Data Mining 2
TP 2.3 SVDD

Thomas Robert

Introduction

L'objectif de ce TP est de tester diverses méthodes de SVDD : linéaire avec et sans erreur, avec kernel.

Generate dataset

On génère un jeu de données

```
n = 50;
p = 2;
Xi = randn(n,p) + ones(n,1) *[1.5  2.5];
```

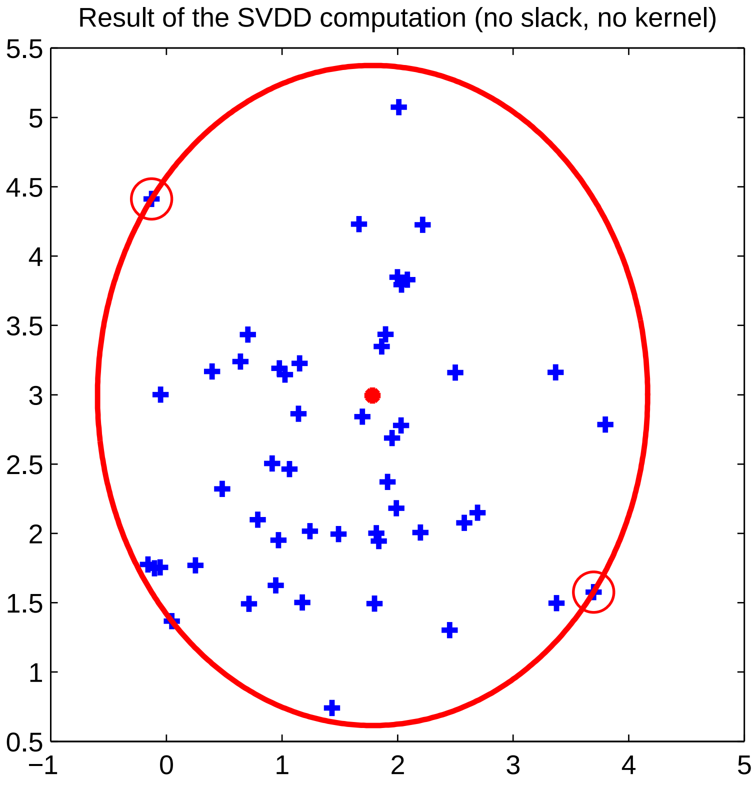
Linear SVDD without slack

On résout le problème de SVDD linéaire sans erreur avec plusieurs méthode, on constate (heureusement...) que les résultats sont tous les même. Comme toujours, cvx est plus lent que monqp.

```
% Solving linear SVDD primal problem with cvx

tic
cvx_begin quiet
    variables R(1) cSVDD(2)
    dual variable d
    minimize( R )
    subject to
    d : sum((Xi - ones(n,1) *cSVDD') .^2 ,2) <= R;
cvx_end
tocPrimal = toc;

% Solving linear SVDD dual problem with cvx

G = Xi*Xi'; % build the Gram matrix
nx = diag(G) ; % compute the norms
e = ones(n,1) ; % vector of n ones

tic
cvx_begin quiet
    variable a(n)
    dual variables eq po
    minimize( a'*G*a - a'*nx )
    subject to
    eq : a'*e == 1;
    po : 0 <= a;
cvx_end

tocDual = toc;

% Solving linear SVDD dual problem as QP with monQP

C = inf; % no slack
l = 10^ -12; % duality gap
verbose = 0; % to see what's going on (set it to 0 to mute it)
tic
[am, lambda , pos] = monqp(2*G,nx,e,1 ,C,l,verbose) ;
```

```
tocMonQP = toc;

cm = am'*Xi(pos ,:) ; % as line vector! To be transposed if necessary
Rm = lambda + cm*cm';

% Comparing results
% 1st : primal with cvx
% 2nd : dual with cvx
% 3rd : dual with monQP

disp('=== Comparing the results ===');

disp('Different R results');
disp([R lambda+cm*cm' Rm ]) % the radius

disp('Different C results');
disp([ cm' Xi'*a cSVDD]) % the centers

disp('Different alpha results');
aM = 0*a; % rebuilding a full vector of dual variables
aM(pos) = am;
disp([d a aM]) % the dual variables

disp('Computation times');
disp([tocPrimal, tocDual, tocMonQP]);

visualize_SVDD(Xi,cSVDD ,R,pos, 'r');
title('Result of the SVDD computation (no slack, no kernel)');
```

```
=== Comparing the results ===
Different R results
    5.6663    5.6663    5.6663

Different C results
    1.7837    1.7836    1.7836
    2.9949    2.9947    2.9947

Different alpha results
    0.0000    0.0000         0
    [...]
    0.0000    0.0000         0
    0.5000    0.5000    0.5000
    0.0000    0.0000         0
    [...]
    0.0000    0.0000         0
    0.4999    0.4999    0.5000
    0.0000    0.0000         0
    [...]

Computation times
    2.7198    1.1348    0.3960
```



