ECE 271A Homework Set Five (Quiz)

CHENG-YAN JUANG

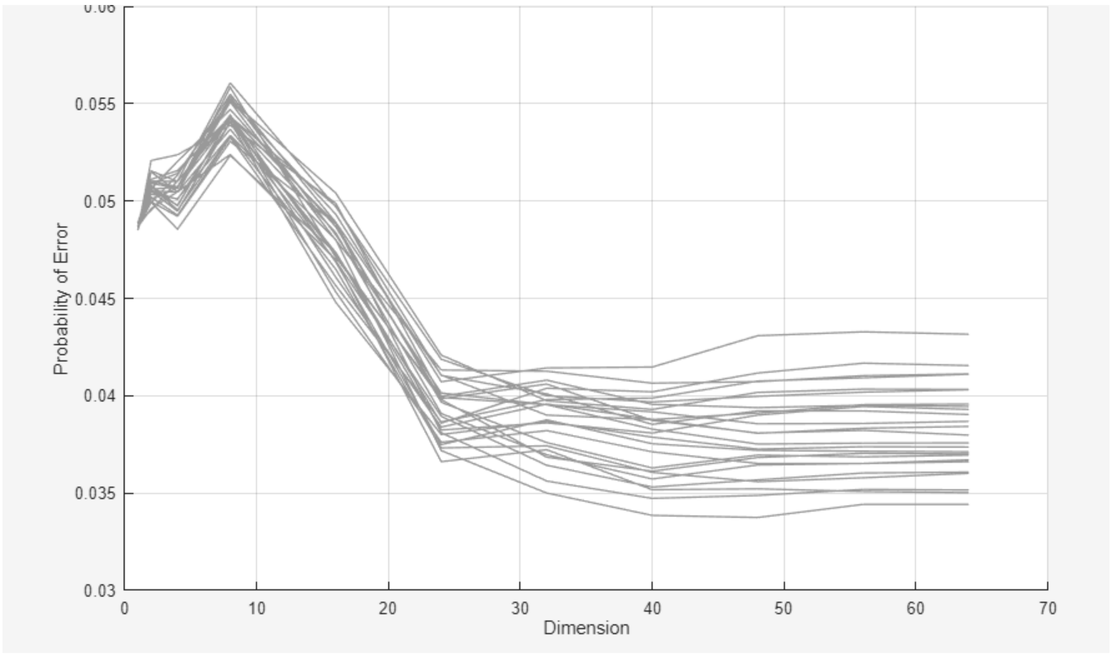
(a)

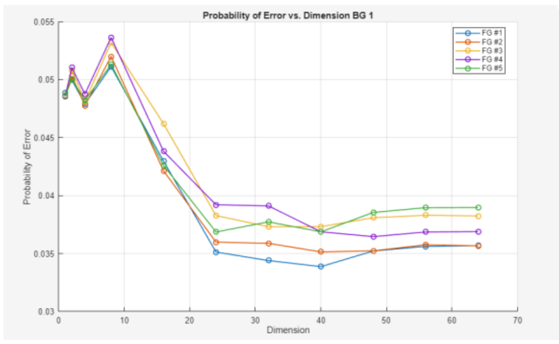
The Gaussian Mixture Model EM update equations, where n is the number of training samples, given the responsibilities h_{ij} derived in the E-step, the update rules for component j are:

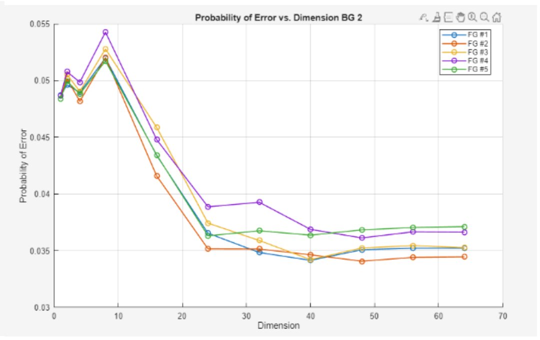
$$\mu_j^{(n+1)} = \frac{\sum_i h_{ij}\mathbf{x}_i}{\sum_i h_{ij}} \qquad \pi_j^{(n+1)} = \frac{1}{n}\sum_i h_{ij}$$

