**a)**

A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generated

*Figure 1: Probability of error vs number of dimensions for classifier combinations 1 - 5*

A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generated

*Figure 2: Probability of error vs number of dimensions for classifier combinations 6 - 10*

A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generated

*Figure 3: Probability of error vs number of dimensions for classifier combinations 11 - 15*

A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA screenshot of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generated

*Figure 4: Probability of error vs number of dimensions for classifier combinations 16 - 20*

A close up of a map

Description automatically generatedA screenshot of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generated

*Figure 5: Probability of error vs number of dimensions for classifier combinations 21 – 25*

All 25 combinations of classifiers generally follow the same trend: as the number of dimensions increases the error percentage decreases, but after about 30 (in most cases) or 40 (in a few cases) dimensions are used the error percentage begins to increase. There are only 2 combinations that result in a continuous decrease in error percentage up to 48 dimensions. This behavior is likely due to overfitting. Based on the extra computation time it took to run the classifications for >30 dimensions, and the overall benefit of using more dimensions, it is not worth performing classifications past 30 dimensions as the error percentage only decreased by a miniscule amount in a few combinations and increased in the vast majority of combinations. The minimum values of error percentage ranged from 5.0281% (combination 18) to 5.3641% (combination 10) and the maximum values of error percentage ranged from 6.5509% (in combination 14) to 6.7510% (combination 3).

The only aspect that was different between the classification combinations was the random initialization of the classifier parameters. It is clear that the initialization step for the parameters is important as the approximately 0.3% and 0.2% difference in the minimum and maximum error percentages respectively is not negligible in most classification/machine learning applications. I created a distribution of values and randomly selected values from the distribution to initialize the parameters. I provided plots of the cheetah classifications for combination 1 in Figure 6, and as one can see the random initialization works quite well. However, using k-means would be a more robust approach to parameter initialization and would aid in minimizing the error percentage difference between combinations.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

*Figure 6: Classification results for varying dimensions of combination 1*

As a side note, I used the same number of iterations (300) for learning parameters. This number was chosen by running approximately 20 convergence tests to observe how many iterations were generally required for convergence. The number of iterations ranged from approximately 75 to 250 in the tests I ran – so just to be safe I set the number of iterations to 300 to ensure everything converged. A delta limit would be a more robust approach, however for the sake of this assignment setting a high iteration count worked just fine and the extra computation time was negligible. Figure 7 displays an example of one of the convergence tests.

A screenshot of a cell phone

Description automatically generated

*Figure 7: Convergence test example (delta of parameters vs iteration)*

**A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedb)**

**A close up of a map

Description automatically generatedA close up of a map

Description automatically generated**

**A screenshot of a map

Description automatically generatedA close up of a map

Description automatically generated**

*Figure 8: Probability of error vs number of dimensions for varying components*

As one can see in Figure 8, the number of dimensions has varying affects on the error percentage as the number of components used in the gaussian mixture model varies. When the number of components is minimal (1 and 2) the error percentage initially decreases as extra dimensions are used, but then begins to increase once 8 and 24 dimensions are used for 1 and 2 components respectively. The increase in the error percentage is most dramatic when only 1 component is used as the error percentage reaches a value of approximately 9.3% at 64 dimensions as opposed to approximately 6.3% when 2 components are used. Although the error percentage increases after 16 dimensions in the 2 component mixture model, the error percentage at 64 dimensions is still smaller than the error percentage at the initial 1 dimensions by approximately 0.4%. Whereas in the 1 component mixture model, the error percentage at 64 dimensions is approximately 2.75% greater than the initial 1 dimension error percentage.

As the number of components increases to 4 and 8, the additional dimensions further decrease the error percentage up to about 30 dimensions. When more than 30 dimensions are used for mixture models containing 4 and 8 components the error percentage begins to increase. Once 16 and 32 components are used, however, the extra dimensions decrease the error percentage up until 48 dimensions are used. When more than 48 dimensions are used, once again, the error percentage begins to increase. This is likely due to overfitting the data. The best results are observed when 32 components are used as the minimum error percentage is approximately 4.9%. The minimum error percentage continues to get smaller (i.e. the classification is better) as components are added, but the decrease in error percentage is especially small between the increase of 16 to 32 components and the extra computation time required for 32 components as opposed to 16 components was not negligible.

