BLG527E Machine Learning HW3

Aziz KOÇANAOĞULLARI 504141303

November 2, 2014

Question-1

In this question the aim is to write and process PCA over the 'opdigits' dataset and determine which application is better. As it is unnecessary to explain code projections are enough to prove that code works, however one can check and run '.m' code if required in the appendix.

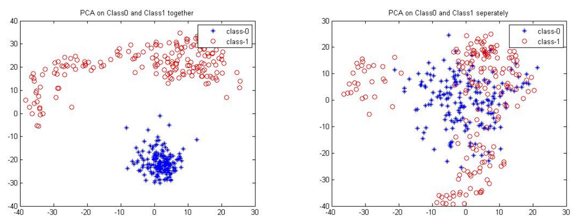

Figure 1: PCA together-PCA 1 by 1

Criticising can be done by computing KNN5 or by our visual system. Since PCA aims to analyse correlation between feature vectors and project all into n (2 in this question) features, the more discrete two classes are the better PCA algorithm is. Just by looking at the projections one can say using PCA on all dataset is better. To give numerical results, one should apply k-nearest neighbourhood algorithm and estimate the classes. The error values are 0 and 5 respectively. This also proves that PCA should be applied on all dataset. In addition mathematically the projection features does not have to be same since they are calculated separately, plotting them in one graph may be wrong as well.

Question-2

In this question one has to decrease the number of features by removing the unrelated ones. The algorithm mRMR-MID selects the most dominant 'n' features of the dataset. Application of PCA over these features should be more beneficial by decreasing computation time or increasing efficiency.
Plots of PCAs after mRMR-MID algorithm are;

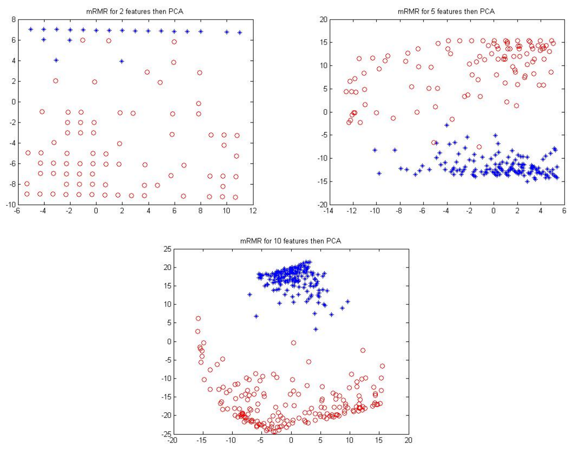

Figure 2: mRMR with 2-5-10

The algorithm is modified and PCA is applied in correct manner since the plots are identical with ones in the homework. The difference can be easily noticed, where 2 most significant features does not mean anything, 5 is approximately divided the data. However 10 is the best at separation. To give numerical results, even considering minor error rate of built in MATLAB predictors, 2 and 5 results in high errors whereas the 10 mRMR gives 0 error.

Question-3

In this question one is required to write K-means algorithm and compute. The algorithm applied on the projected sample space obtained in Question 2 with 10 features. To visualize;

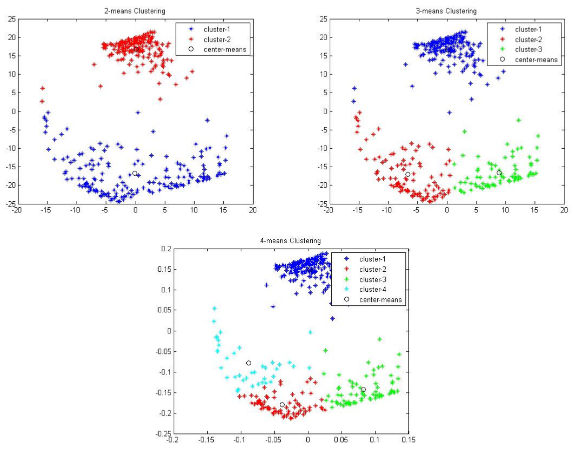

Figure 3: 2-3-4 means

The reconstruction error term;