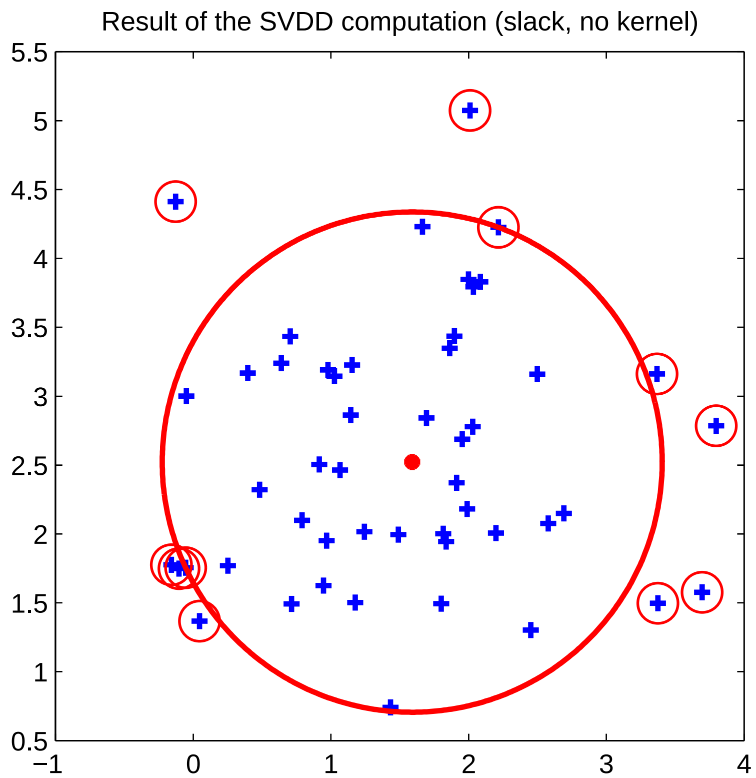
Linear SVDD with slack

On applique le même principe avec un SVDD linéaire avec erreur et on constate qu'effectivement, un certain nombre de points ont été exclus du cercle.

```
% Primal linear SVDD with slack with cvx

tic;
C = .1;
cvx_begin quiet
    variables m(1) cSVDD(2) xi(n)
    dual variables d dp
    minimize( .5*cSVDD'*cSVDD - m + C * sum(xi) )
    subject to
        d : Xi * cSVDD >= m + .5*nx - xi;
        dp: xi >= 0;
cvx_end
tocPrimal = toc;

R = cSVDD'*cSVDD - 2*m;
pos = find(d > eps^.5) ;

% Dual linear SVDD with slack with cvx and monQP

tic
cvx_begin quiet
    variable a(n)
    dual variables eq po pC
    minimize( a'*G*a - a'*nx )
    subject to
        eq : a'*e == 1;
        po : 0 <= a;
        pC : a <= C;
cvx_end
tocDual = toc;

tic;
[am, lambda , pos] = monqp(2*G,nx,e,1 ,C,l,verbose) ;
tocMonQP = toc;

% Comparing results
```

```
disp('=== Comparing the results ===');

disp('Different R results');
disp([[R lambda+am'*G(pos,pos)*am ]]) % the radius

disp('Different C results');
disp([ cSVDD Xi(pos ,:)'*am ]) % the centers

disp('Computation times');
disp([tocPrimal, tocDual, tocMonQP]);

figure;
visualize_SVDD(Xi,cSVDD ,R,pos, 'r');
title('Result of the SVDD computation (slack, no kernel)');
```

```
=== Comparing the results ===
Different R results
    3.2961    3.2961

Different C results
    1.5903    1.5903
    2.5215    2.5215

Computation times
    1.1165    1.1807    0.0349
```



SVDD with gaussian kernel

Pour cette partie de SVDD avec kernel, on programme 2 fonctions SVDDClass et SVDDVal permettant d'apprendre puis d'utiliser n'importe quel type de kernel.

On test ces fonctions avec des noyaux gaussiens et polynomiaux. On voit encore une fois l'effet de la bande passante sur le kernel gaussien, et on voit que le kernel polynomial est bien plus "lisse" que le kernel gaussien.

