Kernel Method Homework 2 (Prof. Anca Ralescu)

Sequential Minimal Optimization

2009/5/5

Chunsheng Fang, Aravind R.

www.VictorFang.com

1. In the first set, the classes will be linearly separable, this means that you can use the kernel $K(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$.

We create a naïve linear separable 2D dataset. Our SMO implementation can successfully classify them using linear kernel, in < 10 seconds.

The result is shown as below, which identify support vectors as green circles.

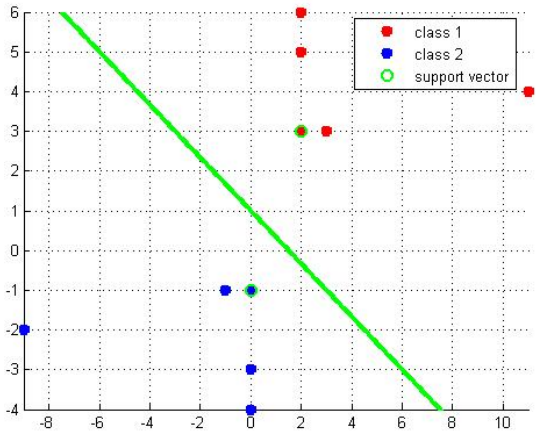


Figure 1. SMO on naïve dataset

2. In the second case I the classes will not be linearly separable. In this case, I will ask you to
   (a) Apply the algorithm with $K(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$, that is, as if they were linearly separable. I will then ask you to discuss the generalization power of the algorithm.
   (b) Apply the algorithm using kernels, for example, $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^d$, $d = 2, 3, 4$. I will then ask you to discuss the behavior of the algorithm for these different values of d.

(a) The SMO is difficult to converge with linear kernel, because it will never find the solution, and here is the result of running for 1 hour.

Objective function = 973.8835.

Training error rate = 35%

Note: there are 96 support vectors, which means that the linear classifier is a combination of almost all dataset.

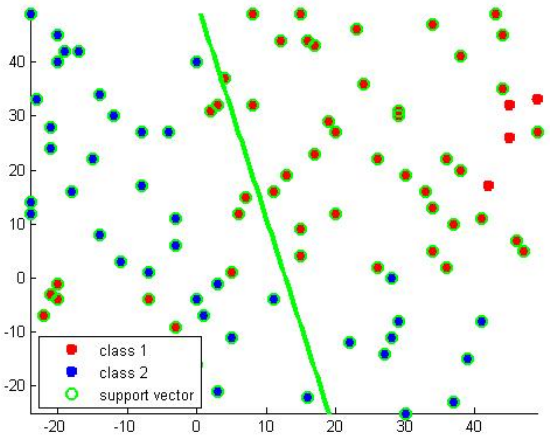


Figure 2. SMO with linear kernel on “simple dataset”

(b) With polynomial kernel, SMO converges quickly, and generate a perfect classification result.

- D = 2;

Here are the results and the kernel plane for d=2; It explain why kernel trick can actually do a better job in classifying nonlinear separable data.

Objective function = 0.0179.