$$E(\{m_i\}_{i=1}^{k}|X) = \sum_i \sum_t b_i^t \|x^t - m_i\| \tag{1}$$

by it's nature decreases as number of center means increases. However this leads to some kind of over clustering. But errors can show if the number of clusters are sufficient or not. Whereas the error for 2-means is 22663 where the errors for 3 and 4 are 12200 and 11096 respectively. Considering 1/2 error ratio between 2 and 3 clusters, one addition cluster does not change much. So one can say that 3 means are sufficient to model this data.

Question-4

In this question, the assignment is to implement Expectation-Maximization Algorithm for clustering with Gaussians. The code calculated the multivariate Gaussian probabilities, since we have 2 features, for expectation step and maximizes the likelihood function w.r.t. parameters, mean, covariance and probability of the cluster. Where;

$$p(x) = \frac{1}{(2\pi)^{\frac{k}{2}} \|\Sigma\|} exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)) \tag{2}$$

Calculating inverse of the covariance is a problematic task. Since covariance matrix should be symmetric, it is non-singular. However during iterations the sigma ends up being singular. This problem is solved by regularizing the covariance matrix. Also the algorithm processes for a finite number of iterations. Unlike k-means, there is no criterion checked to end process because of the problematic cases. However 100 iterations are surely enough for convergence. The function also requires initial start conditions for means, the means generated from k-means can be used for this process.

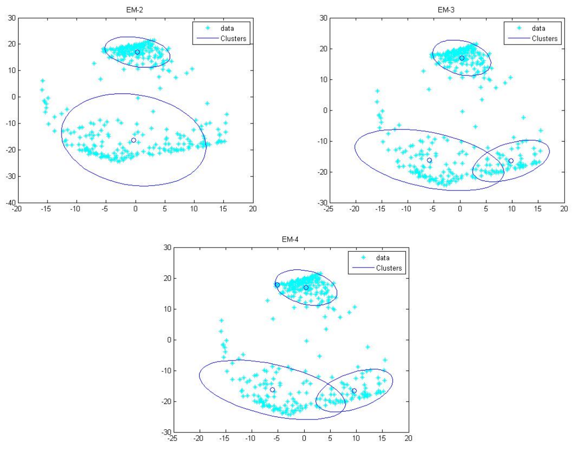

Figure 4: 2-3-4 clusters (100-iterations)

Since means are close to the K-means algorithm's results the reconstruction errors will be close.However it can be clearly seen that EM can not calculate 4th cluster, in other words its variance and probability are relatively low. As can be seen from the figure, there are 3 valid Gaussian generators, the 4th is unnecessary.

How to run code

To run code, one can simply use Matlab interface and call Hw3.m simply by typing Hw3 in the Matlab command prompt. However, even the code will run perfectly until EM-4 calculation, EM-4 calculation should be done repeatedly. This is because in EM-4 the fourth covariance matrix end up being singular. If one can not bypass the error, removing cov-fixer function and recalculating should be enough. The report contains the calculation of EM-4 where the 4th generator is inside the first with nearly 0 variance.
Covariance matrix fixer and Gaussian plot functions are made-functions.