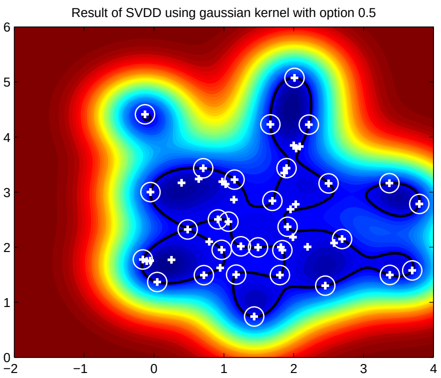
Comme toujours en Data Mining, l'étape suivante devrait être le réglage correct de l'hyperparamètre kerneloption afin d'avoir le meilleur résultat possible.

```
function [Xsup, alpha, b] = SVDDClass(Xi, C, kernel, kerneloption, options)

n = size(Xi, 1);

% compute the kernel
if (strcmp(kernel, 'polynomial'))
    G = (Xi*Xi' + ones(n) ) .^kerneloption;
else
    G = svmkernel(Xi, kernel, kerneloption);
end

% create usefull vectors
nx = diag(G);
e = ones(n,1);

% compute the solution
l = sqrt(eps);
[alpha, b, pos] = monqp(2*G,nx,e,1 ,C,l, 0) ;

% X support
Xsup = Xi(pos,:);
```

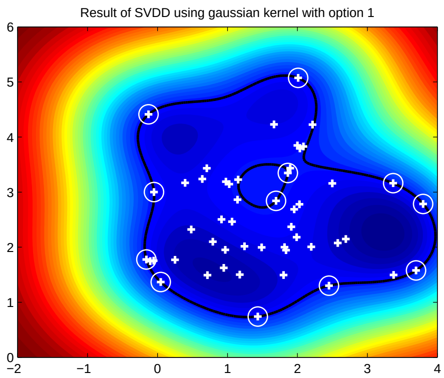
```
function [ypred] = SVDDVal(Xtest, Xsup, alpha, b, kernel, kerneloption)

[n, p] = size(Xtest);

if (strcmp(kernel, 'polynomial'))
    K = (Xtest*Xsup' + ones(n, length(Xsup))) .^kerneloption;
    N_K = (sum(Xtest.^2, 2) + ones(n,1) ) .^kerneloption;
    ypred = N_K - 2*K*alpha - b;
else
    K = svmkernel(Xtest, kernel, kerneloption, Xsup);
    ypred = 1 - 2*K*alpha - b;
end
```

```
C = 10;

kernels = {};

kernels{end + 1} = struct('kernel', 'gaussian', 'kerneloption', 0.5);
kernels{end + 1} = struct('kernel', 'gaussian', 'kerneloption', 1);
kernels{end + 1} = struct('kernel', 'gaussian', 'kerneloption', 2);
kernels{end + 1} = struct('kernel', 'polynomial', 'kerneloption', 1);
kernels{end + 1} = struct('kernel', 'polynomial', 'kerneloption', 2);
kernels{end + 1} = struct('kernel', 'polynomial', 'kerneloption', 3);
kernels{end + 1} = struct('kernel', 'polynomial', 'kerneloption', 6);
kernels{end + 1} = struct('kernel', 'polynomial', 'kerneloption', 9);
kernels{end + 1} = struct('kernel', 'polynomial', 'kerneloption', 15);

for i = 1:length(kernels)
    kernel = kernels{i}.kernel;
    kerneloption = kernels{i}.kerneloption;

    % Learn SVDD
```

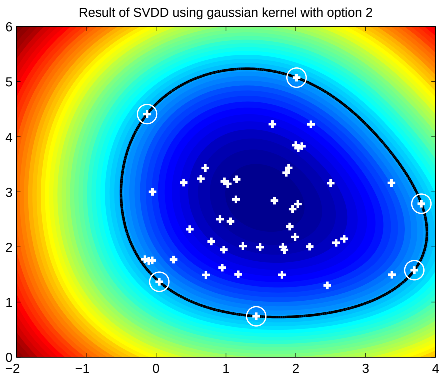
```
    [Xsup, alpha, b] = SVDDClass(Xi, C, kernel, kerneloption);

    % Class test data
    [xtest1 xtest2] = meshgrid([ -1:.01:1]*3+1 ,[ -1:0.01:1]*3+3) ;
    nn = length(xtest1) ;
    Xgrid = [reshape(xtest1 ,nn*nn,1) reshape(xtest2 ,nn*nn,1) ];
    ypred = reshape(SVDDVal(Xgrid, Xsup, alpha, b, kernel, kerneloption),nn,nn);

    % Plot data
    figure;
    contourf(xtest1 ,xtest2 ,ypred ,50) ; shading flat; hold on;
    [cc,hh]=contour(xtest1 ,xtest2 ,ypred ,[ -1 0] , 'k' , 'LineWidth' ,2) ;
    plot(Xi(: ,1) ,Xi(: ,2) , '+w' , 'LineWidth' ,2) ;
    plot(Xsup(:,1) ,Xsup(:,2) , 'ob' , 'LineWidth' ,1 ,...
    'MarkerEdgeColor' , 'w' , 'MarkerSize' ,15) ;
    hold off
    title(['Result of SVDD using ' kernel ' kernel with option ' num2str(kerneloption)])
end
```



