 |  -0.00000119
 3 |  -0.00000120
 4 |  -0.00000108


  /* Sherman-Morrison-Woodbury modification */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 1;
  bw0 = 0. |> cons(n,1,1.);
  < bias, w, ind > = scadsvm(xtrn,y,a,lamb,tol,bw0,smw);
  print "SCAD Result: bias=", bias;
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;


iter= 1 sres= 3423907.8 nsel= 268
```

```
iter= 1 diff= 18.460
iter= 2 sres= 7761.26 nsel= 268
iter= 2 diff= 7.9807
iter= 3 sres= 28.391 nsel= 226
iter= 3 diff= 0.7359
iter= 4 sres= 32.928 nsel= 175
iter= 4 diff= 0.1695
iter= 5 sres= 37.049 nsel= 120
iter= 5 diff= 0.02424
iter= 6 sres= 38.440 nsel= 92
iter= 6 diff= 0.003300
iter= 7 sres= 38.978 nsel= 79
iter= 7 diff= 0.001329
iter= 8 sres= 39.242 nsel= 63
iter= 8 diff= 0.001694
iter= 9 sres= 39.403 nsel= 27
iter= 9 diff= 0.001972
iter= 10 sres= 39.519 nsel= 11
iter= 10 diff= 0.001906
iter= 11 sres= 39.609 nsel= 8
iter= 11 diff= 0.001680
iter= 12 sres= 39.680 nsel= 6
iter= 12 diff= 0.001419
iter= 13 sres= 39.738 nsel= 4
iter= 13 diff= 0.001178
iter= 14 sres= 39.786 nsel= 2
Convergence after 14 iterations.


xind=
   |    1    2
--------------
 1 |   18   19


Return: ind= .  bias=-0.9956 weights=
   |            1
-----------------
 1 |   0.00000125
 2 |   0.00000116
```

The results are different.

Next an example with binary response variable y where $Nobs > nvar$:

```
  print "Problem HEART from Statlog collection";
  print "Michie, Spiegelhalter, & Taylor, 1994";
  data = rspfile("..\\tdata\\heart_scale.dat");
  x = data[,2:14]; y = data[,1];
  m = nrow(x); n = ncol(x);


  /* Original Zhang algorithm */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 0;
  bw0 = 0. |> cons(n,1,1.);
  < bias, w, ind > = scadsvm(x,y,a,lamb,tol,bw0,smw);
  print "SCAD Result: bias=", bias;
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;
```

```
iter= 1 sres= 3795.90 nsel= 13
iter= 1 diff= 3.2056
iter= 2 sres= 138.152 nsel= 13
iter= 2 diff= 0.2735
iter= 3 sres= 150.189 nsel= 13
iter= 3 diff= 0.1912
iter= 4 sres= 170.545 nsel= 13
iter= 4 diff= 0.2912
iter= 5 sres= 222.678 nsel= 12
iter= 5 diff= 0.2507
iter= 6 sres= 289.788 nsel= 10
iter= 6 diff= 0.1765
iter= 7 sres= 349.558 nsel= 10
iter= 7 diff= 0.1159
iter= 8 sres= 394.786 nsel= 9
iter= 8 diff= 0.07368
iter= 9 sres= 426.130 nsel= 9
iter= 9 diff= 0.04581
iter= 10 sres= 446.672 nsel= 7
iter= 10 diff= 0.02807
iter= 11 sres= 459.659 nsel= 5
iter= 11 diff= 0.01704
iter= 12 sres= 467.686 nsel= 5
iter= 12 diff= 0.01028
iter= 13 sres= 472.581 nsel= 2
iter= 13 diff= 0.006186
iter= 14 sres= 475.541 nsel= 2
iter= 14 diff= 0.003715
iter= 15 sres= 477.324 nsel= 2
```

```
iter= 15 diff= 0.002229
iter= 16 sres= 478.395 nsel= 2
Convergence after 16 iterations.