Appendix

Hw-3 Demo

```
%Machine Learning HW3 Demo
%Aziz Kocanaogullari - 504141303
close all;
clear all;

X=dlmread('optdigits01.txt');

[m,n]=size(X);

sumX=sum(X,1);
%Indices of improper features
impind=find(sumX<=0);

%Reconstruct the data matrix with proper values
DatRe=X;
DatRe(:,impind)=[];
XRe=DatRe(:,1:end-1);

%%
%Question-1
%PCA on entire dataset
%Calculate PCA except the last feature ('class label')
[Map1,w{1}]=PCAAziz(XRe,2);

figure();
plot(Map1(1:178,2),Map1(1:178,1),'*b');
hold();
plot(Map1(179:end,2),Map1(179:end,1),'or');
title('PCA on Class0 and Class1 together');
legend('class-0','class-1');

%Knn5-model
mdl = fitcknn(Map1,X(:,end),'NumNeighbors',5);

label=predict(mdl,Map1);
disp(strcat('Error for PCA on all:',num2str(abs(sum(label-X(:,65))))));

%PCA on class1 and class0 divided
%Calculate two PCAs seperetely. This is not feasible where the two
%transformations are different. So their mappings are different.
[PCAX1,w{1}]=PCAAziz(XRe(1:178,:),2);
[PCAX2,w{2}]=PCAAziz(XRe(179:end,:),2);
```

```

figure();
plot(PCAX1(:,2),PCAX1(:,1),'*b');
hold();
plot(PCAX2(:,2),PCAX2(:,1),'or');
title('PCA on Class0 and Class1 seperately');
legend('class—0','class—1');

mdl = fitcknn([PCAX1;PCAX2],X(:,end),'NumNeighbors',5);
label=predict(mdl,[PCAX1;PCAX2]);
disp(strcat('Error for PCA seperately:',num2str(abs(sum(label—X(:,65))))));

%%
%Question—2
%Usage of mRMR—MID to choose most efficient features
%first input := candidete feature matrix
%second input := class labels for given features (should be a vector)
%third input := Number of features to be picked off
%output := most feasible k features to be used

rec{1}=mrmr_mid_d(XRe,X(:,65),2),
ind=rec{1};
[Map2el,w{3}]=PCAAziz(XRe(:,ind),2);

figure();
plot(Map2el(1:178,2),Map2el(1:178,1),'*b');
hold();
plot(Map2el(179:end,2),Map2el(179:end,1),'or');
title('mRMR for 2 features then PCA');
mdl = fitcknn(Map2el,X(:,end),'NumNeighbors',5);
label=predict(mdl,Map2el);
disp(strcat('Error for mRMR—2:',num2str(abs(sum(label—X(:,65))))));


rec{2}=mrmr_mid_d(XRe,X(:,65),5),
ind=rec{2};
[Map5el,w{4}]=PCAAziz(XRe(:,ind),2);

figure();
plot(Map5el(1:178,2),Map5el(1:178,1),'*b');
hold();
plot(Map5el(179:end,2),Map5el(179:end,1),'or');
title('mRMR for 5 features then PCA');
mdl = fitcknn(Map5el,X(:,end),'NumNeighbors',5);
label=predict(mdl,Map5el);
disp(strcat('Error for mRMR—5:',num2str(abs(sum(label—X(:,65))))));

rec{3}=mrmr_mid_d(XRe,X(:,65),10),
ind=rec{3};
[Map5el,w{5}]=PCAAziz(XRe(:,ind),2);

figure();
plot(Map5el(1:178,2),Map5el(1:178,1),'*b');
hold();
plot(Map5el(179:end,2),Map5el(179:end,1),'or');
title('mRMR for 10 features then PCA');
mdl = fitcknn(Map5el,X(:,end),'NumNeighbors',5);
label=predict(mdl,Map5el);
disp(strcat('Error for mRMR—10:',num2str(abs(sum(label—X(:,65))))));


%%
```

```
%Question—3

%K—means clustering with last data obtained in Q2
[b,mean,error]=KmeansAziz(Map5el,2);
figure();
plot(Map5el(find(b(:,1)>0),2),Map5el(find(b(:,1)>0),1),'*b');
hold();
plot(Map5el(find(b(:,2)>0),2),Map5el(find(b(:,2)>0),1),'*r');
plot(mean(:,2),mean(:,1),'ok');
title('2—means Clustering');
legend('cluster—1','cluster—2','center—means');
disp(strcat('error for 2=',num2str(error)));

[b,mean,error]=KmeansAziz(Map5el,3);
figure();
plot(Map5el(find(b(:,1)>0),2),Map5el(find(b(:,1)>0),1),'*b');
hold();
plot(Map5el(find(b(:,2)>0),2),Map5el(find(b(:,2)>0),1),'*r');
plot(Map5el(find(b(:,3)>0),2),Map5el(find(b(:,3)>0),1),'*g');
plot(mean(:,2),mean(:,1),'ok');
title('3—means Clustering');
legend('cluster—1','cluster—2','cluster—3','center—means');
disp(strcat('error for 3=',num2str(error)));

[b,mean,error]=KmeansAziz(Map5el,4);
figure();
plot(Map5el(find(b(:,1)>0),2),Map5el(find(b(:,1)>0),1),'*b');
hold();
plot(Map5el(find(b(:,2)>0),2),Map5el(find(b(:,2)>0),1),'*r');
plot(Map5el(find(b(:,3)>0),2),Map5el(find(b(:,3)>0),1),'*g');
plot(Map5el(find(b(:,4)>0),2),Map5el(find(b(:,4)>0),1),'*c');
plot(mean(:,2),mean(:,1),'ok');
title('4—means Clustering');
legend('cluster—1','cluster—2','cluster—3','cluster—4','center—means');
disp(strcat('error for 4=',num2str(error)));

[mean,var]=EM(Map5el,2,100,[—0.01858 0.004171; —0.1686 0.1589]);
figure();
plot(Map5el(:,2),Map5el(:,1),'*c');
plot_gaussian_ellipsoid(fliplr(mean{1}.'),var{1},2);
meandummy=mean{1};
plot(meandummy(2),meandummy(1),'ob');
plot_gaussian_ellipsoid(fliplr(mean{2}.'),var{2},2);
meandummy=mean{2};
plot(meandummy(2),meandummy(1),'ob');
title('EM—2');
legend('data','Clusters');

[mean,var]=EM(Map5el,3,100,[0.1045 —0.083 —0.1686 ; 0.15 0.1589 —1.10]);
figure();
plot(Map5el(:,2),Map5el(:,1),'*c');
plot_gaussian_ellipsoid(fliplr(mean{1}.'),var{1},2);
plot_gaussian_ellipsoid(fliplr(mean{2}.'),var{2},2);
plot_gaussian_ellipsoid(fliplr(mean{3}.'),var{3},2);
meandummy=mean{1};
plot(meandummy(2),meandummy(1),'ob');
meandummy=mean{2};
plot(meandummy(2),meandummy(1),'ob');
meandummy=mean{3};
plot(meandummy(2),meandummy(1),'ob');
title('EM—3');
legend('data','Clusters');
```

```

[mean,var]=EM(Map5el,4,100,[0.1045 -0.083 -0.1686 2 ; 0.15 0.1589 -1.10 -1]);
figure();
plot(Map5el(:,2),Map5el(:,1),'*c');
plot_gaussian_ellipsoid(fliplr(mean{1}.'),var{1},2);
plot_gaussian_ellipsoid(fliplr(mean{2}.'),var{2},2);
plot_gaussian_ellipsoid(fliplr(mean{3}.'),var{3},2);
plot_gaussian_ellipsoid(fliplr(mean{4}.'),var{4},2);
meandummy=mean{1};
plot(meandummy(2),meandummy(1),'ob');
meandummy=mean{2};
plot(meandummy(2),meandummy(1),'ob');
meandummy=mean{3};
plot(meandummy(2),meandummy(1),'ob');
meandummy=mean{4};
plot(meandummy(2),meandummy(1),'ob');

title('EM-4');
legend('data','Clusters');
```