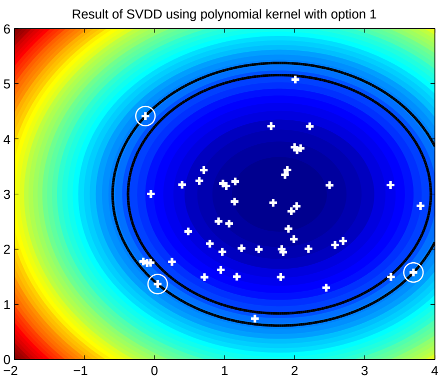
(a) Gaussien ($\sigma = 0.5$)



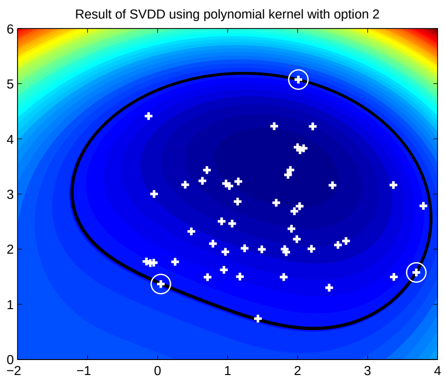
(b) Gaussien ($\sigma = 1$)



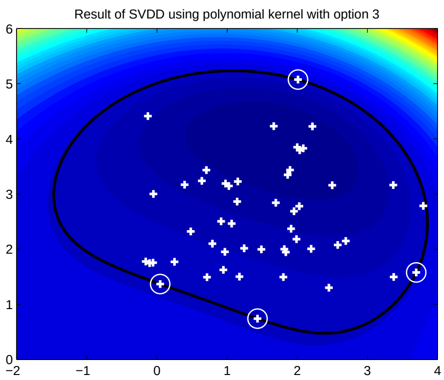
(c) Gaussien ($\sigma = 2$)



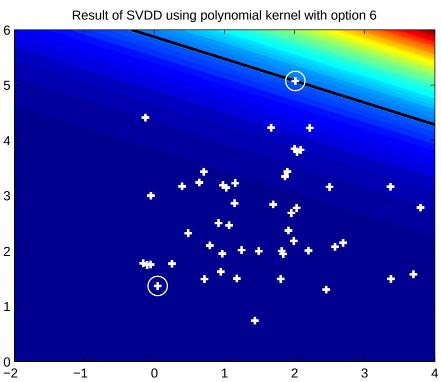
(d) Polynomial degré 1



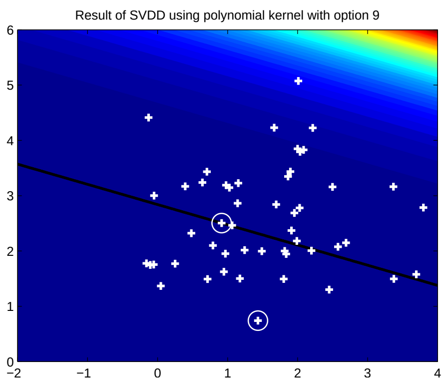
(e) Polynomial degré 2



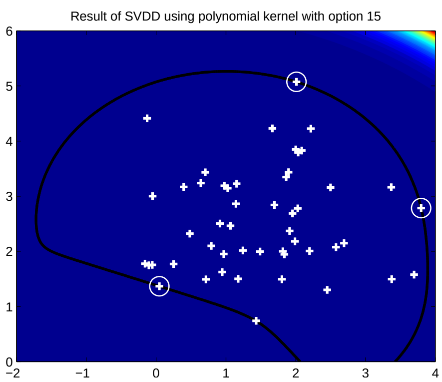
(f) Polynomial degré 3



(g) Polynomial degré 6



(h) Polynomial degré 9



(i) Polynomial degré 15

Résultats du SVDD avec différents kernels