Training error rate = 0%

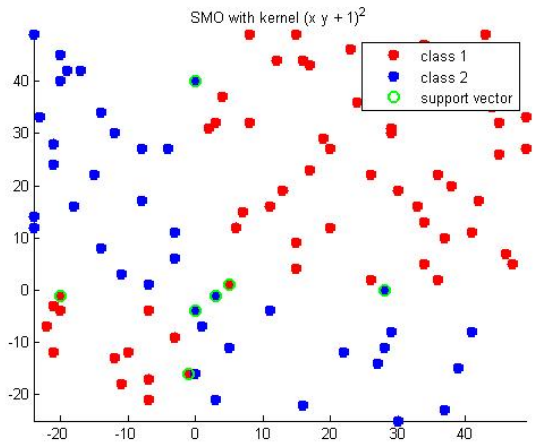


Figure 4. poly kernel d=2

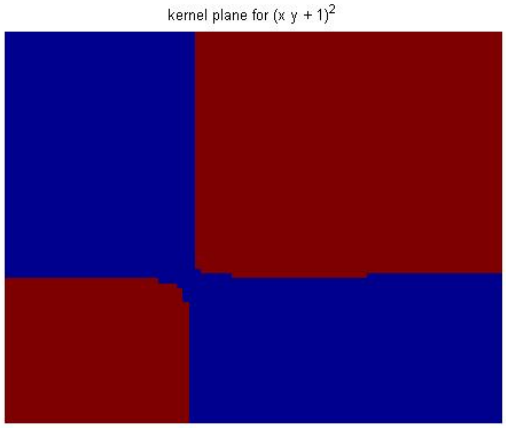


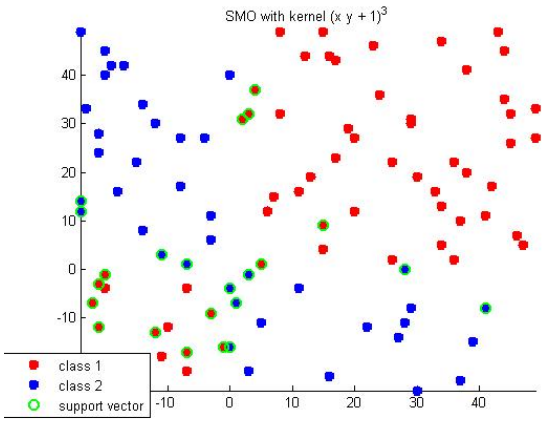
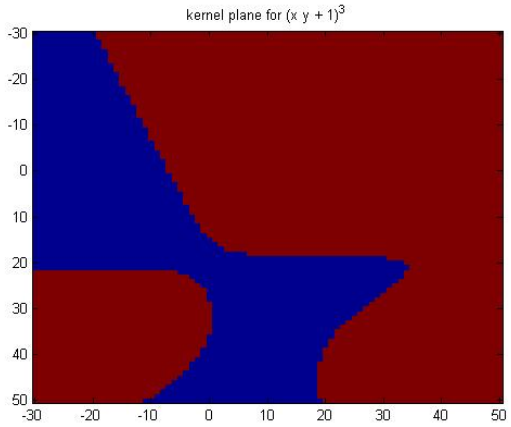
Figure 4. poly kernel d=2, classification plane

- D = 3;

Here are the results and the kernel plane for d=3; Actually it’s not performing as well as d=2, since it’s overfitting the data, and cause an increase in training error rate.

Objective function = 0.028

Training error rate = 0.06

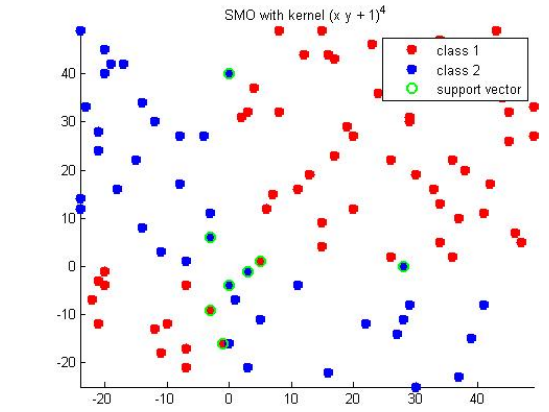
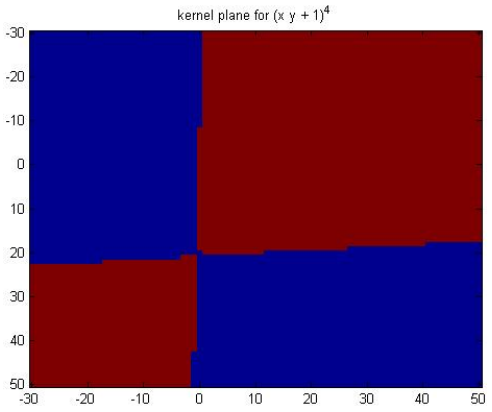
	
poly kernel d=3	classification plane

- D = 4;

Still overfitting.

