ECE 271A Problem Set 5.

Jun Hao (Simon) Hu

December 7, 2018

1 Part 1, $C = 8$.

1.1 Results.

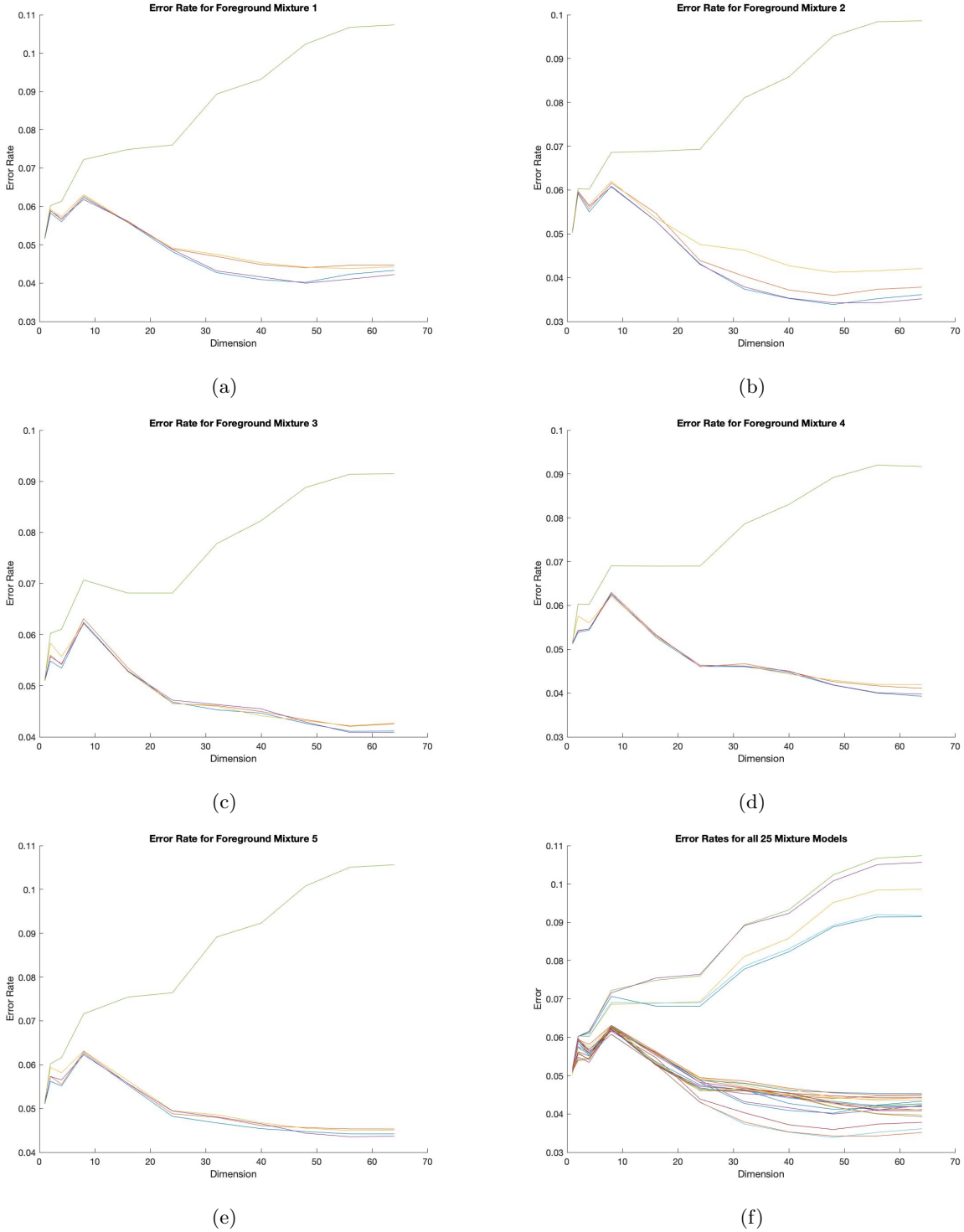

Figure 1: (a)-(e) Plots of the classification error as a function of the dimension, for the five background (BG) mixture models, with the corresponding FG mixture model. (f) Plot of the classification error as a function of the dimension for all 25 mixture combinations.

