Data Mining 2
TP 2.2 SVM et hyperparamètres

Thomas Robert

Introduction

L'objectif du TP est d'étudier une manière de trouver correctement le meilleur réglage des hypermètres d'un SVM sur les données d'apprentissage.

```
[Xi,yi] = read_libsvm('splice.a') ;
[na, p] = size(Xi);
```

Coarse grid

On génère une d'hyperparamètres très large et peu precise pour trouver une zone dans laquelle chercher plus finement le meilleur réglage.

Pour accélérer les calculs, on utilise que 10% du jeu de données le calcul d'erreur.

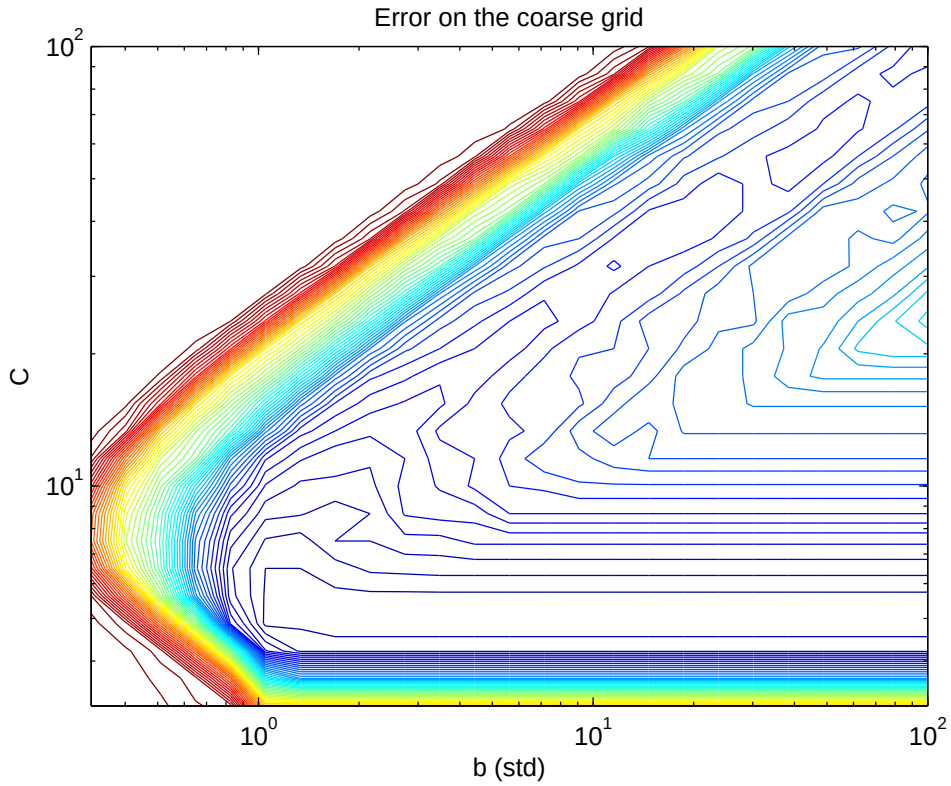
On fait ce calcul 2 fois (avec des tirages du jeu d'apprentissage différents) pour le stabiliser un peu.

```
% Generate grid
b_grid = logspace(0.5 ,2 ,25) ;
C_grid = logspace( -0.5 ,2 ,25) ;

% Split data 10 / 90 %
percent = 0.1;
[Xa,ya,Xval,yval] = split_data(Xi,yi,percent) ;
la = eps^.5;
kernel= 'gaussian';

% Create error grid
Err = zeros(length(b_grid), length(C_grid));
for i = 1:2
    [Xa,ya,Xval,yval] = split_data(Xi,yi,percent) ;
    tic
    [b_opt, C_opt, ErrTemp] = svm_CV(Xa,ya,Xval,yval,b_grid ,C_grid ,kernel ,la) ;
    disp(['Coarse grid iteration ' int2str(i) ' : ' num2str(toc) ' s'])
    Err = (Err + ErrTemp) / i;
end

% Minimum error
[Errmin C_ind] = min(min(Err));
C_opt = C_grid(C_ind);
[Errmin b_ind] = min(min(Err'));
b_opt = b_grid(b_ind);
```

```
Coarse grid iteration 2 : 28.6499 s
```