xind=
   |    1    2
--------------
 1 |    9   13

Return: ind= 2 bias=-0.9980 weights= 0.00001236


  /* Sherman-Morrison-Woodbury modification */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 1;
  bw0 = 0. |> cons(n,1,1.);
  < bias, w, ind > = scadsvm(x,y,a,lamb,tol,bw0,smw);
  print "SCAD Result: bias=", bias;
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;


iter= 1 sres= 3795.90 nsel= 13
iter= 1 diff= 3.2056
iter= 2 sres= 138.159 nsel= 13
iter= 2 diff= 0.2730
iter= 3 sres= 150.164 nsel= 13
iter= 3 diff= 0.1899
iter= 4 sres= 170.390 nsel= 13
iter= 4 diff= 0.2701
iter= 5 sres= 218.677 nsel= 12
iter= 5 diff= 0.1110
iter= 6 sres= 248.453 nsel= 11
iter= 6 diff= 0.09504
iter= 7 sres= 253.867 nsel= 10
iter= 7 diff= 0.06989
iter= 8 sres= 258.513 nsel= 9
iter= 8 diff= 0.1765
iter= 9 sres= 278.898 nsel= 9
iter= 9 diff= 0.02414
iter= 10 sres= 277.918 nsel= 8
iter= 10 diff= 0.003386
iter= 11 sres= 278.832 nsel= 6
iter= 11 diff= 0.008500
iter= 12 sres= 278.352 nsel= 3
```

```
iter= 12 diff= 0.1442
iter= 13 sres= 300.791 nsel= 3
iter= 13 diff= 0.01843
iter= 14 sres= 304.541 nsel= 3
iter= 14 diff= 0.1406
iter= 15 sres= 281.449 nsel= 2
iter= 15 diff= 0.1716
iter= 16 sres= 311.173 nsel= 2
iter= 16 diff= 0.1103
iter= 17 sres= 338.663 nsel= 2
iter= 17 diff= 0.02943
iter= 18 sres= 347.111 nsel= 2
iter= 18 diff= 0.07494
iter= 19 sres= 326.539 nsel= 2
Convergence after 19 iterations.


xind=
   |    1    2
--------------
 1 |    9   13

Return: ind= 2 bias=-0.3946 weights= 0.000008709
```

The results are similar.

8.3 CMAT Code for SCAD Least Squares Regression

The following CMAT code shows a version of the SCAD method for the common least squares regression problem and includes the modification for $nvar >> Nobs$ using the Sherman-Morrison-Woodbury formula.

```
function scadlse(X,y,a,lamb,tol,w0,smw) {
/* this implements SCAD SVM classification
 % Input:
 %   xtrain[m,n] : m-by-n data matrix to train
 %   ytrain[m]   : column vector of target
 %   a : tuning parameter in scad function
 %       (default: 3.7 or whatever the paper uses)
 %   lambda : tuning parameter in scad function
 %       (default : 2 or whatever the paper uses)
 %   tol: the cut-off value to be taken as 0
 %   w0[np1]: initial values for p weights
```

```
%   smw : use Sherman-Morrison-Woodbury
% Output:
%   w[n] : estimates of linear model
%   ind  : Indices of remained variables */

m = nrow(X); n = ncol(X); np1 = n + 1;
if (a == .) a = 3.7;
if (lamb == .) lamb = 1.;
if (tol == .) tol = 1.e-4;
if (smw == .) smw = (m < n) ? 1 : 0;
/* delta is threshold for zero weight */
delta = 1.e-3;

/* Initial values should be supplied */
if (w0 == .) {
  w = cons(n,1,1.);
} else {
  nr = nrow(w0); nc = ncol(w0);
  nmin = min(nr,nc); nmax = max(nr,nc);
  if (nmin != 1 || nmax != n)
  print "Error dimension of initial estimates",nmin,nmax;
  w = w0;
}
/* print "Initial: w=", w; */
tf1 = .5 / m;