1.2 Analysis.

One immediate observation is that the probability of error (POE), a way to measure the effectiveness of a classifier, is highly dependent on the intialization of the parameters. In figure 1, notice that there is one BG/FG mixture model pair that fares much worse than the others. The other pairs, compared to this one, have a lower POE and this difference becomes more apparent as the dimension size increases. As the dimension size increases we see that the initialization choice for the parameters can make the difference between an error of approximately 0.15 or an error of approximately 0.045.

To explain the importance of a good initialization we rely on ideas from optimization theory. The EM algorithm is at its core, an *iterative* algorithm that approximates the likelihood function in the presence of incomplete data. To be more specific, the EM algorithm can be used to approximate, or estimate, the parameters that define the likelihood function. In the case of the mixture of Gaussians (MOG) model, we estimate the likelihood function using C weighted Gaussians with mean μ_c and σ_c, weighted by π_c. To estimate these parameters, the EM algorithm must maximize some function f that depends on the parameters θ, with respect to θ. In general it is safe to assume that we will not always have access to information about f itself (in particular we are interested in the number of maximums and minimums it attains). In the case where there are multiple maximums, the choice of our initial guess (which is required since EM is an iterative algorithm) becomes very important. Each maximum point has what is called a *basin of attraction*. There is a theorem in the calculus of variations which states that all fixed points of some bounded functional I that satisfies the Banach Fixed-Point theorem have a ball centered at the fixed point with radius r such that initial guesses within that ball converge to the origin. To conclude, if a function f has multiple maximums (or minimums) then our initial guess converges to the nearest maximum. Hence the bad FG/BG pair described above may have been the result of an initialization that led to another maximum point.

2 Part 2, $C \in \{1, 2, 4, 8, 16, 32\}$.

2.1 Results.

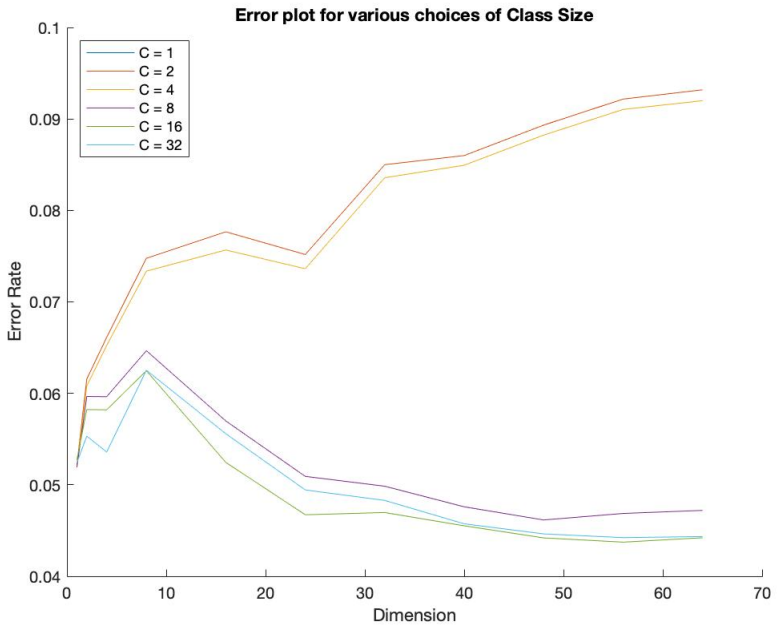

Figure 2: Plot of the classification error as a function of the dimension, for different choices of C.

2.2 Analysis.

One immediate observation is the POE decreases as the number of classes and dimension size increases. In particular, when $C = 8, 16, 32$ we are able to achieve a POE less than 0.05, as the dimension size increased. Under the MOG model, we essentially estimate the likelihood function by a weighted linear combination of Gaussians. By increasing the number of Gaussians used to estimate the likelihood function, we have a better change of fitting more complicated functions. This makes the classifier a bit more robust. However, we must be careful with the choice of C, since increasing the number of Gaussian classes *also* increases the computation time, and we may have issues with overfitting/overestimating.