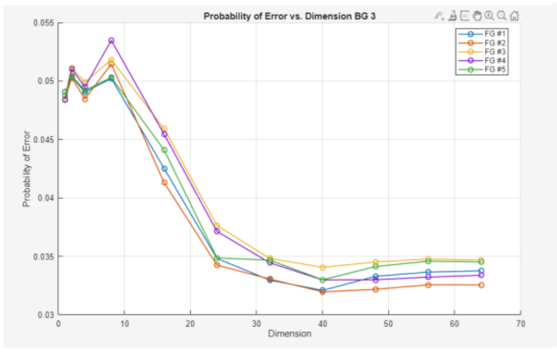
$$\sigma_j^{2(n+1)} = \frac{\sum_i h_{ij}(\mathbf{x}_i - \mu_j)^2}{\sum_i h_{ij}}$$

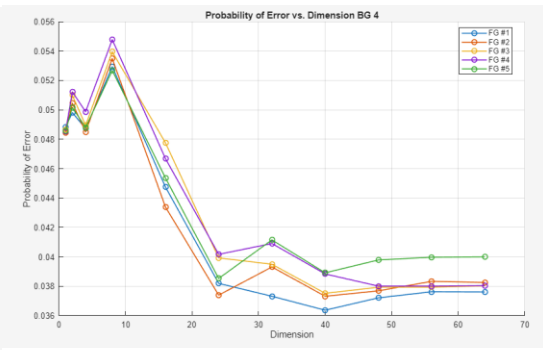
For each class (FG and BG), we trained five mixtures with C = 8 components, each starting from a different random initialization of π_c, μ_c, and Σ_c.
This gives us a total of 25 classifier pairs (5 FG models × 5 BG models). To better visualize the 25 classifiers obtained from all the possible mixture model pairs, we replace the original plotting approach with 25 curves in a single figure, to five separate figures, each corresponding to a fixed background model (BG 1 – BG 5).

