DM_2 –Practical session 5: Multiple kernel learning with SVM

Stéphane Canu
scanu@insa-rouen.fr, asi.insa-rouen.fr\~scanu

April 16, 2014

Practical session description

This practical session aims at showing how to chose the kernel using the multiple kernel learning strategy.

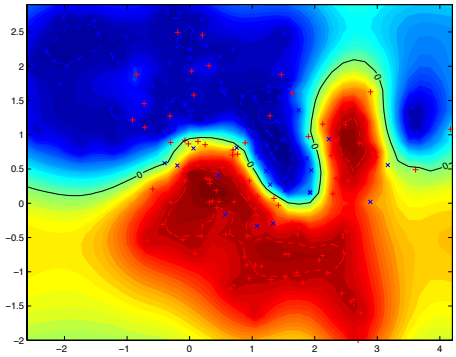

Figure 1: Result of practical session 9: an example of MKL on the mixture dataset.

Ex. 1 — Multiple kernel learning with SVM

1. For this practical session we are going to use the mixture data from the book `http://statweb.stanford.edu/~tibs/ElemStatLearn/`. They are also available on moodle (`TP5 - Les Données`). Load it and visualize it.

```
load 'mixtureexampleTRAIN.dat'
Xapp = mixtureexampleTRAIN(1:400);
Xapp = reshape(Xapp,200,2);
yapp = [ones(100,1);-ones(100,1)];
load 'mixtureexampleTEST1.dat'
Xt = mixtureexampleTEST1(1:2*6831);
Xt = reshape(Xt,6831,2);
load 'mixtureexamplePtest.dat';
yt = -sign(mixtureexamplePtest-.5);
prob = mixtureexamplePtest;
nt = length(yt);
load 'mixtureexampleMarg.dat'
marg = mixtureexampleMarg;
```

2. Classification using SVM
   a) Train a SVM with gaussian kernel with $b = .1$ and $C = 100$

```
kernel='gaussian';
kerneloption1 = .1;
K1=svmkernel(Xapp,kernel,kerneloption1);
G1 = (yapp*yapp').*K1;

C = 100;
lambda = 1e-12;
e = ones(n,1);
[alpha,b,pos] = monqp(G1,e,yapp,0,C,lambda,0);

Kt1 = svmkernel(Xt,kernel,kerneloption1,Xapp(pos,:));
ypred = Kt1*(yapp(pos).*alpha) + b;
nerr = 100*length(find(yt.*ypred<0))/(nt);
```

b) Train another SVM with another gaussian kernel but with $b = .5$

```
kerneloption2 = .5;
K2=svmkernel(Xapp,kernel,kerneloption2);
G2 = (yapp*yapp').*K2;
[alpha,b,pos] = monqp(G2,e,yapp,0,C,lambda,0);
Kt2 = svmkernel(Xt,kernel,kerneloption2,Xapp(pos,:));
ypred = Kt2*(yapp(pos).*alpha) + b;
nerr = 100*length(find(yt.*ypred<0))/(nt);
```

c) Train a third SVM with a third gaussian kernel with $b = 5$

```
kerneloption3 = 5;
K3=svmkernel(Xapp,kernel,kerneloption3);
G3 = (yapp*yapp').*K3;
[alpha,b,pos] = monqp(G3,e,yapp,0,C,lambda,0);
Kt3 = svmkernel(Xt,kernel,kerneloption3,Xapp(pos,:));
ypred = Kt3*(yapp(pos).*alpha) + b;
nerr = 100*length(find(yt.*ypred<0))/(nt);
```

3. A mixing kernel
   a) Build a kernel as the weighted mean of the 3 previously computed kernels with weights $\mu_1 = 10$ and $\mu_2 = \mu_3 = 1$.

```
mu = [10 ; 1 ; 1];
mu = mu/sum(mu);
G = (yapp*yapp').*(mu(1)*K1+mu(2)*K2+mu(3)*K3);
```

   b) Train a new SVM on this kernel kernel using `monqp`

```
[alpha,b,pos] = monqp(G,e,yapp,0,C,lambda,0);
```

   c) Evaluate the SVM on the test set

```
Kt1 = svmkernel(Xt,kernel,kerneloption1,Xapp(pos,:));
Kt2 = svmkernel(Xt,kernel,kerneloption2,Xapp(pos,:));
Kt3 = svmkernel(Xt,kernel,kerneloption3,Xapp(pos,:));
Kt = mu(1)*Kt1+mu(2)*Kt2+mu(3)*Kt3;
ypred = Kt*(yapp(pos).*alpha) + b;
```

   d) Calculates the error rate

```
nerr = 100*length(find(yt.*ypred<0))/(nt);
```

4. Gradient iteration
   a) Compute the projected gradient of the cost function with respect to the weight vector μ

```
g =[alpha'*G1(pos,pos)*alpha;alpha'*G2(pos,pos)*alpha;alpha'*G3(pos,
    pos)*alpha];
g = g/norm(g);
d = g-g(1);
d(1) = -sum(d);
```