Objective function = 2.3331e-005

Training error rate = 0.02

	
poly kernel d=4	classification plane

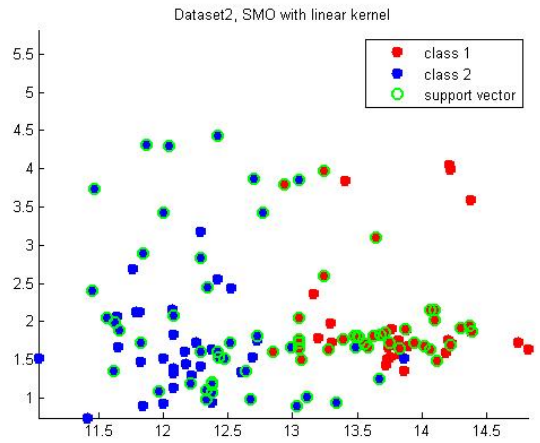
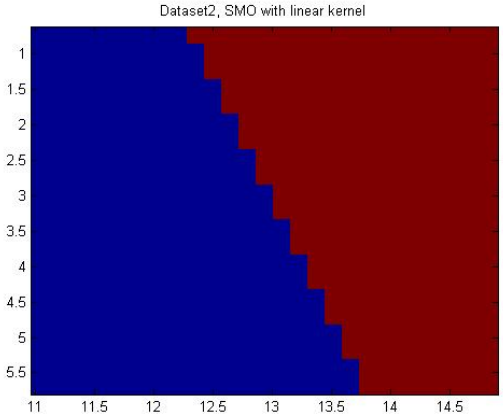
On dataset 2, I run the SMO for 1 minute and force stop. Here are the results.

Since the data has 13 dimensions, I plot the first 2 dimensions.

- Linear kernel:

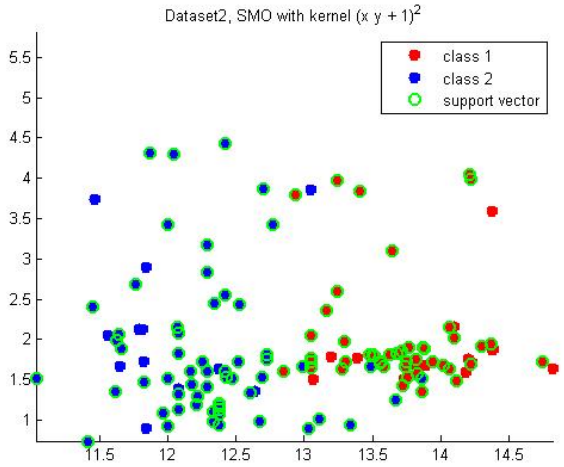
Objective function = 0.5413

error rate = 0.046154

	
SMO with linear kernel on “Dataset 2”, plotting first 2 dimensions.	classification plane

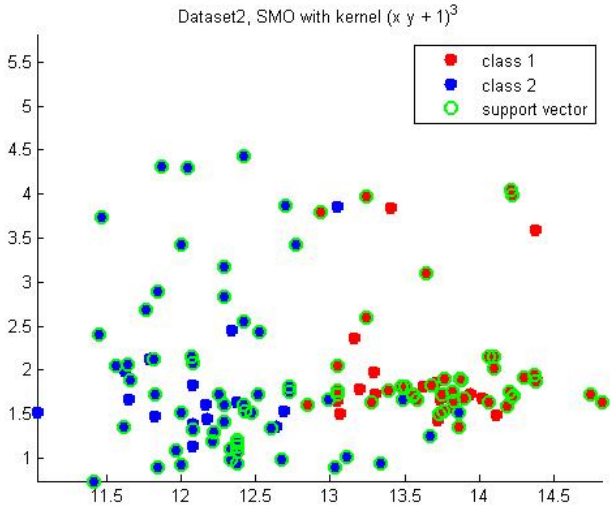
- D = 2

Again, d=2 generates the best error rate, which is about 1.5% error, in just 1 minute runtime.
Objective function = 0.6850
Training error rate = 0.015385



- D = 3

Overfitting.
Objective function = -5.1061e+007
Training error rate = 0.092308

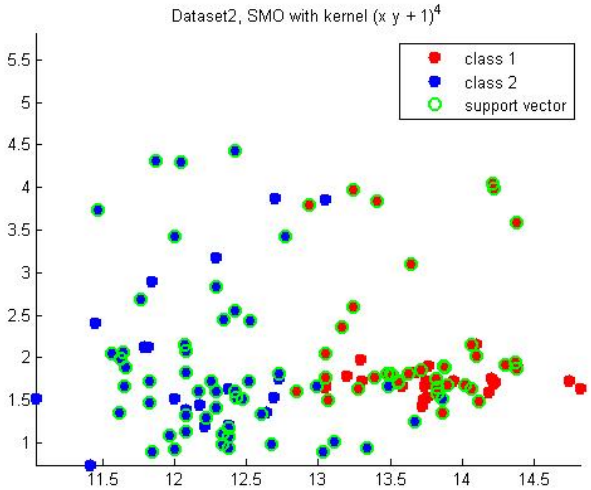


- D = 4

Overfitting.