**Code**

%Joseph Bell

%ECE271 HW5

clc;

clear;

load('TrainingSamplesDCT\_8\_new.mat');

num\_mixtures = 5;

num\_components = 8;

%8 components part

% start of loop to learn parameters for each component for FG

%Using containers for easy way to store values and access them later

learned\_mu\_FG = containers.Map; %should hold num\_mixtures (5) items

learned\_covariance\_FG = containers.Map;

learned\_weights\_FG = containers.Map;

for mix=1:num\_mixtures

mu\_c\_FG = [];

covariance\_c\_FG = []; %storing covariance as 1d array diag it later

weights = [];

disp(['Randomly initializing cheetah parameters for mixture: ' num2str(mix)]);

for i=1:num\_components

%initialized random covariance and mu

mu\_c\_FG = [mu\_c\_FG; normrnd(2,0.2,[1,64])]; %random mu

covariance\_c\_FG = [covariance\_c\_FG; abs(normrnd(3,0.1,[1,64]))]; %random covariance

weights = [weights, 1/num\_components]; %giving all initial equal weights

end

%gaussian function - taken from my last homework

%tried this function I made but dimensions were not correct so used mvnpdf

%and it worked

%fun\_cheetah = @(x, i) 1/sqrt((det(diag(covariance\_c\_FG(i,:)))\*(2\*pi)^64))\*exp(-1/2\*(x-mu\_c\_FG(i,:))\*inv(diag(covariance\_c\_FG(i,:)))\*transpose(x-mu\_c\_FG(i,:)));

%calculating weights for all 8 components for num of iterations

sum\_diff\_of\_weights = [];

%ran a bunch of samples and found most converge by about 100-200, but

%some took more so just in case made it 300 iterations. The amount of extra

%time for the extra iterations is negligible

disp(['Learning cheetah parameters for mixture: ' num2str(mix)]);

num\_iterations = 300;

for iter=1:num\_iterations

prob\_x\_given\_c\_times\_weight = [];

for i=1:num\_components

result = mvnpdf(TrainsampleDCT\_FG, mu\_c\_FG(i,:), diag(covariance\_c\_FG(i,:))); %returns 250 by 1

result\_times\_weight = result.\*weights(1,i); %multiply each prob by weight for component

prob\_x\_given\_c\_times\_weight = [prob\_x\_given\_c\_times\_weight result\_times\_weight];

end

%after this loop I have a 250x8 need to convert to a 1x8 via summation and

%normalization to ensure the sum of the weights is 1

%divide each column by sum of rows

sum\_rows = sum(prob\_x\_given\_c\_times\_weight,2);

prob\_c\_given\_x = prob\_x\_given\_c\_times\_weight./sum\_rows;

%above line gives me P(c|x) in 250\*8 form

sum\_columns = sum(prob\_c\_given\_x,1);

new\_weights = sum\_columns/250;

sum\_diff\_of\_weights = [sum\_diff\_of\_weights sum(new\_weights-weights)];

weights = new\_weights;

check\_add\_to\_one = sum(new\_weights);

%modifying mean

mu\_c\_FG = [];

for i=1:num\_components

new\_mu = sum(prob\_c\_given\_x(:,i).\*TrainsampleDCT\_FG)./sum(prob\_c\_given\_x(:,i));

mu\_c\_FG = [mu\_c\_FG; new\_mu];

end

covariance\_c\_FG = [];

%modifying covariance using new mean

for i=1:num\_components

new\_covariance = sum(prob\_c\_given\_x(:,i).\*(TrainsampleDCT\_FG - mu\_c\_FG(i,:)).^2);

new\_covariance = new\_covariance./sum(prob\_c\_given\_x(:,i));

covariance\_c\_FG = [covariance\_c\_FG; abs(new\_covariance)];

end

end

%figure(mix)

%plot(linspace(1,num\_iterations,num\_iterations),sum\_diff\_of\_weights);

learned\_mu\_FG(num2str(mix)) = mu\_c\_FG;

learned\_covariance\_FG(num2str(mix)) = covariance\_c\_FG;

learned\_weights\_FG(num2str(mix)) = weights;