Expectation-Maximization Algorithm

```
function [ mean,eps,pi ] = EM( X ,dim , max_iter , initmean )

mulvargauss= @(x,mean,eps) (1/sqrt((2*3.14*norm(eps))))..
..*exp(-((x.'-mean).'*inv(eps)*(x.'-mean)));

%Input variables
%X         := Nx2 data matrix
%dim       := number of clusters
%max_iter  := maximum iterations
%initmean  := 2xdim matrix contains [x;y] values for initial mean

%Output variables
%pi    := probs of each cluster
%mean  := means of clusters
%eps   := covariance matrices

%Initialization
%Mean initialization
for count=1:dim,

    mean{count}=initmean(:,count);

end
%Random initialization
pi=abs(randn(1,dim));
pi=pi/(sum(pi(:)));

for k=1:dim,

    eps{k}=[1 0; 0 1];

end

%Iteration
wb = waitbar(0,'Processing EM-Algorithm');

for iter=1:max_iter,
    waitbar(iter / max_iter)

    %Expectation-Step
    for i=1:size(X,1),
        dum=0;
        for k=1:dim,

            dum=dum+pi(k)*mulvargauss(X(i,:),mean{k},eps{k});

        end

        for k=1:dim,

            %gamma_ki value (random gaussian)
            gamki(i,k)=pi(k)*mulvargauss(X(i,:),mean{k},eps{k})/dum;

        end
    end
    clear dum;

    %Maximization-Step
    for k=1:dim,

        %update mean
        dum1=sum(gamki(:,k).*X(:,1))/sum(gamki(:,k));
        dum2=sum(gamki(:,k).*X(:,2))/sum(gamki(:,k));
        meannew{k}=[dum1 ; dum2];
        %clear dummy variable for other purposes
        clear dum1 dum2;

        %update probabilities of clusters
        pi(k)=sum(gamki(:,k))/size(X,1);

        %update covariances
        dummean=mean{k};
        summ=zeros(2,2);

        %summ=0;
        for i=1:size(X,1),

            summ=summ+gamki(i,k)*([X(i,1)-dummean(1);X(i,2)-dummean(2)]*..
            ..[X(i,1)-dummean(1);X(i,2)-dummean(2)].');

        end

        clear dum1 dum2 dum3 dummean;
        dumeps=summ/sum(gamki(:,k));
        eps{k}=gmmb_covfixer(dumeps);
        mean{k}=meannew{k};

        clear summ;

    end
end
close(wb);

end
```