Plot error

L'affichage de l'erreur nous montre comment l'erreur varie en fonction des réglages des hyperparamètres sur la grille.




```
figure
contour(C_grid, b_grid, Err, [10:.5:50]) ;
hold on
set(gca,'xscale','log')
set(gca,'yscale','log')
title('Error on the coarse grid');
xlabel('b (std)');
ylabel('C');
```

Fine grid

On génère cette fois une grille plus fine autour de l'optimum déterminé sur la grille large.

Cette fois, on applique la méthode de la validation croisée avec 2 blocs pour évaluer l'erreur.

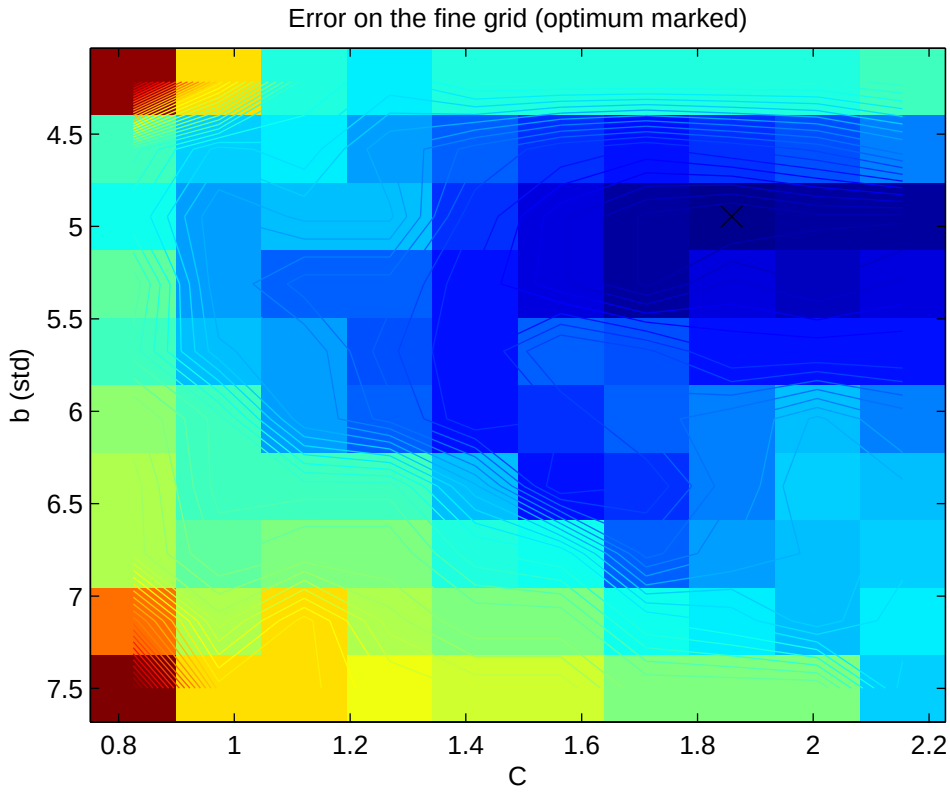
```
% Generate fine grid
b_min = b_grid(b_ind -2) ;
b_max = b_grid(b_ind+2) ;
b_grid = linspace(b_min ,b_max ,10) ;
C_min = C_grid(C_ind -2) ;
C_max = C_grid(C_ind+2) ;
C_grid = linspace(C_min ,C_max ,10) ;

% Split data
percent = 0.5;
[X1,y1,X2,y2] = split_data(Xi,yi,percent) ;

% Create error grid with cross validation
tic
[b_opt ,C_opt ,Err1] = svm_CV(X1,y1,X2,y2, b_grid, C_grid ,kernel ,la) ;
disp(['Fine grid iteration 1 : ' num2str(toc) ' s'])
tic
[b_opt ,C_opt ,Err2] = svm_CV(X2,y2,X1,y1, b_grid, C_grid ,kernel ,la);
disp(['Fine grid iteration 2 : ' num2str(toc) ' s'])
Err = (Err1+ Err2) /2;

% Best parameters
[Errmin C_ind] = min(min(Err));
C_opt = C_grid(C_ind);
[Errmin b_ind] = min(min(Err'));
```

```
b_opt = b_grid(b_ind);
```

```
Fine grid iteration 1 : 92.2067 s
Fine grid iteration 2 : 84.5412 s
```

