6601 – Assignment 3: GMM and Image Segmentation

Instructor: Thad Starner TA: Daniel Kohlsdorf, Titilayo Craig

Weiren Wang(903076444) weirenwang@gatech.edu

https://www.linkedin.com/in/weirenwang

https://github.com/JeffreyWeirenWang

Implement: Gaussian Mixture Models and EM(30%)

To segment the images, we use EM algorithm combined with Gaussian Mixture Models and for each pixel we use the maximum a posteriori probability to do the clustering. There are 526*700 pixels. We extract each pixel as a data point. We implement the EM algorithm as follows:

EM Algorithm:

Repeat until convergence {

(E-step) For each i, set

$$Q_i\left(z^{(i)}\right) := p\left(z^{(i)} \middle| x^{(i)}; \theta\right)$$

(M-step) Set

$$\theta := argmax_\theta \ \sum_i \sum_{z^{(i)}} Q_i\left(z^{(i)}\right) log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i\left(z^{(i)}\right)}$$

}

The result:

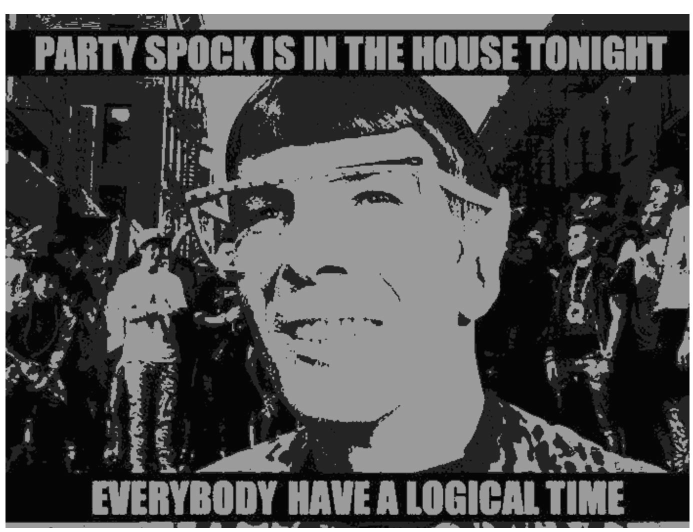

Figure 1: EM image segmentation with 3 components

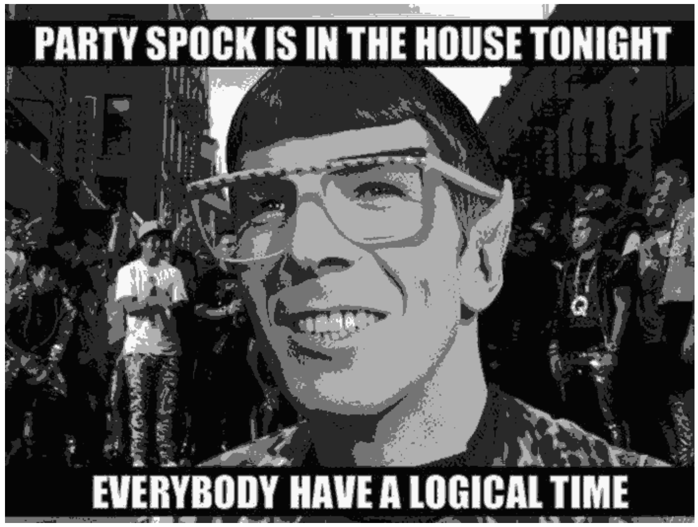

Figure 2: EM image segmentation with 5 components

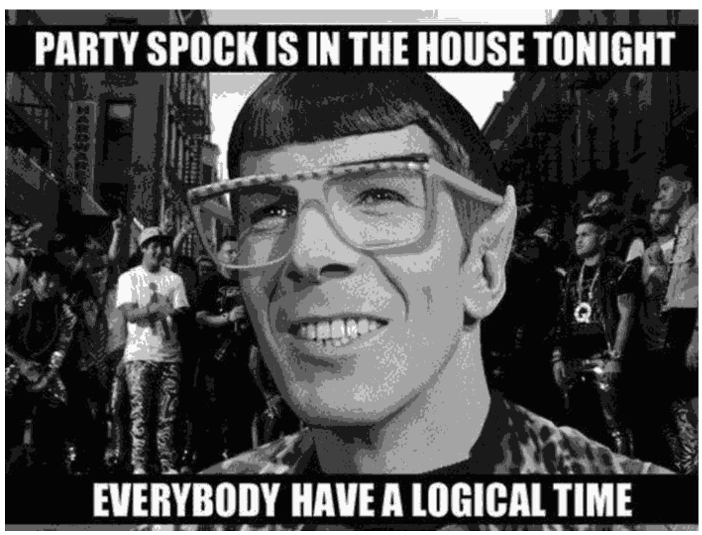

Figure 3: EM image segmentation with 8 components

Experiment: Another Initialization(40%)

For initialization, we could use a clustering algorithm. This time, we use the result of k-means clustering as the initialization for the EM algorithm,
We use K=5 components and run the procedure with random initialization and –means initialization 100 runs. The result as show follows:

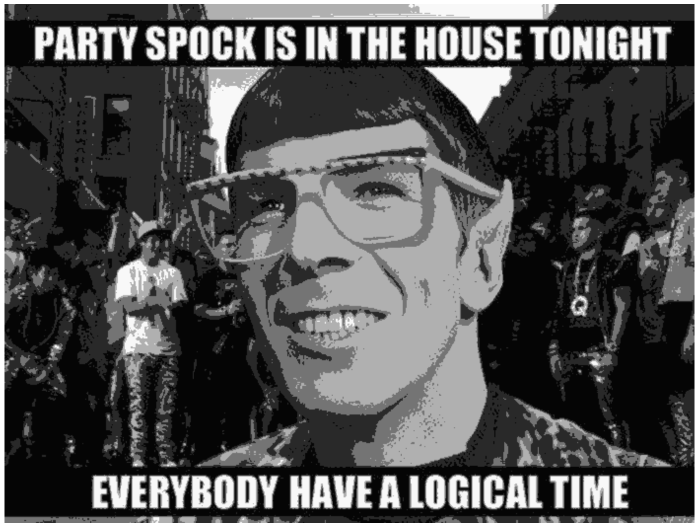

Figure 4: K-means clustering initialization

Research: Bayesian Information Criterion and Mode Selection(30%)

a. 8 free parameters. Because the entire possibility sum is 1. $\pi_k, \mu_k, \sigma_k, k = 1,2,3$.

b. In Bayesian information criterion (BIC) model, it introduce a penalty term to prevent the overfitting when selecting model from a finite number of models. L = -200,N=50,k=8(Gaussian mixture model with 3 components)

$$BIC = -2\times \mathrm{L} + \mathrm{k}\times \ln(\mathrm{N}) = 431.3$$

c. L = -200, N=50,k=299(Gaussian mixture model with 100 components),

$$BIC = -2\times \mathrm{L} + \mathrm{k}\times \ln(\mathrm{N}) = 1569.7$$

d. When L=-2000, N=50, k=8(Gaussian mixture model with 3 components),

$$BIC = -2\times \mathrm{L} + \mathrm{k}\times \ln(\mathrm{N}) = 4031.3$$

e. BIC decreases as L increase when k and N are fixed. For a given model and the same number of data points, a smaller BIC indicates a larger L. When L and N are fixed, BIC increases as k increase, for the same number of data points and the same likelihood level, BIC increases with number of free parameters.

f. BIC is helpful to find a K and choose a model: we can simply choose a model with a smaller BIC. Since a larger L, i.e. a larger likelihood indicates a better fit, and results with a smaller BIC. Furthermore, a larger number of parameters is more likely to cause overfitting, thus a penalty is needed. This is exactly what BIC can do, since a larger k indicates a larger BIC.