gmmb-covfixer

```
%GMMB_COVFIXER   - force matrix to be a valid covariance matrix
%
% covmatrix = GMMB_COVFIXER(matrix)
% Matrix is forced (complex conjugate) symmetric,
% positive definite and its diagonal real valued.
%
% [covmatrix, loops] = GMMB_COVFIXER(...)
%    loops - number of rounds the positive definite fixer had to run.
%
% [covmatrix, loops, symerr] = GMMB_COVFIXER(...)
%    symerr - symmetry error matrix

%
%
% gmmb_covfixer.m,v 1.2 2004/11/02 09:00:18 paalanen Exp
% Copyright 2003, Pekka Paalanen <pekka.paalanen@lut.fi>

% except isspd() function which is from The MathWorks Matlab mvnpdf.m.

function [nsigma, varargout] = gmmb_covfixer(sigma);

D = size(sigma, 1);
fixrate = 0.01;
covfixmat = ones(D) + fixrate*eye(D);
loops = 0;
min_limit = eps*10;

if ~all( isfinite(sigma(:))  )
    error('covariance matrix is not finite');
end

% Running imagfixer is not counted as covariance fixing,
% the changes are assumed to be so small.
nsigma = imagfixer(sigma);

if nargout>2
    varargout(2) = {(sigma-nsigma)};
end

while isspd(nsigma) == 0
    % covariance matrix is not positive definite
    % fix it
    loops  = loops+1;
    d = diag(nsigma);
    if any(d <= min_limit)
        % negative or zero (<eps) on the diagonal
        m = max(abs(d)) * fixrate;
        neg = min(d);
        if neg < 0
            % there is a negative component on the diagonal
            % get rid of it.
            addit = (m-neg)*eye(D);
        else
            if m < min_limit
                m = min_limit;
            end
            addit = m*eye(D);
        end
        nsigma = nsigma + addit;
    else
        % increase diagonal values by 1 percent
        nsigma = nsigma .* covfixmat;
    end
end