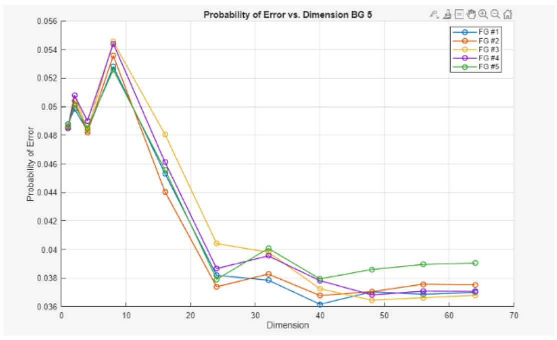






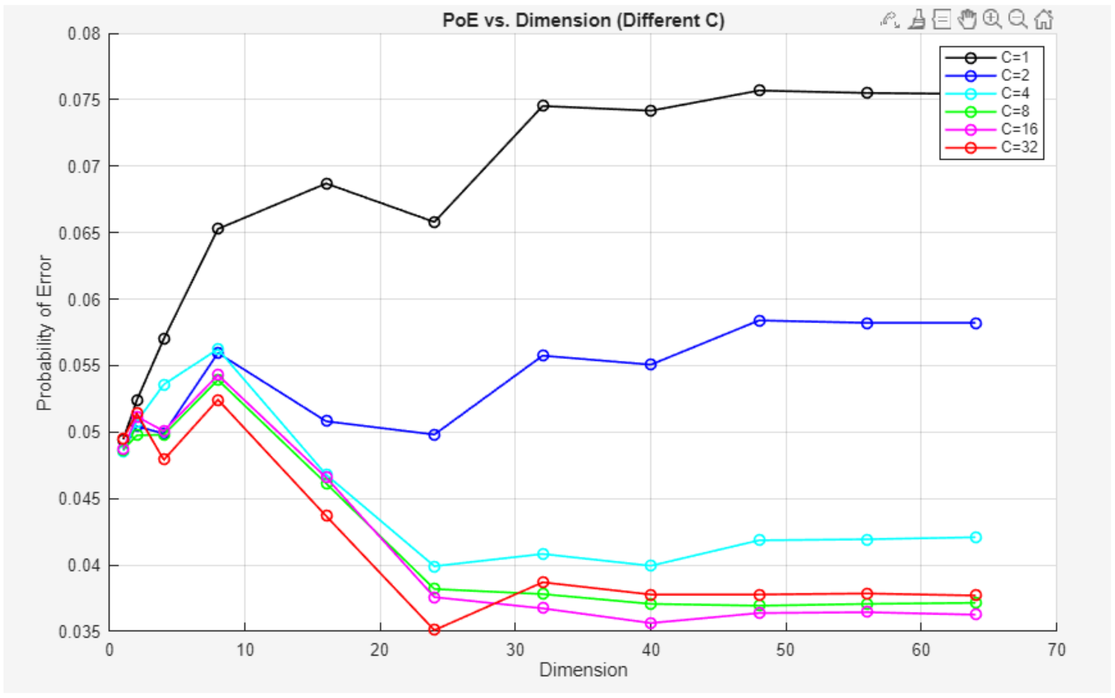






The five plots show that the probability of error depends significantly on the EM initialization when the dimensionality is low, because the merged Gaussian mixture models are more sensitive to how mixture components are initially placed. However, as the number of dimensions increases, the classifiers become more stable. In high dimensional spaces, the error rates of the 25 classifiers converge to nearly same values. This indicates that while initialization is critical in low dimensional feature spaces, its influence becomes smaller as more features are introduced, eventually having only a minor effect on the final classification performance.

(b)



This plot shows that the error decreases when the number of dimensions increases. After the dimensions increase to a certain point, the improvement becomes subtle, and the curves begin to flatten out. Comparing the curves across different values of C, we observed that small C models cannot fully capture the variability in each class, therefore the error is slightly higher. On the other hand, large C models increase the number of mixture components, it makes the model more complex but the amount of training data for each component becomes small, which leads to noisy covariance estimates and overfitting. Moderate C models $C = 8$ or $C = 16$ obtains the lowest probability of error. These models balance flexibility and stability; they fit the data distribution well without overfitting.

Thus, the results indicate that increasing C improves performance only up to a certain point, after which additional mixture components do not provide further benefit. A moderate complexity provides the best trade-off between flexibility and generalization.

Table of Contents

```
clear; clc;
```

0) Load data & images

```
load('TrainingSamplesDCT_8_new.mat')
test_img = im2double(imread('cheetah.bmp'));
ground_truth = imread('cheetah_mask.bmp');
ground_truth = im2double(ground_truth);
ground_truth = ground_truth > 0.5;

N_FG = size(TrainsampleDCT_FG, 1);
N_BG = size(TrainsampleDCT_BG, 1);
Py_cheetah = N_FG / (N_FG + N_BG);
Py_grass = 1 - Py_cheetah;
```

1) Zig-zag

```
zig_zag_array = load('Zig-Zag Pattern.txt');
zig_zag_array = zig_zag_array + 1;
order_coef = zeros(1,64);
for r = 1:8
    for c = 1:8
        k = zig_zag_array(r,c);
        order_coef(k) = sub2ind([8,8], r, c);
    end
end
```

