LAMPIRAN A
LISTING PROGRAM

LINIER SVC

```
%%%%%%%%%%%%%%%%%%%%%%
%%% Data Generation %%%
%%%%%%%%%%%%%%%%%%%%%%

x=[randn(1,10)-1 randn(1,10)+1;randn(1,10)-1 randn(1,10)+1]';
y=[-ones(1,10) ones(1,10)]';

%%%%%%%%%%%%%%%%%%%%%%
%%% SVC Optimization %%%
%%%%%%%%%%%%%%%%%%%%%%

R=x*x'; % Dot products
Y=diag(y);
H=Y*R*Y+1e-6*eye(length(y)); % Matrix H regularized
f=-ones(size(y)); a=y'; K=0;
Kl=zeros(size(y));
C=100; % Functional Trade-off
Ku=C*ones(size(y));
alpha=quadprog(H,f,[],[],a,K,Kl,Ku); % Solver
w=x'*(alpha.*y); % Parameters of the Hyperplane

%%% Computation of the bias b %%%

e=1e-6; % Tolerance to errors in alpha
ind=find(alpha>e & alpha<C-e) % Search for 0 < alpha_i < C
b=mean(y(ind) - x(ind,:)*w) % Averaged result

%%%%%%%%%%%%%%%%%%%%%%
%%% Representation %%%
%%%%%%%%%%%%%%%%%%%%%%

data1=x(find(y==1),:);
data2=x(find(y==-1),:);
svc=x(find(alpha>e),:);
plot(data1(:,1),data1(:,2),'o')
hold on
plot(data2(:,1),data2(:,2),'*')
plot(svc(:,1),svc(:,2),'s')
% Separating hyperplane
plot([-3 3],[(3*w(1)-b)/w(2) (-3*w(1)-b)/w(2)])
% Margin hyperplanes
plot([-3 3],[(3*w(1)-b)/w(2)+1 (-3*w(1)-b)/w(2)+1],'--')
plot([-3 3],[(3*w(1)-b)/w(2)-1 (-3*w(1)-b)/w(2)-1],'--')
y_pred=sign(x*w+b); %Test
error=mean(y_pred~=y); %Error Computation
```

Linier SVR

```
function [b,w,error]=reg(x,y)
R_=x*x';
R=[R_ -R_;-R_ R_];
a=[ones(size(y')) -ones(size(y'))];
y2=[y;-y];
H=(R+1e-9*eye(size(R,1)));
epsilon=0.1; C=100;
f=-y2'+epsilon*ones(size(y2'));
K=0;
K1=zeros(size(y2'));
Ku=C*ones(size(y2'));
alpha=quadprog(H,f,[],[],a,K,K1,Ku); % Solver
beta=(alpha(1:end/2)-alpha(end/2+1:end));
w=beta'*x;
%% Computation of bias b %%
e=1e-6; % Tolerance to errors in
alpha
ind=find(abs(beta)>e & abs(beta)<C-e) % Search for
% 0 < alpha_i < C
b=mean(y(ind) - x(ind,:)*w) % Averaged result
plot(x,y,'.') % All data
hold on
ind=find(abs(beta)>e);
plot(x(ind),y(ind),'s') % Support Vectors
plot([0 1],[b w+b]) % Regression line
plot([0 1],[b+epsilon w+b+epsilon],'--') % Margins
plot([0 1],[b-epsilon w+b-epsilon],'--')
plot([0 1],[1 1.5+1],':') % True model}
```