Appendix

```
function [W,M,V,L] = EM_GM(X,k,ltol,maxiter,pflag,Init)
% [W,M,V,L] = EM_GM(X,k,ltol,maxiter,pflag,Init)
%
% EM algorithm for k multidimensional Gaussian mixture estimation
%
% Inputs:
%   X(n,d) - input data, n=number of observations, d=dimension of
variable
%   k - maximum number of Gaussian components allowed
%   ltol - percentage of the log likelihood difference between 2
iterations ([] for none)
%   maxiter - maximum number of iteration allowed ([] for none)
%   pflag - 1 for plotting GM for 1D or 2D cases only, 0 otherwise ([]
for none)
%   Init - structure of initial W, M, V: Init.W, Init.M, Init.V ([] for
none)
%
% Ouputs:
%   W(1,k) - estimated weights of GM
%   M(d,k) - estimated mean vectors of GM
%   V(d,d,k) - estimated covariance matrices of GM
%   L - log likelihood of estimates
%
% Written by
%   Patrick P. C. Tsui,
%   PAMI research group
%   Department of Electrical and Computer Engineering
%   University of Waterloo,
%   March, 2006
%

%%%% Validate inputs %%%%
if nargin <= 1,
    disp('EM_GM must have at least 2 inputs: X,k!/n')
    return
elseif nargin == 2,
    ltol = 0.1; maxiter = 1000; pflag = 0; Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    if err_X | err_k, return; end
elseif nargin == 3,
    maxiter = 1000; pflag = 0; Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    if err_X | err_k | err_ltol, return; end
elseif nargin == 4,
    pflag = 0;  Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    [maxiter,err_maxiter] = Verify_maxiter(maxiter);
    if err_X | err_k | err_ltol | err_maxiter, return; end
elseif nargin == 5,
```

```
     Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    [maxiter,err_maxiter] = Verify_maxiter(maxiter);
    [pflag,err_pflag] = Verify_pflag(pflag);
    if err_X | err_k | err_ltol | err_maxiter | err_pflag, return; end
elseif nargin == 6,
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    [maxiter,err_maxiter] = Verify_maxiter(maxiter);
    [pflag,err_pflag] = Verify_pflag(pflag);
    [Init,err_Init]=Verify_Init(Init);
    if err_X | err_k | err_ltol | err_maxiter | err_pflag | err_Init,
return; end
else
    disp('EM_GM must have 2 to 6 inputs!');
    return
end

%%%% Initialize W, M, V,L %%%%
t = cputime;
if isempty(Init),
    [W,M,V] = Init_EM(X,k); L = 0;
else
    W = Init.W;
    M = Init.M;
    V = Init.V;
end
Ln = Likelihood(X,k,W,M,V); % Initialize log likelihood
Lo = 2*Ln;

%%%% EM algorithm %%%%
niter = 0;
while (abs(100*(Ln-Lo)/Lo)>ltol) & (niter<=maxiter),
    E = Expectation(X,k,W,M,V); % E-step
    [W,M,V] = Maximization(X,k,E);  % M-step
    Lo = Ln;
    Ln = Likelihood(X,k,W,M,V);
    niter = niter + 1;
end
L = Ln;

%%%% Plot 1D or 2D %%%%
if pflag==1,
    [n,d] = size(X);
    if d>2,
        disp('Can only plot 1 or 2 dimensional applications!/n');
    else
        Plot_GM(X,k,W,M,V);
    end
    elapsed_time = sprintf('CPU time used for EM_GM: %5.2fs',cputime-
t);
    disp(elapsed_time);
    disp(sprintf('Number of iterations: %d',niter-1));
end
%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%% End of EM_GM %%%%
%%%%%%%%%%%%%%%%%%%%%%

function E = Expectation(X,k,W,M,V)
[n,d] = size(X);
a = (2*pi)^(0.5*d);
S = zeros(1,k);
iV = zeros(d,d,k);
for j=1:k,