2) Sliding-window

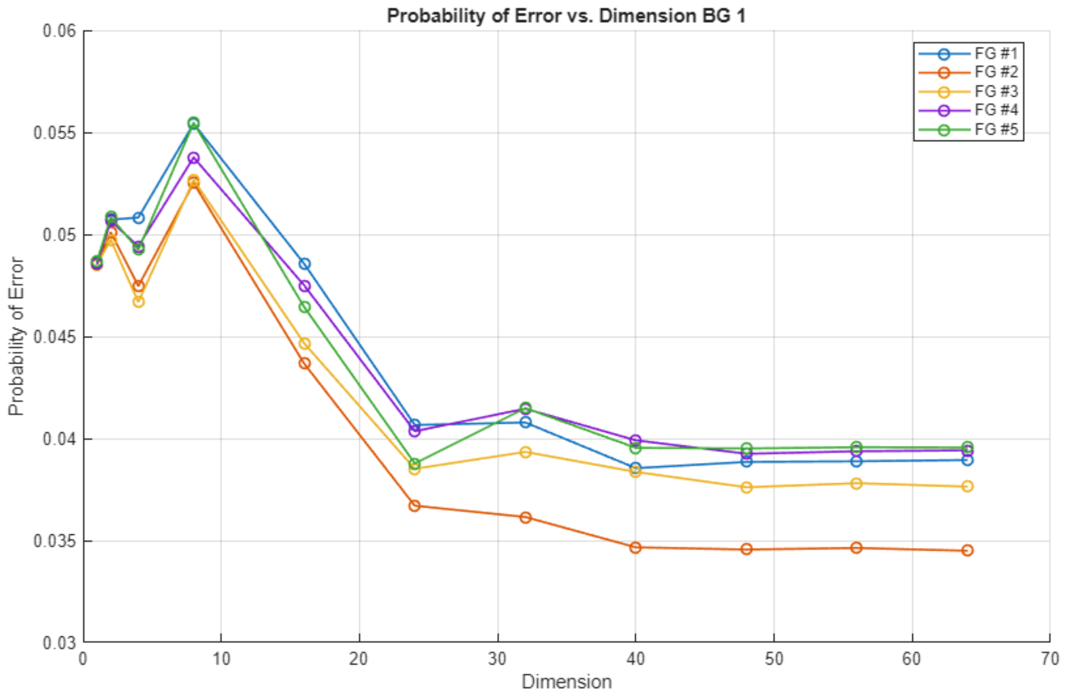
```
[H,W] = size(test_img);

num_blocks = (H-7) * (W-7);
Test_Features = zeros(num_blocks, 64);

Test_Labels = zeros(num_blocks, 1);

row_count = 0;

for i = 1:(H-7)
```

```
    for j = 1:(W-7)
        row_count = row_count + 1;
        block = test_img(i:i+7, j:j+7);
        DCT = dct2(block);

        vec = DCT(order_coef);
        Test_Features(row_count, :) = vec;
        Test_Labels(row_count, 1) = ground_truth(i+4, j+4);
    end
end
```