Objective function = -2.5682e+012

Training error rate = 0.13077



Source code: (For best visualization, the code is runnable for dataset1, d=2)

```
% Sequential Minimal Optimization for SVM
% Chunsheng Fang, Aravind R.
% Univ. of Cincinnati, 20090505
% www.cs.uc.edu/~fangcg
% www.VictorFang.com

clc
clear all
close all
```

```
global C;
global tol; % KKT tolerance, in checkkkt()

C = 10000; tol = 1e-3;

% load DataSet.mat;
% data = data12;

load SimpleDataSet.mat
data = simpleexampledata;

% Test Data
% data = zeros(10,3);
% data(1:5,1:2) = [ 1 2; 2 2; 10 3; 1 4; 1 5];
% data(6:10,1:2) = -data(1:5, 1:2);
% data(1:5,3) = ones(5,1);
% data(6:10,3) = -ones(5,1);
% data(:,1:2) = data(:,1:2) + ones(10,2);

% Extract X and Y from the Training Dataset
numtrain = size(data, 1);
dim = size(data, 2);

% for dataset 2
% x = data(:,2:dim);
% y = data(:,1); % first column is label

% for dataset 1
x = data(:,1:dim-1);
y = data(:,dim); % last column is label

% Initialize alpha(i) to 0
alpha = zeros(numtrain, 1);

% Initialize the b values for each xi
b = 0;

iter = 1;
maxiter = 1000;
inner = 1;
innermax = 100; % force stop if iter haven't increased for long time

% E = zeros(numtrain, 1);

while( iter < maxiter )
```

```
num_chg_alpha = 0;

for i = 1 : numtrain

   % Calculate Ei = f(x(i)) ? y(i)

   fx_i = 0;
   for k = 1:numtrain
      fx_i = fx_i + alpha(k) * y(k) * kernel(x(k,:), x(i,:));
   end

   fx_i = fx_i + b;

   Ei = fx_i - y(i) ;

   % Check KKT

   if ( ( y(i) * Ei < -tol && alpha(i) < C ) || ( y(i) * Ei > tol && alpha(i) > 0 ) )

      % Select j (!= i) randomly
      %           j = i;
      %           while( j == i && j > 0 )
      %               j = rand() * numtrain ;
      %           end
      j = i;
      while(i == j)
         j = round(rand()*numtrain+1);
         if(j > numtrain) j=numtrain;    end;
      end

      disp(['iter = ', num2str(iter), ' ; i = ', num2str(i),' ; j = ', num2str(j)] );


      % Calculate Ej = f(x(j)) ? y(j)

      fx_j = 0;
      for k = 1:numtrain
         fx_j = fx_j + alpha(k) * y(k) * kernel(x(k,:), x(j,:))  ;
      end

      fx_j = fx_j + b;

      Ej = fx_j - y(j) ;

      % Store old values
      aiold = alpha(i);
```

```
ajold = alpha(j);

% Compute L & H
if(y(i)~= y(j))
   L = max(0, ajold - aiold );
   H = min(C, C + ajold-aiold );
else
   L = max(0, ajold + aiold - C);
   H = min(C, ajold + aiold );
end

if( L == H )
   disp(['L == H  jump!! ' num2str(L) ]);
   continue;
end;

% Compute alphaj(new, unclipped)
A = 2 * kernel(x(i,:), x(j,:) ) - kernel( x(j,:), x(j,:) ) - kernel(x(i,:), x(i,:)) ;
if( A >= 0 )
   disp(' A >= 0');
   continue;
end;

ajnew = ajold - ( y(j) * ( Ei - Ej ) / A );

% Compute alphaj(new, clipped)
if(ajnew > H )
   alpha(j) = H;
end

if( ajnew >= L && ajnew <= H )
   alpha(j) = ajnew;
end

if(ajnew < L)
   alpha(j) = L;
end

if( abs( alpha(j) - ajold ) < 1e-5 )
   disp( ' abs( alpha(j) - ajold ) ' );
   continue;
end;

% Compute alpha1(new)
s = y(i) * y(j);
alpha(i) = aiold + s * ( ajold - alpha(j) );
```

```
          % Compute b1 & b2