% learned\_covariance\_FG stores covariance as 1x64, must diag()

% to use in pdf

end

%Using containers for easy way to store values and access them later

learned\_mu\_BG = containers.Map; %should hold num\_mixtures (5) items

learned\_covariance\_BG = containers.Map;

learned\_weights\_BG = containers.Map;

% start of loop to learn parameters for each component for BG

for mix=1:num\_mixtures

mu\_c\_BG = [];

covariance\_c\_BG = []; %storing covariance as 1d array diag it later

weights = [];

disp(['Randomly initializing grass parameters for mixture: ' num2str(mix)]);

for i=1:num\_components

%initialized random covariance and mu

mu\_c\_BG = [mu\_c\_BG; normrnd(3,0.3,[1,64])]; %random mu

covariance\_c\_BG = [covariance\_c\_BG; abs(normrnd(3,0.1,[1,64]))]; %random covariance

weights = [weights, 1/num\_components]; %giving all initial equal weights

end

%gaussian function - taken from my last homework

%tried this function I made but dimensions were not correct so used mvnpdf

%and it worked

%fun\_cheetah = @(x, i) 1/sqrt((det(diag(covariance\_c\_BG(i,:)))\*(2\*pi)^64))\*exp(-1/2\*(x-mu\_c\_BG(i,:))\*inv(diag(covariance\_c\_BG(i,:)))\*transpose(x-mu\_c\_BG(i,:)));

%calculating weights for all 8 components for num of iterations

sum\_diff\_of\_weights = [];

%ran a bunch of samples and found most converge by about 100-200, but

%some took more so just in case made it 300 iterations. The amount of extra

%time for the extra iterations is negligible

num\_iterations = 300;

disp(['Learning grass parameters for mixture: ' num2str(mix)]);

for iter=1:num\_iterations

prob\_x\_given\_c\_times\_weight = [];

for i=1:num\_components

result = mvnpdf(TrainsampleDCT\_BG, mu\_c\_BG(i,:), diag(covariance\_c\_BG(i,:))); %returns 250 by 1

result\_times\_weight = result.\*weights(1,i); %multiply each prob by weight for component

prob\_x\_given\_c\_times\_weight = [prob\_x\_given\_c\_times\_weight result\_times\_weight];

end

%after this loop I have a 250x8 need to convert to a 1x8 via summation and

%normalization to ensure the sum of the weights is 1

%divide each column by sum of rows

sum\_rows = sum(prob\_x\_given\_c\_times\_weight,2);

prob\_c\_given\_x = prob\_x\_given\_c\_times\_weight./sum\_rows;

%above line gives me P(c|x) in 250\*8 form

sum\_columns = sum(prob\_c\_given\_x,1);

new\_weights = sum\_columns/250;

sum\_diff\_of\_weights = [sum\_diff\_of\_weights sum(new\_weights-weights)];

weights = new\_weights;

check\_add\_to\_one = sum(new\_weights);

%modifying mean

mu\_c\_BG = [];

for i=1:num\_components

new\_mu = sum(prob\_c\_given\_x(:,i).\*TrainsampleDCT\_BG)./sum(prob\_c\_given\_x(:,i));

mu\_c\_BG = [mu\_c\_BG; new\_mu];

end

covariance\_c\_BG = [];

%modifying covariance using new mean

for i=1:num\_components

new\_covariance = sum(prob\_c\_given\_x(:,i).\*(TrainsampleDCT\_BG - mu\_c\_BG(i,:)).^2);

new\_covariance = new\_covariance./sum(prob\_c\_given\_x(:,i));

covariance\_c\_BG = [covariance\_c\_BG; abs(new\_covariance)];

end

end

%figure(mix)

%plot(linspace(1,num\_iterations,num\_iterations),sum\_diff\_of\_weights);

learned\_mu\_BG(num2str(mix)) = mu\_c\_BG;

learned\_covariance\_BG(num2str(mix)) = covariance\_c\_BG;

learned\_weights\_BG(num2str(mix)) = weights;

% learned\_covariance\_BG stores covariance as 1x64, must diag()

% to use in pdf

end

[row\_FG, col\_FG] = size(TrainsampleDCT\_FG);