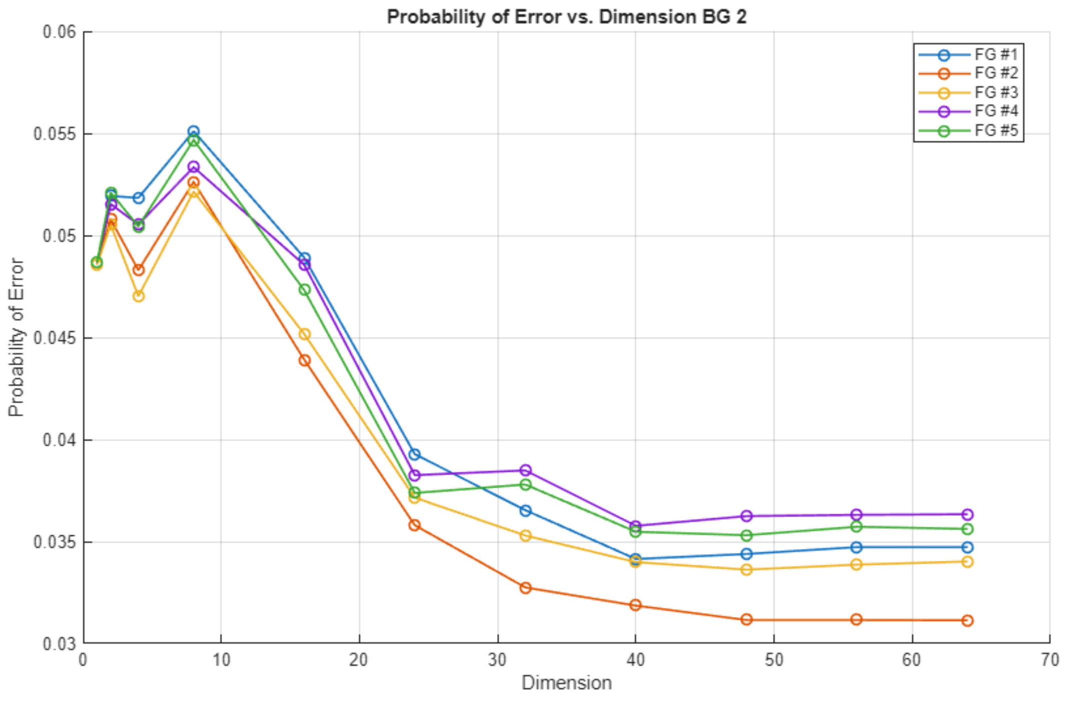
(a) C=8, FG/BG learn 5

```
C = 8;
runs = 5;
dims = [1, 2, 4, 8, 16, 24, 32, 40, 48, 56, 64];

% 1)
pi_BG = cell(runs,1);
mu_BG = cell(runs,1);
var_BG = cell(runs,1);
pi_FG = cell(runs,1);
mu_FG = cell(runs,1);
var_FG = cell(runs,1);

% 2) FG
for i = 1:runs
    [pi_FG{i}, mu_FG{i}, var_FG{i}] = EM(TrainsampleDCT_FG, C);
end

% 3) BG
for i = 1:runs
    [pi_BG{i}, mu_BG{i}, var_BG{i}] = EM(TrainsampleDCT_BG, C);
end

% 4) 25
for j = 1:runs
    figure;
    hold on;
    grid on;
    title(sprintf('Probability of Error vs. Dimension BG %d', j));
    xlabel('Dimension');
    ylabel('Probability of Error');

    for i = 1:runs
            current_errors = zeros(1, numel(dims));

            for k = 1:numel(dims)
                d = dims(k);
                X = Test_Features(:, 1:d);

                prob_FG = zeros(size(X, 1), 1);
                for c = 1:C
```