```

```
if nargout>1
    varargout(1) = {loops};
end


% --------------------

function [t,R] = isspd(Sigma)
%ISPDS Test if a matrix is positive definite symmetric
% T = ISPDS(SIGMA) returns a logical indicating whether the matrix SIGMA is
% square, symmetric, and positive definite, i.e., it is a valid full rank
% covariance matrix.
%
% [T,R] = ISPDS(SIGMA) returns the cholesky factor of SIGMA in R.  If SIGMA
% is not square symmetric, ISPDS returns [] in R.

%   Copyright 1993-2002 The MathWorks, Inc.
%   Revision: 1.2   Date: 2002/03/28 16:51:27

% Test for square, symmetric
% NOTE: imagfixer already enforces squareness and symmetricity,
% and fixing affects only the diagonal, so this is not necessary
%[n,m] = size(Sigma);
%if (n == m) & all(all(abs(Sigma - Sigma') < 10*eps*max(abs(diag(Sigma)))));

    % Test for positive definiteness
    [R,p] = chol(Sigma);
    if p == 0
        t = 1;
    else
        t = 0;
    end

%else
%    R = [];
%    t = 0;
%end

% --------------------

function nsigma = imagfixer(sigma);

% force symmetric
nsigma = sigma - (sigma - sigma')/2;
% purge imag
purge = imag(diag(nsigma));
nsigma = nsigma - diag(purge)*1i;

if max(purge) > 1e-4
    warning_wrap('gmmbayes:covfixer:imagfixer', 'Quite big..
    .. imaginary components removed from the diagonal');
end
```

K-Means Algorithm

```
function [ b , mean , error] = KmeansAziz( X , dim  )
%This function clusters given data set using the k-means algorithm
%Input Variables
%X          := input data matrix
%dim        := number of clusters to be obtained

```

```
%Output variables
%b     := calculated labels
%mean  := means of clusters
%error := reconstruction error

[m,n]=size(X);


%Initialize means
%Convergence is independent of starting point so picking up starting
%condition randomly wrt. Forgy—Pertition is suitable
for count=1:dim,

    %Forgy Pertition
    mean(count,:)=X(randint(1,1,[1,m]),:);

end

%Algorithm
criterion=1;
bold=ones(m,dim);
b=zeros(m,dim);

while criterion>0
    b=zeros(m,dim);
    %Calculate labels
    for i=1:m,

        %Calculate label vectors
        for count=1:dim,

            dum(count)=sum((X(i,:)—mean(count,:)).^2);

        end

        ind=find(dum==min(dum));

        b(i,ind)=1;

    end

    %Update means
    for count=1:dim,

        dumx=sum(b(:,count).*X(:,1))/sum(b(:,count));
        dumy=sum(b(:,count).*X(:,2))/sum(b(:,count));
        mean(count,1)=dumx;
        mean(count,2)=dumy;

    end


    %Compute criterion
    criterion=sum(sum((bold—b).^2)));
    bold=b;

end

%Calculate reconstruction error
error=0;
for count=1:dim,
    for i=1:m,

        error=error+b(i,count)*sum((X(i,:)-mean(count,:)).^2);

    end
end

end
```