    if V(:,:,j)==zeros(d,d), V(:,:,j)=ones(d,d)*eps; end
    S(j) = sqrt(det(V(:,:,j)));
    iV(:,:,j) = inv(V(:,:,j));
end
E = zeros(n,k);
for i=1:n,
    for j=1:k,
        dXM = X(i,:)'-M(:,j);
        pl = exp(-0.5*dXM'*iV(:,:,j)*dXM)/(a*S(j));
        E(i,j) = W(j)*pl;
    end
    E(i,:) = E(i,:)/sum(E(i,:));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Expectation %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [W,M,V] = Maximization(X,k,E)
[n,d] = size(X);
W = zeros(1,k); M = zeros(d,k);
V = zeros(d,d,k);
for i=1:k,  % Compute weights
    for j=1:n,
        W(i) = W(i) + E(j,i);
        M(:,i) = M(:,i) + E(j,i)*X(j,:)';
    end
    M(:,i) = M(:,i)/W(i);
end
for i=1:k,
    for j=1:n,
        dXM = X(j,:)'-M(:,i);
        V(:,:,i) = V(:,:,i) + E(j,i)*dXM*dXM';
    end
    V(:,:,i) = V(:,:,i)/W(i);
end
W = W/n;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Maximization %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function L = Likelihood(X,k,W,M,V)
% Compute L based on K. V. Mardia, "Multivariate Analysis", Academic
Press, 1979, PP. 96-97
% to enchance computational speed
[n,d] = size(X);
U = mean(X)';
S = cov(X);
L = 0;
for i=1:k,
```

```
    iV = inv(V(:,:,i));
    L = L + W(i)*(-0.5*n*log(det(2*pi*V(:,:,i))) ...
        -0.5*(n-1)*(trace(iV*S)+(U-M(:,i))'*iV*(U-M(:,i))));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Likelihood %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function err_X = Verify_X(X)
err_X = 1;
[n,d] = size(X);
if n<d,
    disp('Input data must be n x d!/n');
    return
end
err_X = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_X %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

function err_k = Verify_k(k)
err_k = 1;
if ~isnumeric(k) | ~isreal(k) | k<1,
    disp('k must be a real integer >= 1!/n');
    return
end
err_k = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_k %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

function [ltol,err_ltol] = Verify_ltol(ltol)
err_ltol = 1;
if isempty(ltol),
    ltol = 0.1;
elseif ~isreal(ltol) | ltol<=0,
    disp('ltol must be a positive real number!');
    return
end
err_ltol = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_ltol %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [maxiter,err_maxiter] = Verify_maxiter(maxiter)
err_maxiter = 1;
if isempty(maxiter),
    maxiter = 1000;
elseif ~isreal(maxiter) | maxiter<=0,
    disp('ltol must be a positive real number!');
    return
end
err_maxiter = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_maxiter %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
function [pflag,err_pflag] = Verify_pflag(pflag)
err_pflag = 1;
if isempty(pflag),
    pflag = 0;
elseif pflag~=0 & pflag~=1,
    disp('Plot flag must be either 0 or 1!/n');
    return
end
err_pflag = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_pflag %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Init,err_Init] = Verify_Init(Init)
err_Init = 1;
if isempty(Init),
    % Do nothing;
elseif isstruct(Init),
    [Wd,Wk] = size(Init.W);
    [Md,Mk] = size(Init.M);
    [Vd1,Vd2,Vk] = size(Init.V);
    if Wk~=Mk | Wk~=Vk | Mk~=Vk,
        disp('k in Init.W(1,k), Init.M(d,k) and Init.V(d,d,k) must
equal!/n')
        return
    end