[row\_BG, col\_BG] = size(TrainsampleDCT\_BG);

prior\_FG = row\_FG/(row\_FG+row\_BG);

prior\_BG = row\_BG/(row\_FG+row\_BG);

cheetah\_mask = imread('cheetah\_mask.bmp');

cheetah\_mask = im2double(cheetah\_mask);

cheetah\_img = imread('cheetah.bmp');

cheetah\_img = im2double(cheetah\_img); %converting to double values since training data is of type double

[cheetah\_rows, cheetah\_cols] = size(cheetah\_img);

cheetah\_img = cheetah\_img(1:8\*floor(cheetah\_rows/8),1:8\*floor(cheetah\_cols/8)); %modifying image so it can be split into 8x8 blocks

cheetah\_mask = cheetah\_mask(1:8\*floor(cheetah\_rows/8),1:8\*floor(cheetah\_cols/8));

[cheetah\_rows, cheetah\_cols] = size(cheetah\_img); %overwriting for modified dimensions

dimensions = [1 2 4 8 16 24 32 40 48 56 64];

[row\_dim, col\_dim] = size(dimensions);

counter = 0;

error\_storage = containers.Map;

for mix\_FG=1:num\_mixtures

mix\_index\_FG = num2str(mix\_FG);

cov\_FG = learned\_covariance\_FG(mix\_index\_FG);

mu\_FG = learned\_mu\_FG(mix\_index\_FG);

weights\_FG = learned\_weights\_FG(mix\_index\_FG);

for mix\_BG=1:num\_mixtures

counter = counter + 1;

disp('Beginning Classification Process');

mix\_index\_BG = num2str(mix\_BG);

cov\_BG = learned\_covariance\_BG(mix\_index\_BG);

mu\_BG = learned\_mu\_BG(mix\_index\_BG);

weights\_BG = learned\_weights\_BG(mix\_index\_BG);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

new\_image = zeros(cheetah\_rows, cheetah\_cols);

pct\_error = [];

for dim=1:col\_dim

num\_dimensions = dimensions(dim);

disp(['Beginning dimension ' num2str(num\_dimensions) ' classification']);

for i=1:cheetah\_cols-7 %shift scan pointer over a column

for j=1:cheetah\_rows-7

block = cheetah\_img(j:7+j,i:7+i); %grab 8x8 block

block\_dct = dct2(block);

zzblock\_dct = zigzag(block\_dct);

zzblock\_dct = zzblock\_dct(1,1:num\_dimensions);

cheetah\_result = 0;

grass\_result = 0;

for k=1:num\_components

cheetah\_pd = mvnpdf(zzblock\_dct,mu\_FG(k,1:num\_dimensions),diag(cov\_FG(k,1:num\_dimensions)));

cheetah\_result = cheetah\_result + cheetah\_pd\*weights\_FG(1,k);

grass\_pd = mvnpdf(zzblock\_dct,mu\_BG(k,1:num\_dimensions),diag(cov\_BG(k,1:num\_dimensions)));

grass\_result = grass\_result + grass\_pd\*weights\_BG(1,k);

end

choose\_cheetah = cheetah\_result\*prior\_FG;

choose\_grass = grass\_result\*prior\_BG;

if choose\_cheetah > choose\_grass

new\_image(j:j,i:i) = 1;

end

end

end

counter\_correct = 0;

total\_pixels = cheetah\_rows\*cheetah\_cols;

for i=1:cheetah\_rows

for j=1:cheetah\_cols

if cheetah\_mask(i,j) == new\_image(i,j)

counter\_correct = counter\_correct + 1;

end

end

end

percent\_correct = counter\_correct/total\_pixels\*100;

percent\_error = 100 - percent\_correct;

pct\_error = [pct\_error percent\_error];

%{

if counter == 1

figure()

imagesc(new\_image)

colormap(gray(255))

title(['Dimensions: ' num2str(num\_dimensions)]);

end

%}

end

disp(counter);

error\_storage(num2str(counter))= pct\_error;

end

end

for i=1:counter

prob\_error = error\_storage(num2str(i));

figure()

plot(dimensions, prob\_error, 'r--o');

xlabel('Number of Dimensions');

ylabel('Probability of Error');

title(['Classifier Combination ' num2str(i)]);

end