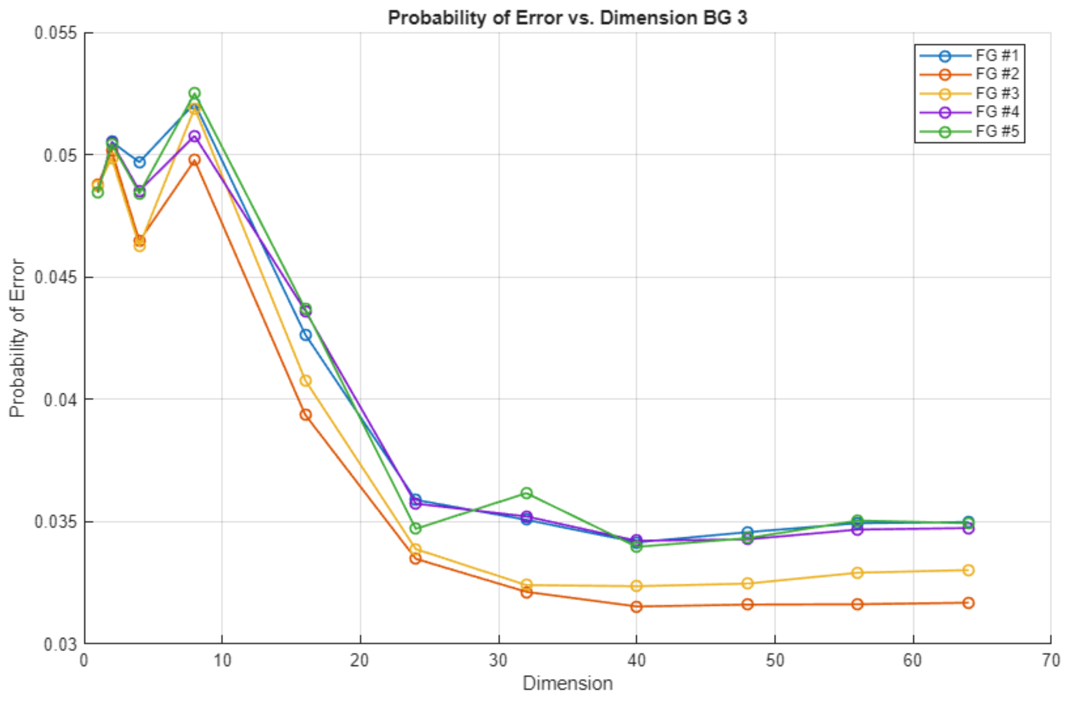
```
                    mu = mu_FG{i}(c, 1:d);
                    sigma = var_FG{i}(c, 1:d);
                    weight = pi_FG{i}(c);

                    prob_FG = prob_FG + weight * multv_npdf(X, mu, sigma);
                end

                total_FG = prob_FG * Py_cheetah;

                prob_BG = zeros(size(X, 1), 1);
                for c = 1:C
                    mu = mu_BG{j}(c, 1:d);
                    sigma = var_BG{j}(c, 1:d);
                    weight = pi_BG{j}(c);

                    prob_BG = prob_BG + weight * multv_npdf(X, mu, sigma);
                end

                total_BG = prob_BG * Py_grass;

                pred = (total_FG > total_BG);
                current_errors(k) = mean(pred ~= Test_Labels);
            end

        plot(dims, current_errors, '-o', 'LineWidth', 1.2, 'DisplayName',
sprintf('FG #%d', i));
    end

    legend('show', 'Location', 'best');
end
```