Plot results

On affiche également les résultats sur la grille fine.




```
figure;
imagesc(C_grid, b_grid, (Err))
hold on
contour(C_grid, b_grid , (Err), [10:.08:50]) ;
plot(C_opt, b_opt, 'xk', 'MarkerSize', 12);

title('Error on the fine grid (optimum marked)');
xlabel('C');
ylabel('b (std)');
```

Estimate error with the best parameters with learning dataset

On estime d'abord l'erreur réelle de notre classifieur à partir du jeu d'apprentissage.

Pour stabiliser cette estimation, on répète 10 fois le calcul.

```
Err = 0;
for i = 1:10
    [Xa,ya,Xval,yval] = split_data(Xi,yi,percent) ;

    [n,p] = size(Xa) ;
    kerneloption = b_opt;
    K=svmkernel(Xa,kernel ,kerneloption) ;
```

```
    G = (ya*ya') .*K;
    e = ones(n,1) ;
    [alpha ,b,pos] = monqp(G,e,ya,0 ,C_opt ,la ,0) ;
    Kt = svmkernel(Xval,kernel ,kerneloption ,Xa(pos ,:) ) ;
    predict_label = sign(Kt*(ya(pos) .*alpha) + b) ;
    [Err_rate, ~] = Error_count(yval, predict_label);
    Err = (Err * (i-1) + Err_rate) / i;
end
disp(['Erreur réelle estimée : ' num2str(Err) '%']);
```

```
Erreur réelle estimée : 14.8%
```

Estimate error with the best parameters with test dataset

On constate une grande différence entre l'erreur

```
[Xt,yt] = read_libsvm('splice.t') ; % test will be only available 15 min.
[nt, p] = size(Xt) ; % before the end of the session

[n,p] = size(Xi) ;
kerneloption = b_opt;
K=svmkernel(Xi,kernel ,kerneloption) ;
G = (yi*yi') .*K;
e = ones(n,1) ;
[alpha ,b,pos] = monqp(G,e,yi,0 ,C_opt ,la ,0) ;
Kt = svmkernel(Xt,kernel ,kerneloption ,Xi(pos ,:) ) ;
predict_label = sign(Kt*(yi(pos) .*alpha) + b) ;
[Err_rate, ~] = Error_count(yt, predict_label);
disp(['Erreur réelle sur jeu de test : ' num2str(Err_rate) '%']);
```

```
Erreur réelle sur jeu de test : 9.7471%
```

L0 kernel

On essaye cette fois un calcul de l'erreur avec un kernel L0.

On se rend compte que les résultats sont bien meilleurs (3,77 %). Cette démonstration prouve qu'avant de faire de longs calculs pour régler les hyperparamètres d'un classifieur, une étape tout aussi importante est le choix du classifieur le plus adapté aux données sur lesquelles ont fait une analyse.

```
for i=1:na % computing le L0 norm by counting the disagreements
    d(: ,i) = sum(Xi' ~=Xi(i ,:)' * ones(1, na))';
end

kerneloption = 16;
monK = exp( -d/kerneloption) ;

G = (yi*yi') .*monK;
e = ones(n,1) ;
C = 10;

[alpha, b, pos] = monqp(G,e,yi,0 ,C,la ,0) ;
dt = zeros(nt,length(pos) ) ;

for i=1:length(pos)
    dt(: ,i) = sum(Xt' ~= Xi(pos(i), :)'*ones(1 ,nt))';
end

Kt = exp( -dt/kerneloption) ;
predict_label = sign(Kt*(yi(pos) .*alpha) + b) ;

[Err_rate, ~] = Error_count(yt, predict_label)
```