          b1 = b - Ei - (alpha(i) - aiold)*y(i)*kernel( x(i,:), x(i,:) ) - ( alpha(j) - ajold )
* y(j) * kernel( x(j,:), x(i,:) );
          b2 = b - Ej - ( alpha(i) - aiold ) * y(i) * kernel( x(i,:), x(j,:) ) - ( alpha(j)
- ajold ) * y(j) * kernel( x(j,:), x(j,:) );

          if( 0 < alpha(i) && alpha(i) < C )
             b = b1;
          end
          if( 0 < alpha(j) && alpha(j) < C )
             b = b2;
          else
             b = ( b1 + b2 ) / 2;
          end

          num_chg_alpha = num_chg_alpha + 1 ;

       end


    end

    if( num_chg_alpha == 0 )
       iter = iter + 1 ;
    else
       iter = 1;
    end
end

disp( alpha' );

objfun(alpha, x, y )
plotsvm



% Kernel function, Sequential Minimal Optimization for SVM
% Chunsheng Fang, Aravind r.
% Univ. of Cincinnati, 20090505

function kout = kernel(p1 , p2)

if(size(p1,1)<size(p1,2))
    p1 = p1';
end
```

```
if(size(p2,1)<size(p2,2))
    p2 = p2';
end

% linear kernel
% kout = p1' * p2;


% % polynomial kernel
d = 2;
kout = (p1'*p2 +1)^d;




% Plotting SVM, Sequential Minimal Optimization for SVM
% Chunsheng Fang, Aravind R.
% Univ. of Cincinnati, 20090505

w = zeros(1,dim-1);

tmp = w;

for i = 1:length(alpha)
    tmp = tmp + y(i)*alpha(i)*x(i,:);
end

w = tmp;

%% compute training error rate
errvec = zeros(numtrain, 1);
for j = 1:length(alpha)
    pred = 0;
    fx_j = 0;
    for k = 1:numtrain
        fx_j = fx_j + alpha(k) * y(k) * kernel(x(k,:), x(j,:))  ;
    end
    pred = sign(fx_j + b);
    if(pred ~= y(j))
        errvec(j) = 1;
    end
end
err = sum(errvec);
disp(['error rate = ' num2str(err/numtrain) ]);
```

```
%% plot
xmin = min(x(:,1));
xmax = max(x(:,1));
xstep = (xmax-xmin)/200;
xaxis = xmin:xstep:xmax;

% linear kernel
yaxis = (-w(1)*xaxis/w(2))-b/w(2);
% ypos = (-w(1)*xaxis/w(2))-(b-1)/w(2);
% yneg = (-w(1)*xaxis/w(2))-(b+1)/w(2);


figure, hold on;
list = find(y== 1);
plot(x(list,1), x(list,2), 'r*', 'linewidth', 5 );

list = find(y== -1);
plot(x(list,1), x(list,2), 'b*', 'linewidth', 5);

% list = find(errvec == 1);
% plot(data(list,1), data(list,2), 'mX', 'linewidth', 1, 'MarkerSize', 8);

list = find(alpha > 0);
plot(x(list,1), x(list,2), 'gO', 'MarkerSize', 8, 'linewidth', 2);

axis tight;

h = legend('class 1','class 2','support vector');
set(h,'Interpreter','none')

hold off;

% Plotting the classification plane, Sequential Minimal Optimization for SVM
% Chunsheng Fang, Aravind R.
% Univ. of Cincinnati, 20090505
% www.cs.uc.edu/~fangcg
% www.VictorFang.com

% [plx ply] = meshgrid(-30:50,-30:50 );
xmin = min(x(:,1));
xmax = max(x(:,1));
xstep = (xmax-xmin)/20;
xrange = xmin:xstep:xmax;
ymin = min(x(:,2));
ymax = max(x(:,2));
ystep = (xmax-xmin)/20;
```

```
yrange = ymin:ystep:ymax;

plz = zeros(length(xrange), length(yrange));

for ix = 1:length(xrange)
    for iy = 1:length(yrange)
        fx_j = 0;
        position = [xrange(ix),  yrange(iy)];
        for k = 1:numtrain
            fx_j = fx_j + alpha(k) * y(k) * kernel(x(k,1:2), position )  ;
        end
        pred = sign(fx_j + b);
        plz(ix, iy) = pred;
    end
end

figure, imagesc( xrange, fliplr(yrange), plz);
```