    if Md~=Vd1 | Md~=Vd2 | Vd1~=Vd2,
        disp('d in Init.W(1,k), Init.M(d,k) and Init.V(d,d,k) must
equal!/n')
        return
    end
else
    disp('Init must be a structure: W(1,k), M(d,k), V(d,d,k) or []!');
    return
end
err_Init = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_Init %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [W,M,V] = Init_EM(X,k)
[n,d] = size(X);
[Ci,C] = kmeans(X,k,'Start','cluster', ...
    'Maxiter',100, ...
    'EmptyAction','drop', ...
    'Display','off'); % Ci(nx1) - cluster indeices; C(k,d) - cluster
centroid (i.e. mean)
while sum(isnan(C))>0,
    [Ci,C] = kmeans(X,k,'Start','cluster', ...
        'Maxiter',100, ...
        'EmptyAction','drop', ...
        'Display','off');
end
M = C';
Vp = repmat(struct('count',0,'X',zeros(n,d)),1,k);
for i=1:n, % Separate cluster points
    Vp(Ci(i)).count = Vp(Ci(i)).count + 1;
    Vp(Ci(i)).X(Vp(Ci(i)).count,:) = X(i,:);
```

```
end
V = zeros(d,d,k);
for i=1:k,
    W(i) = Vp(i).count/n;
    V(:,:,i) = cov(Vp(i).X(1:Vp(i).count,:));
end
%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Init_EM %%%%
%%%%%%%%%%%%%%%%%%%%%%%%

function Plot_GM(X,k,W,M,V)
[n,d] = size(X);
if d>2,
    disp('Can only plot 1 or 2 dimensional applications!/n');
    return
end
S = zeros(d,k);
R1 = zeros(d,k);
R2 = zeros(d,k);
for i=1:k,  % Determine plot range as 4 x standard deviations
    S(:,i) = sqrt(diag(V(:,:,i)));
    R1(:,i) = M(:,i)-4*S(:,i);
    R2(:,i) = M(:,i)+4*S(:,i);
end
Rmin = min(min(R1));
Rmax = max(max(R2));
R = [Rmin:0.001*(Rmax-Rmin):Rmax];
clf, hold on
if d==1,
    Q = zeros(size(R));
    for i=1:k,
        P = W(i)*normpdf(R,M(:,i),sqrt(V(:,:,i)));
        Q = Q + P;
        plot(R,P,'r-'); grid on,
    end
    plot(R,Q,'k-');
    xlabel('X');
    ylabel('Probability density');
else % d==2
    plot(X(:,1),X(:,2),'r.');
    for i=1:k,
        Plot_Std_Ellipse(M(:,i),V(:,:,i));
    end
    xlabel('1^{st} dimension');
    ylabel('2^{nd} dimension');
    axis([Rmin Rmax Rmin Rmax])
end
title('Gaussian Mixture estimated by EM');
%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Plot_GM %%%%
%%%%%%%%%%%%%%%%%%%%%%%%

function Plot_Std_Ellipse(M,V)
[Ev,D] = eig(V);
d = length(M);
if V(:,:)==zeros(d,d),
    V(:,:) = ones(d,d)*eps;
end
```

```
iV = inv(V);
% Find the larger projection
P = [1,0;0,0];  % X-axis projection operator
P1 = P * 2*sqrt(D(1,1)) * Ev(:,1);
P2 = P * 2*sqrt(D(2,2)) * Ev(:,2);
if abs(P1(1)) >= abs(P2(1)),
    Plen = P1(1);
else
    Plen = P2(1);
end
count = 1;
step = 0.001*Plen;
Contour1 = zeros(2001,2);
Contour2 = zeros(2001,2);
for x = -Plen:step:Plen,
    a = iV(2,2);
    b = x * (iV(1,2)+iV(2,1));
    c = (x^2) * iV(1,1) - 1;
    Root1 = (-b + sqrt(b^2 - 4*a*c))/(2*a);
    Root2 = (-b - sqrt(b^2 - 4*a*c))/(2*a);
    if isreal(Root1),
        Contour1(count,:) = [x,Root1] + M';
        Contour2(count,:) = [x,Root2] + M';
        count = count + 1;
    end
end
Contour1 = Contour1(1:count-1,:);
Contour2 = [Contour1(1,:);Contour2(1:count-1,:);Contour1(count-1,:)];
plot(M(1),M(2),'k+');
plot(Contour1(:,1),Contour1(:,2),'k-');
plot(Contour2(:,1),Contour2(:,2),'k-');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Plot_Std_Ellipse %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```