3 Conclusion.

From the results above we cocnlude that the effectiveness of the classifier highly depends on the initialization of the parameters, the dimension of the classifier, and the nunmber of Gaussian classes used for estimating the likelihood function. The best POEs obtained from the EM method for the MOG model were much better than the POEs obtained in previous problem sets.

4 MATLAB Code.

```
% Written by Jun Hao Hu, University of California San Diego.
% All rights reserved.

%------------------------------------------------------------------------
% Main MATLAB routine for homework 4.
%------------------------------------------------------------------------

%% Load data and define parameters.
load('TrainingSamplesDCT_8_new.mat');

cheetah_ori = im2double(imread('cheetah.bmp')); % Convert the image into the range [0,1].
cheetah_pad = padarray(cheetah_ori, [4 4], 'both');
zig = load('Zig-Zag Pattern.txt');
zig = zig+1;
M = 64;

[rows_cheetah_ori,cols_cheetah_ori] = size(cheetah_ori);
[rows_cheetah_pad,cols_cheetah_pad] = size(cheetah_pad);
[rows_fg,~] = size(TrainsampleDCT_FG);
[rows_bg,~] = size(TrainsampleDCT_BG);

p_fg = rows_fg/(rows_fg+rows_bg); % Calculate the priors using the MLE.
p_bg = rows_bg/(rows_bg+rows_fg); % Calculate the priors using the MLE.

dims = [1 2 4 8 16 24 32 40 48 56 64]; % Desired dimensions.
[~,n_dims] = size(dims);
n_class = 8; % C = 8.
n_mix = 5; % Part 1 is to train five mixtures.

mu_fg = zeros(n_mix,M*n_class);
sigma_fg = zeros(n_mix,M*n_class);
pi_fg = zeros(n_mix,n_class);
mu_bg = zeros(n_mix,M*n_class);
sigma_bg = zeros(n_mix,M*n_class);
```

```
pi_bg = zeros(n_mix,n_class);
%% Training.
fprintf('Loading completed...starting training.\n');
tic;
for idx_mix = 1:n_mix
    % idx_mix % For debugging purposes. Comment later.
    [mu_fg(idx_mix,:),sigma_fg(idx_mix,:),pi_fg(idx_mix,:)] = em_calc(TrainsampleDCT_FG,
        n_class,M,200);
    [mu_bg(idx_mix,:),sigma_bg(idx_mix,:),pi_bg(idx_mix,:)] = em_calc(TrainsampleDCT_BG,
        n_class,M,200);
end
toc;
fprintf('\nTraining completed...starting prediction.');
%% Prediction and error analysis.
% To save on time, calculate the DFT coefficients of the image before-hand.
%dct_coeffs = zeros(rows_cheetah_ori*cols_cheetah_ori,M);
p_err_25 = zeros(n_mix*n_mix,n_dims);
%A_25 = zeros(rows_cheetah_ori,cols_cheetah_ori,n_mix*n_mix);
%
% idx_track = 1;
% for idx_x = 5:rows_cheetah_pad-6
%     for idx_y = 5:cols_cheetah_pad-5
%         temp_block = dct2(cheetah_pad(idx_x-4:idx_x+3,idx_y-4:idx_y+3));
%         v(zig(:)) = temp_block(:);
%         dct_coeffs(idx_track,:) = v;
%         idx_track = idx_track+1;
%     end
% end
dct_coeffs = get_dct(cheetah_ori,zig);

% Perform the BDR prediction.
tic;
for idx_mix1 = 1:5
    for idx_mix2 = 1:5
        for idx_dim = 1:n_dims
            A_25 = bdr_predict(dct_coeffs,dims(idx_dim),...
                mu_fg(idx_mix1,:),mu_bg(idx_mix2,:),sigma_fg(idx_mix1,:),sigma_bg(idx_mix2
                    ,:),...
                pi_fg(idx_mix1,:),pi_bg(idx_mix2,:),p_fg,p_bg,rows_cheetah_ori,
                    cols_cheetah_ori,n_class);
            p_err_25((idx_mix1-1)*5+idx_mix2,idx_dim) = calc_error(A_25,p_fg,p_bg);
        end
    end
end
toc;
% Plot the results from the 25 different classifiers.
% 12/05/2018 : Leave the plotting until you have all of the data from the
% training and predicting.
```

```
% Written by Jun Hao Hu, University of California San Diego.
% All rights reserved.

%-------------------------------------------------------------------
% MATLAB program to perform expectation-maximization (EM).
```

```
% Input:
%   dct_coeffs : vector containing the DCT coefficients