Non-Linier SVC

```
function [errors,b]=nonlin(k,ktest,C);
ro=2*pi*rand(k,1);
r=5+randn(k,1);
x1=[r.*cos(ro) r.*sin(ro)];
x2=[randn(k,1) randn(k,1)];
x=[x1;x2];
y=[-ones(1,k) ones(1,k)]';
ro=2*pi*rand(ktest,1);
r=5+randn(ktest,1);
x1=[r.*cos(ro) r.*sin(ro)];
x2=[randn(ktest,1) randn(ktest,1)];
xtest=[x1;x2];
ytest=[-ones(1,ktest) ones(1,ktest)]';
N=2*k; % Number of data
sigma=1; % Parameter of the kernel
D=buffer(sum([kron(x,ones(N,1))...
- kron(ones(1,N),x')'].^2,2),N,0);
% This is a recipe for fast computation
% of a matrix of distances in MATLAB
R=exp(-D/(2*sigma)); % Kernel Matrix
Y=diag(y);
H=Y*R*Y+1e-6*eye(length(y)); % Matrix H regularized
f=-ones(size(y)); a=y'; K=0;
Kl=zeros(size(y));
C=100 % Functional Trade-off
Ku=C*ones(size(y));
alpha=quadprog(H,f,[],[],a,K,Kl,Ku); % Solver
e=1e-6;,
ind=find(alpha>e);
x_sv=x(ind,:); % Extraction of the support
% vectors
N_SV=length(ind); % Number of SV
%%% Computation of the bias b %%%
e=1e-6; % Tolerance to
% errors in alpha
ind=find(alpha>e & alpha<C-e); % Search for
% 0 < alpha_i < C
N_margin=length(ind);
D=buffer(sum([kron(x_sv,ones(N_margin,1)) ...
- kron(ones(1,N_SV),x(ind,:)')'].^2,2),N_margin,0);
% Computation of the kernel matrix
R_margin=exp(-D/(2*sigma));
y_margin=R_margin*(y(ind).*alpha(ind));
b=mean(y(ind) - y_margin); % Averaged result
N_test=2*ktest; % Number of test data
%%Computation of the kernel matrix%%
D=buffer(sum([kron(x_sv,ones(N_test,1))...
- kron(ones(1,N_SV),xtest')'].^2,2),N_test,0);
% Computation of the kernel matrix
R_test=exp(-D/(2*sigma));
% Output of the classifier
y_output=sign(R_test*(y(ind).*alpha(ind))+b);
errors=sum(ytest~=y_output) % Error Computation
```

```
data1=x(find(y==1),:);
data2=x(find(y==-1),:);
svc=x(find(alpha>e),:);
plot(data1(:,1),data1(:,2),'o')
hold on
plot(data2(:,1),data2(:,2),'*')
plot(svc(:,1),svc(:,2),'s')
g=(-8:0.1:8)'; % Grid between -8 and 8
x_grid=[kron(g,ones(length(g),1)) kron(ones(length(g),1),g)];
N_grid=length(x_grid);
D=buffer(sum([kron(x_sv,ones(N_grid,1))...
- kron(ones(1,N_SV),x_grid')'].^2,2),N_grid,0);
% Computation of the kernel matrix
R_grid=exp(-D/(2*sigma));
y_grid=(R_grid*(y(ind).*alpha(ind))+b);
contour(g,g,buffer(y_grid,length(g),0),[0 0]) % Boundary draw
```

SVM MVDM Train

```
function [beta,b0,R,r,a_d]=SVM_MVDM_TRAIN(x,s_d,ker,par,gamma,C,epsilon)
s_d=s_d/norm(s_d(:,1));
x=x/norm(x(:,1));
N=size(x,2);
r=[1;-1]; %Possible Symbols
%These are the steering vectors of the desired directions of
arrival
a_d=s_d(:,1);
a_d=kron(a_d,r');
%Compute the matrices K and K_d
K=kernel_matrix(ker,x,x,par);
A=ones(size(K))/N;
K_=K-A*K-K*A+A*K*A;
K_d=kernel_matrix(ker,x,a_d,par);
a=ones(size(x,2),size(K_d,2))/N;
K_d_=(K_d-K*a-A*K_d+A*K*a);
%Compute the matrix of the optimization
R_=K_d_'*pinv(K_)*K_d_*N;
l=[ones(size(r'))];
H=(R+gamma*eye(size(R,l)));
f=ones(size(r'));
OPTIONS = optimset('LargeScale','off',...
'diffmaxchange',1e-4,'Diagnostics','off');
alpha=quadprog(H,-Y'+epsilon*f,[],[],f1,0,...
zeros(size(Y')),C*f,[],OPTIONS);
beta=conj((alpha(1:end/2)-alpha(end/2+1:end)));
b0 = 0;
svi = find( abs(beta) > epsilon);
svii = find(abs(beta) > epsilon & abs(beta) < (C - epsilon));
if length(svii) > 0
b0 = (1/length(svii))*sum(Y(svii) - ...
H(svii,svi)*beta(svi).*ones(size(Y(svii))));
end
```