PCA-Algorithm

```
function [ Z , W ] = PCAAziz( X , dim )
%This function analizes principal components of X and returns transform
%Input variables
%X = input data matrix
%dim = desired output dimension

%Output variables
%Z = mapped data
%W = transform function

[m,n]=size(X);

for count=1:n,

    X(:,count)=X(:,count)-(sum(X(:,count))/m);

end

%Covariance matrix of X
covX=cov(X);

%Find eigenvalues and eigenvectors
[v,d]=eig(covX);
d=max(d).';

%Sort eigenvalues to find maximums and sort vectors with the same indices
[d,ind]=sort(abs(d));
v=v(:,ind);

%Construct transform matrix with eigenvectors corresponding to largest
%eigenvalues
for count=1:dim

    W(:,count)=v(:,end-count+1);

end

Z=(((W.')*(X.')).');


end
```

Gaussian Plot Function in 2D

```
function h = plot_gaussian_ellipsoid(m, C, sdwidth, npts, axh)
% PLOT_GAUSSIAN_ELLIPSOIDS plots 2-d and 3-d Gaussian distributions
%
% H = PLOT_GAUSSIAN_ELLIPSOIDS(M, C) plots the distribution specified by
%  mean M and covariance C. The distribution is plotted as an ellipse (in
%  2-d) or an ellipsoid (in 3-d).  By default, the distributions are
%  plotted in the current axes. H is the graphics handle to the plotted
%  ellipse or ellipsoid.
%
% PLOT_GAUSSIAN_ELLIPSOIDS(M, C, SD) uses SD as the standard deviation
%  along the major and minor axes (larger SD => larger ellipse). By
%  default, SD = 1. Note:
%  * For 2-d distributions, SD=1.0 and SD=2.0 cover ~ 39% and 86%
%     of the total probability mass, respectively.
%  * For 3-d distributions, SD=1.0 and SD=2.0 cover ~ 19% and 73%
%     of the total probability mass, respectively.
%
% PLOT_GAUSSIAN_ELLIPSOIDS(M, C, SD, NPTS) plots the ellipse or
%  ellipsoid with a resolution of NPTS (ellipsoids are generated
%  on an NPTS x NPTS mesh; see SPHERE for more details). By
%  default, NPTS = 50 for ellipses, and 20 for ellipsoids.
%
% PLOT_GAUSSIAN_ELLIPSOIDS(M, C, SD, NPTS, AX) adds the plot to the
%  axes specified by the axis handle AX.
%
% Examples:
% -------------------------------------------
%  % Plot three 2-d Gaussians
%  figure;
%  h1 = plot_gaussian_ellipsoid([1 1], [1 0.5; 0.5 1]);
%  h2 = plot_gaussian_ellipsoid([2 1.5], [1 -0.7; -0.7 1]);
%  h3 = plot_gaussian_ellipsoid([0 0], [1 0; 0 1]);
%  set(h2,'color','r');
%  set(h3,'color','g');
%
%  % "Contour map" of a 2-d Gaussian
%  figure;
%  for sd = [0.3:0.4:4],
%    h = plot_gaussian_ellipsoid([0 0], [1 0.8; 0.8 1], sd);
%  end
%
%  % Plot three 3-d Gaussians
%  figure;
%  h1 = plot_gaussian_ellipsoid([1 1  0], [1 0.5 0.2; 0.5 1 0.4; 0.2 0.4 1]);
%  h2 = plot_gaussian_ellipsoid([1.5 1 .5], [1 -0.7 0.6; -0.7 1 0; 0.6 0 1]);
%  h3 = plot_gaussian_ellipsoid([1 2 2], [0.5 0 0; 0 0.5 0; 0 0 0.5]);
%  set(h2,'facealpha',0.6);
%  view(129,36); set(gca,'proj','perspective'); grid on;
%  grid on; axis equal; axis tight;
% -------------------------------------------
%
%  Gautam Vallabha, Sep-23-2007, Gautam.Vallabha@mathworks.com

%  Revision 1.0, Sep-23-2007
%    - File created
%  Revision 1.1, 26-Sep-2007
%    - NARGOUT==0 check added.
%    - Help added on NPTS for ellipsoids

if ~exist('sdwidth', 'var'), sdwidth = 1; end
if ~exist('npts', 'var'), npts = []; end
if ~exist('axh', 'var'), axh = gca; end

if numel(m) ~= length(m),
    error('M must be a vector');
end
if ~( all(numel(m) == size(C)) )
```

```
    error('Dimensionality of M and C must match');
end
if ~(isscalar(axh) && ishandle(axh) && strcmp(get(axh,'type'), 'axes'))
    error('Invalid axes handle');
end

set(axh, 'nextplot', 'add');

switch numel(m)
   case 2, h=show2d(m(:),C,sdwidth,npts,axh);
   case 3, h=show3d(m(:),C,sdwidth,npts,axh);
   otherwise
      error('Unsupported dimensionality');
end

if nargout==0,
    clear h;
end

%-----------------------------
function h = show2d(means, C, sdwidth, npts, axh)
if isempty(npts), npts=50; end
% plot the gaussian fits
tt=linspace(0,2*pi,npts)';
x = cos(tt); y=sin(tt);
ap = [x(:) y(:)]';
[v,d]=eig(C);
d = sdwidth * sqrt(d); % convert variance to sdwidth*sd
bp = (v*d*ap) + repmat(means, 1, size(ap,2));
h = plot(bp(1,:), bp(2,:), '-', 'parent', axh);

%-----------------------------
function h = show3d(means, C, sdwidth, npts, axh)
if isempty(npts), npts=20; end
[x,y,z] = sphere(npts);
ap = [x(:) y(:) z(:)]';
[v,d]=eig(C);
if any(d(:) < 0)
   fprintf('warning: negative eigenvalues\n');
   d = max(d,0);
end
d = sdwidth * sqrt(d); % convert variance to sdwidth*sd
bp = (v*d*ap) + repmat(means, 1, size(ap,2));
xp = reshape(bp(1,:), size(x));
yp = reshape(bp(2,:), size(y));
zp = reshape(bp(3,:), size(z));
h = surf(axh, xp,yp,zp);
```