xind = [ 1:n ];
diff = tol + 1.; iter = 0;
while (diff > tol) {
  iter++;
  Xtr = X[,xind];
  nc = ncol(Xtr); nsel = ncol(xind);
  sres = y - Xtr * w; ss = ssq(sres);
  aw = abs(w);
  c1 = (aw <= lamb);
  c2 = (aw > lamb && aw <= a*lamb);
  dp1 = c1 .* lamb;
  dp2 = c2 .* ((a*lamb-aw) / (a-1.));
  dp = dp1 + dp2;
  tp =  m * (dp .* aw);
  crit = .5 * ss + tp[+];
  print "iter=", iter, " sres=", ss,
        " crit=", crit, " nsel=", nsel;
  q1 = m * (dp ./ aw);
  /* print "Ridge=", q1; */
  f = m * sign(w);
```

```
    P = f .* dp - Xtr‘ * sres;
    qmin = q1[><]; qmax = q1[<>];
    if (smw && qmax > 0.) {
      /* use Sherman-Morrison-Woodbury */
      thr = 1.e-6 * qmax;
      print "qmin=", qmin, " qmax=", qmax," thr=", thr;
      if (qmin < thr) {
        qloc = loc(q1,"small",thr);
        q1[qloc] = thr;
      }
      Qd = inv(diag(q1));
      /* Coef is small m by m matrix */
      Coef = ide(m) + Xtr * Qd * Xtr‘;
      Qinv = Qd - Qd * Xtr‘ * inv(Coef) * Xtr * Qd;
    } else {
      /* original SCAD algorithm:
         Q is n by n matrix */
      Q = Xtr‘ * Xtr + diag(q1);
      Qinv = pinv(Q);
    }
    sdir = Qinv * P;
    nw = w - sdir;
    diff = norm(sdir);
    ind = loc(abs(nw) > .001);
    /* print "New index vector=", ind; */
    if (ind == .) break;
    w = nw[ind];
    xind = xind[ind];
    print "iter=", iter, " diff=", diff;
  }
  print "Convergence after", iter, " iterations.";
  print "Result: xind=", xind;
  if (ind != .) {
    w = nw[ind]; xind = xind[ind];
    Xtr = X[,xind];
    yhat = Xtr * w;
    xpx = .5 * Xtr * Qinv * Xtr‘;
  }
  print "Return: ind=", ind, " weights=", w;
  return(w,xind);
 }


 print "SCAD LSE applied to data with Nobs < nvar";
/*  (1) NIR Spectra data set: train: nr=21, test: nr=7   */
```

```
 /*      nvar= 268   (use median of y for binary target)  */

  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;
  x = xtrn; y = ytrn`;
  m = nrow(x); n = ncol(x);
  /* print "ytrn=", ytrn; */



  /* Sherman-Morrison-Woodbury modification */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 1;
  w0 = cons(n,1,1.);
  < w, ind > = scadlse(x,y,a,lamb,tol,w0,smw);
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;


SCAD LSE applied to data with Nobs < nvar
NIR Spectra data set: train: nr=21, test: nr=7
nrtrn,nctrn= 21 268


iter= 1 sres= 2882494.4 crit= 1446875.2 nsel= 268
qmin= 21.000 qmax= 21.000 thr= 0.00002100
iter= 1 diff= 20.073
iter= 2 sres= 906.157 crit= 3435.19 nsel= 268
qmin= 8.6123 qmax= 5412.23 thr= 0.005412
iter= 2 diff= 3.5654
iter= 3 sres= 924.110 crit= 2792.90 nsel= 263
qmin= 2.2805 qmax= 17836.75 thr= 0.01784
iter= 3 diff= 5.8505
iter= 4 sres= 722.391 crit= 2080.75 nsel= 242
qmin= 0.0000 qmax= 20910.98 thr= 0.02091
iter= 4 diff= 32.406
iter= 5 sres= 190.093 crit= 1023.92 nsel= 159
qmin= 0.0000 qmax= 20036.93 thr= 0.02004
iter= 5 diff= 32.441
iter= 6 sres= 41.685 crit= 32.039 nsel= 87
qmin= 0.0000 qmax= 20420.12 thr= 0.02042
iter= 6 diff= 13.153
iter= 7 sres= 34.343 crit= 17.172 nsel= 7
```

```
iter= 7 diff= 731.662
iter= 8 sres= 10.141 crit= 5.0704 nsel= 7
iter= 8 diff= 0.00005048
Convergence after 8 iterations.