   b) Perform say 50 iterations of the gradient method with fixed step size $\rho = 0.002$.

```
for i=1:50
    g = [alpha'*G1(pos,pos)*alpha ; alpha'*G2(pos,pos)*alpha ; alpha'*
        G3(pos,pos)*alpha ];
    g = g/norm(g);
    d = g-g(1);
    d(1) = -sum(d);
    step = 0.002;      % fix step gradient is bad!
    mu = max(0,mu - step*d);
    mu = mu/sum(mu);

    G = (yapp*yapp').*(mu(1)*K1+mu(2)*K2+mu(3)*K3);
    [alpha,b,pos] = monqp(G,e,yapp,0,C,lambda,0);
end
```

c) Calculates the error rate after the say 50 iterations.

```
Kt1 = svmkernel(Xt,kernel,kerneloption1,Xapp(pos,:));
Kt2 = svmkernel(Xt,kernel,kerneloption2,Xapp(pos,:));
Kt3 = svmkernel(Xt,kernel,kerneloption3,Xapp(pos,:));
Kt = mu(1)*Kt1+mu(2)*Kt2+mu(3)*Kt3;

ypred = Kt*(yapp(pos).*alpha) + b;

nerr = 100*length(find(yt.*ypred<0))/(nt);
```

5. Use the same kernels with SimpleSKM. First download the SimpleMKL toolbox from `http://asi.insa-rouen.fr/enseignants/~arakoto/code/mklindex.html`.
   a) Set the defat parameters for the MKL by cut and past from the example file `exmklclass`.

```
verbose=1;
options.algo='svmclass'; % Choice of algorithm in mklsvm can be either
                         % 'svmclass' or 'svmreg'
%-------------------------------------------------------
% choosing the stopping criterion
%-------------------------------------------------------
options.stopvariation=0;  % use variation of weights for stopping
options.stopKKT=0;        % set to 1 if you use KKTcondition for
options.stopdualitygap=1; % set to 1 for using duality gap for
%-------------------------------------------------------
% choosing the stopping criterion value
%-------------------------------------------------------
options.seuildiffsigma=1e-5;       % stopping criterion for weight
options.seuildiffconstraint=0.001; % stopping criterion for KKT
options.seuildualitygap=0.001;  % stopping criterion for duality gap
%-------------------------------------------------------
% Setting some numerical parameters
%-------------------------------------------------------
options.goldensearch_deltmax=1e-3; % initial precision of golden
options.numericalprecision=1e-8;   % numerical precision weights below
options.lambdareg = 1e-8;          % ridge added to kernel matrix

%-------------------------------------------------------
% some algorithms paramaters
%-------------------------------------------------------
options.firstbasevariable='first'; % tie breaking method for the base
                                   % variable in the reduced gradient
options.nbitermax=500;             % maximal number of iteration
options.seuil=0;                   % forcing to zero weights lower
options.seuilitermax=10;           % value, for iterations lower than
options.miniter=0;                 % minimal number of iterations
options.verbosesvm=0;              % verbosity of inner svm algorithm
options.efficientkernel=0;         % use efficient storage of kernels
```

   b) Build the Kernel K for the MKL learning

```
kernelt={'gaussian' 'gaussian'};
kerneloptionvect={ [kerneloption1 kerneloption2 kerneloption3] [
    kerneloption1 kerneloption2 kerneloption3]};
variablevec={'all' 'single'};

classcode=[1 -1];
[nbdata,dim]=size(Xapp);

[kernel,kerneloptionvec,variableveccell]=CreateKernelListWithVariable(
    variablevec,dim,kernelt,kerneloptionvect);
[Weight,InfoKernel]=UnitTraceNormalization(Xapp,kernel,kerneloptionvec
    ,variableveccell);
K=mklkernel(Xapp,InfoKernel,Weight,options);
```

c) Train the MKL

```
[beta,w,b,posw,story(i),obj(i)] = mklsvm(K,yapp,C,options,verbose);
```

d) Calculate the kernel on the test set, evaluate the MKL on the test data and calculate the error rate.

```
Kt=mklkernel(Xt,InfoKernel,Weight,options,Xapp(posw,:),beta);
ypred=Kt*w+b;
nerr = 100*length(find(yt.*ypred<0))/(nt);
```

6. Do the nice plot

```
[xtest1 xtest2]  = meshgrid([-1:.05:1.2]*3.5,[-1:0.05:1]*3);
[nn mm] = size(xtest1);
Xtest = [reshape(xtest1 ,nn*mm,1) reshape(xtest2 ,nn*mm,1)];

Kt=mklkernel(Xtest,InfoKernel,Weight,options,Xapp(posw,:),beta);
ypred=Kt*w+b;
ypred = reshape(ypred,nn,mm);

figure(1)
contourf(xtest1,xtest2,ypred,50);shading flat;hold on;
plot(Xapp(yapp==1,1),Xapp(yapp==1,2),'+r');
plot(Xapp(yapp==-1,1),Xapp(yapp==-1,2),'xb');
[cc, hh]=contour(xtest1,xtest2,ypred,[-100000 1],'--r');
[cc, hh]=contour(xtest1,xtest2,ypred,[-100000 -1 ],'--b');
[cc, hh]=contour(xtest1,xtest2,ypred,[-100000 0],'k');
axis([min(Xt(:,1)) max(Xt(:,1)) min(Xt(:,2)) max(Xt(:,2))]);
clabel(cc,hh);
```

7. What is in you opinion a good way to choose a kernel? What about building one? How?