%   n_class : int containing the number of classes
%   n_dim : int containing the number of dimensions we want
%   num_iter : int containing the maximum number of iterations for EM
% Output:
%   mu : the EM estimate of the mean, mu
%   sigma : the EM estimate of the covariance, sigma
%   pi_c : the EM estimate of the weights for the mixture model
%--------------------------------------------------------------------

function [mu,sigma,pi_c] = em_calc(dct_coeffs,n_class,n_dim,num_iter)

% Assign parameters for EM algorithm.

[n_rows, ~] = size(dct_coeffs);
dct_coeffs_dim = dct_coeffs(:,1:n_dim);

sigma_c = diag(diag(2*rand(n_dim*n_class,n_dim*n_class)+2)); % Initialization of sigma.
mu_c = 3*rand(n_class,n_dim)+3; % Initialization of mu.
pi_c = (randi(20,1,n_class)); % Initialization of the weights, pi_c.
pi_c = pi_c/sum(pi_c); % Constraint equation.
epsilon = (1e-06)*ones(size(sigma_c)); % Small paramater to prevent zero covariance.
z = zeros(n_rows,n_class); % Initialization of the vector z.

% Start the EM algorithm.
for iter = 1:num_iter
    % Expectation step.
    z_old = z;
    for idx_exp = 1:n_rows
        p_x = zeros(1,n_class); % Pre-allocating space for likelihood of x.
        for idx_c = 1:n_class
            split_1 = (idx_c-1)*n_dim; % Splitting index 1 for convenience.
            split_2 = idx_c*n_dim; % Splitting index 2 for convenience.
            sigma = sigma_c(split_1+1:split_2,split_1+1:split_2); % Update for sigma.
            mu = mu_c(idx_c,:); % Update for mu.
            p_x(idx_c) = mvnpdf(dct_coeffs_dim(idx_exp,:),mu,sigma)*...
                pi_c(idx_c); % Calculate the likelihood of x.
        end
       z(idx_exp,:) = p_x/sum(p_x); % Normalization of the vector.
    end
    pi_c = sum(z)/n_rows; % Update for pi_c.

    % Maximization step.
    for idx_c = 1:n_class
        split_1 = (idx_c-1)*n_dim; % Splitting index 1 for convenience.
        split_2 = idx_c*n_dim; % Splitting index 2 for convenience.
        sig = (dct_coeffs_dim-repmat(mu_c(idx_c,:),n_rows,1));
        sigma = sig.*(repmat(z(:,idx_c),1,n_dim));
        tot = sum(z(:,idx_c));
        sigma_c(split_1+1:split_2,split_1+1:split_2) = (sigma'*sig)/tot;
        mu_c(idx_c,:) = sum(dct_coeffs_dim.*repmat(z(:,idx_c),1,n_dim))/tot;
    end
    sigma_c = diag(diag(sigma_c + epsilon));
```

```
    if (log(z) - log(z_old)) < 1e-06
           break;
    end
end

% Grab the correct parameters after EM calculation.
mu = zeros(1,n_dim*n_class);
for idx_c = 1:n_class
    mu((idx_c-1)*n_dim+1:idx_c*n_dim) = mu_c(idx_c,:);
end
sigma = diag(sigma_c).';
end
```

```
% Written by Jun Hao Hu, University of California San Diego.
% All rights reserved.

%---------------------------------------------------------------------------
% MATLAB routine that grabs the DCT coefficients of the image.
%---------------------------------------------------------------------------

function dct_coeffs = get_dct(cheetah_img,zig)

[cheetah_rows,cheetah_cols] = size(cheetah_img);
M = 64;
dct_coeffs = zeros(cheetah_rows*cheetah_cols,M);
v = zeros(1,M);
for idx_x = 1:cheetah_rows
    for idx_y = 1:cheetah_cols
        x = idx_x-4;
        y = idx_y-4;

        % Corner/edge cases
        if x < 1
            x = 1;
        end
        if y < 1
            y = 1;
        end
        if x+7>cheetah_rows
            x = cheetah_rows-7;
        end
        if y+7>cheetah_cols
            y = cheetah_cols-7;
        end
        dct_block = dct2(cheetah_img(x:x+7,y:y+7));
        v(zig(:)) = dct_block(:);
        dct_coeffs(cheetah_cols*(idx_x-1)+idx_y,:) = v;
    end
end
end
```

```
% Written by Jun Hao Hu, University of California San Diego.
% All rights reserved.