Result: xind=
   |    1    2    3    4    5    6    7
---------------------------------------
 1 |   18   19   52   53   71   72   73


 weights=
   |         1
--------------
 1 |    58.929
 2 |   -66.630
 3 |   -61.980
 4 |    28.193
 5 |   -233.89
 6 |    620.73
 7 |   -315.42
```

```
  /* Original algorithm */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 0;
  w0 = cons(n,1,1.);
  < w, ind > = scadlse(x,y,a,lamb,tol,w0,smw);
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;
```

```
iter= 1 sres= 2882494.4 crit= 1446875.2 nsel= 268
iter= 1 diff= 20.073
iter= 2 sres= 906.157 crit= 3435.19 nsel= 268
iter= 2 diff= 3.5654
iter= 3 sres= 924.110 crit= 2792.90 nsel= 263
iter= 3 diff= 5.8505
iter= 4 sres= 722.391 crit= 2080.75 nsel= 242
iter= 4 diff= 716.620
iter= 5 sres= 178.542 crit= 973.104 nsel= 150
iter= 5 diff= 1012.30
```

```
iter= 6 sres= 10.044 crit= 9.8968 nsel= 60
iter= 6 diff= 8.7336
iter= 7 sres= 10.142 crit= 5.0709 nsel= 7
iter= 7 diff= 0.1247
iter= 8 sres= 10.141 crit= 5.0704 nsel= 7
iter= 8 diff= 9.194e-009
Convergence after 8 iterations.


Result: xind=
   |    1    2    3    4    5    6    7
---------------------------------------
 1 |   18   19   52   53   71   72   73


 weights=
   |         1
--------------
 1 |    58.929
 2 |   -66.630
 3 |   -61.980
 4 |    28.193
 5 |   -233.89
 6 |    620.73
 7 |   -315.42
```

Both versions show the same iterations and same results.

```
print "SCAD LSE applied to data with Nobs > nvar";
print "Middle-Aged Men in Health Fitness Club";
print "SAS/STAT: Linnerud, NC State University";

fit = [ 191  36  50   5  162   60,   189  37  52   2  110   60,
        193  38  58  12  101  101,   162  35  62  12  105   37,
        189  35  46  13  155   58,   182  36  56   4  101   42,
        211  38  56   8  101   38,   167  34  60   6  125   40,
        176  31  74  15  200   40,   154  33  56  17  251  250,
        169  34  50  17  120   38,   166  33  52  13  210  115,
        154  34  64  14  215  105,   247  46  50   1   50   50,
        193  36  46   6   70   31,   202  37  62  12  210  120,
        176  37  54   4   60   25,   157  32  52  11  230   80,
        156  33  54  15  225   73,   138  33  68   2  110   43 ];
```

```
  x = fit[,1:3]; y = fit[,4];
  m = nrow(x); n = ncol(x);


  /* Original algorithm */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 0;
  w0 = cons(n,1,1.);
  < w, ind > = scadlse(x,y,a,lamb,tol,w0,smw);
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;


SCAD LSE applied to data with Nobs > nvar
Middle-Aged Men in Health Fitness Club
SAS/STAT: Linnerud, NC State University
iter= 1 sres= 1373965.0 crit= 687042.5 nsel= 3
iter= 1 diff= 1.7613
iter= 2 sres= 501.989 crit= 262.048 nsel= 3
iter= 2 diff= 0.1166
iter= 3 sres= 504.459 crit= 260.287 nsel= 3
iter= 3 diff= 0.03890
iter= 4 sres= 505.726 crit= 259.927 nsel= 2
iter= 4 diff= 0.006252
iter= 5 sres= 506.037 crit= 259.910 nsel= 2
iter= 5 diff= 0.001302
iter= 6 sres= 506.108 crit= 259.909 nsel= 2
iter= 6 diff= 0.0002828
iter= 7 sres= 506.123 crit= 259.909 nsel= 2
iter= 7 diff= 0.00006196
Convergence after 7 iterations.