mRMR-Algoirthm

```
function [fea] = mrmr_mid_d(d, f, K)
% function [fea] = mrmr_mid_d(d, f, K)
%
% MID scheme according to MRMR
%
% By Hanchuan Peng
% April 16, 2003
%

bdisp=0;

```

```
%Modified part
mutualinfo= @(x1,x2) abs(corr(x1,x2));

nd = size(d,2);
nc = size(d,1);

t1=cputime;
for i=1:nd,
   t(i) = mutualinfo(d(:,i), f);
end;
fprintf('calculate the marginal dmi costs %5.1fs.\n', cputime-t1);

[tmp, idxs] = sort(-t);
fea_base = idxs(1:K);

fea(1) = idxs(1);

KMAX = min(1000,nd); %500

idxleft = idxs(2:KMAX);

k=1;
if bdisp==1,
fprintf('k=1 cost_time=(N/A) cur_fea=%d #left_cand=%d\n', ...
      fea(k), length(idxleft));
end;

for k=2:K,
   t1=cputime;
   ncand = length(idxleft);
   curlastfea = length(fea);
   for i=1:ncand,
      t_mi(i) = mutualinfo(d(:,idxleft(i)), f);
      mi_array(idxleft(i),curlastfea) = getmultimi(d(:,fea(curlastfea)), d(:,idxleft(i)));
      c_mi(i) = mean(mi_array(idxleft(i), :));
   end;

   [tmp, fea(k)] = max(t_mi(1:ncand) - c_mi(1:ncand));

   tmpidx = fea(k); fea(k) = idxleft(tmpidx); idxleft(tmpidx) = [];

   if bdisp==1,
   fprintf('k=%d cost_time=%5.4f cur_fea=%d #left_cand=%d\n', ...
      k, cputime-t1, fea(k), length(idxleft));
   end;
end;

return;

%=====================================
%Modified as well
function c = getmultimi(da, dt)
for i=1:size(da,2),
   c(i) = abs(corr(da(:,i), dt));
end;
```