```

```
%-------------------------------------------------------------------------
% MATLAB routine that performs prediction using BDR and calculates the
% errors as well.
%-------------------------------------------------------------------------

function [A] = bdr_predict(dct_coeffs,dim,mu_fg,mu_bg,sigma_fg,sigma_bg,pi_fg,pi_bg,p_fg,
    p_bg,rows,cols,n_class)
A = zeros(rows,cols);

for idx_x = 1:rows
    for idx_y = 1:cols
        idx = cols*(idx_x-1)+idx_y;
        dct_coeff = dct_coeffs(idx,:);
        A(idx_x,idx_y) = p_fg*calc_prob(dct_coeff,dim,n_class,mu_fg,sigma_fg,pi_fg) > ...
            p_bg*calc_prob(dct_coeff,dim,n_class,mu_bg,sigma_bg,pi_bg);
    end
end
end
```

```
% Written by Jun Hao Hu, University of California San Diego.
% All rights reserved.

%-------------------------------------------------------------------------
% MATLAB routine that takes the data and calculates the posterior
% probability.
%-------------------------------------------------------------------------

function [dct_likelihood] = calc_prob(dct_coeffs,n_dim,n_class,mu,sigma,pi)
mu_c = zeros(n_class,n_dim);
sigma_c = zeros(n_class*n_dim,n_class*n_dim);
dct_coeffs = dct_coeffs(1:n_dim); % Only keep the first dim dimensions of the DCT
    coefficients.
dct_likelihood = 0;

for idx = 1:n_class
   split_1 = (idx-1)*64;
   split_2 = (idx-1)*64;
   mu_c(idx,:) = mu(split_1+1:split_2+n_dim);
   sigma_c((idx-1)*n_dim+1:idx*n_dim,(idx-1)*n_dim+1:idx*n_dim) = diag(sigma(split_1+1:
       split_2+n_dim)); % Diagonal covariance matrix.
   dct_likelihood = dct_likelihood + mvnpdf(...
       dct_coeffs,mu_c(idx,:),sigma_c((idx-1)*n_dim+1:idx*n_dim,(idx-1)*n_dim+1:idx*n_dim))
           ...
       *pi(idx);
end
end
```

```
% Written by Jun Hao Hu, University of California San Diego.
% All rights reserved.

%-------------------------------------------------------------------------
% MATLAB routine that takes the mask obtained through BDR for error
% calculations.
%-------------------------------------------------------------------------
```

```
8
9  function [p_error] = calc_error(A,p_fg,p_bg)
10 cheetah_mask = imread('cheetah_mask.bmp');
11 cheetah_mask = cheetah_mask == 255;
12
13 cheetah_mask_one = find(cheetah_mask);
14 cheetah_mask_zero = find(~cheetah_mask);
15
16 num_incorrect_zero = sum(A(cheetah_mask_zero) == 1);
17 num_incorrect_one = sum(A(cheetah_mask_one) == 0);
18
19 p_error = (num_incorrect_zero*p_bg)/sum(sum(cheetah_mask == 0)) +...
20     (num_incorrect_one*p_fg)/sum(sum(cheetah_mask == 1));
21
22 end
```