Result: xind=
   |   1   2
------------
 1 |   2   3


 weights=
   |         1
--------------
 1 |  -0.10814
 2 |   0.23416
```

```
  /* Sherman-Morrison-Woodbury modification */
  a = 3.7; lamb = 1.; tol = 1.e-4; smw = 1;
  w0 = cons(n,1,1.);
  < w, ind > = scadlse(x,y,a,lamb,tol,w0,smw);
  print "SCAD Result: weights=", w;
  print "SCAD Result: ind=", ind;
```

```
iter= 1 sres= 1373965.0 crit= 687042.5 nsel= 3
qmin= 20.000 qmax= 20.000 thr= 0.00002000
iter= 1 diff= 1.7613
iter= 2 sres= 501.989 crit= 262.048 nsel= 3
qmin= 75.030 qmax= 912.824 thr= 0.0009128
iter= 2 diff= 0.1166
iter= 3 sres= 504.459 crit= 260.287 nsel= 3
qmin= 81.806 qmax= 3588.66 thr= 0.003589
iter= 3 diff= 0.03890
iter= 4 sres= 505.726 crit= 259.927 nsel= 2
qmin= 83.902 qmax= 174.182 thr= 0.0001742
iter= 4 diff= 0.006252
iter= 5 sres= 506.037 crit= 259.910 nsel= 2
qmin= 85.095 qmax= 182.584 thr= 0.0001826
iter= 5 diff= 0.001302
iter= 6 sres= 506.108 crit= 259.909 nsel= 2
qmin= 85.344 qmax= 184.445 thr= 0.0001844
iter= 6 diff= 0.0002828
iter= 7 sres= 506.123 crit= 259.909 nsel= 2
qmin= 85.399 qmax= 184.855 thr= 0.0001849
iter= 7 diff= 0.00006196
Convergence after 7 iterations.
```

```
Result: xind=
   |   1   2
-----------
 1 |   2   3
```

```
 weights=
   |         1
-------------
 1 |  -0.10814
 2 |   0.23416
```

Both versions show the same iterations and same results.

9 The Bibliography

References

[1] Fung, G. & Mangasarian, O.L. (2000), "Proximal Support Vector Machines", Technical Report, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[2] 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, 2002.

[3] Fung, G. & Mangasarian, O.L. (2003), "A Feature Selection Newton Method for Support Vector Machine Classification", *Computational Optimization and Aplications*, 1-18.

[4] Golub, G., & Van Loan, C.F. (1989), *Matrix Computations*, John Hopkins University Press, 2nd ed., Baltimore, MD.

[5] Lee, Y.-Y. & Mangasarian, O. (1999), *SSVM: A smooth support vector machine*, Technical Report 99-03, Computer Sciences Dept., University of Wisconsin, Madison.

[6] Lee, Y.-Y. & Mangasarian, O. (2000), *RSVM: Reduced support vector machines*, Technical Report 00-07, Computer Sciences Dept., University of Wisconsin, Madison.

[7] Mangasarian, O.L. & Musicant, D.R (2000a), "Active Support Vector Machine Classification", Technical Report 00-04, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[8] Mangasarian, O.L. & Musicant, D.R (2000b), "Lagrangian Support Vector Machines", Technical Report 00-06, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[9] 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.

[10] 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.

[11] Mangasarian, O.L. & Wild, E.W. (2004), "Feature Selection in k-Median Clustering", Technical Report 04-01, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[12] Mangasarian, O.L. & Wild, E.W. (2006), "Feature Selection for Nonlinear Kernel Support Vector Machines", Technical Report 06-03, Data Mining Institute, University of Wisconsin, Madison, Wisconsin.

[13] Zhang, H.H., Ahn, J., Lin, X., and Park, C. (2006), "Gene selection using support vector machines with nonconvex penalty," *bioinformatics*, **22**, pp. 88-95.