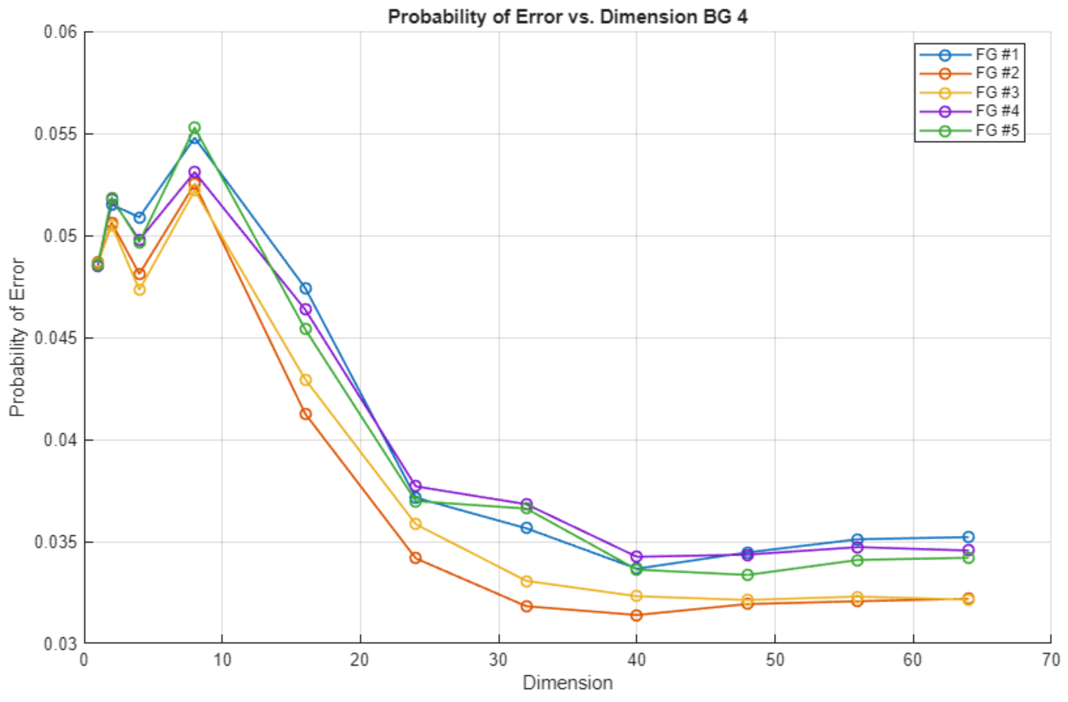


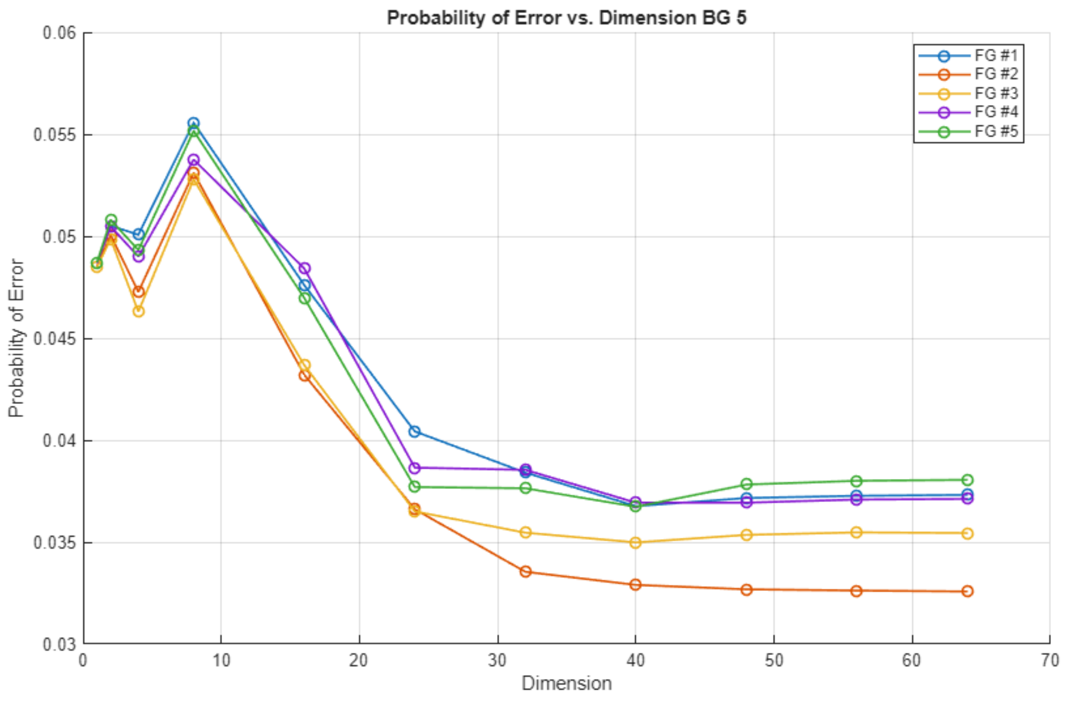
Probability of Error vs. Dimension BG 1
Probability of Error
Dimension
FG #1
FG #2
FG #3
FG #4
FG #5
0.06
0.055
0.05
0.045
0.04
0.035
0.03
0
10
20
30
40
50
60
70



Probability of Error vs. Dimension BG 2
Probability of Error
Dimension
FG #1
FG #2
FG #3
FG #4
FG #5
0.06
0.055
0.05
0.045
0.04
0.035
0.03
0
10
20
30
40
50
60
70

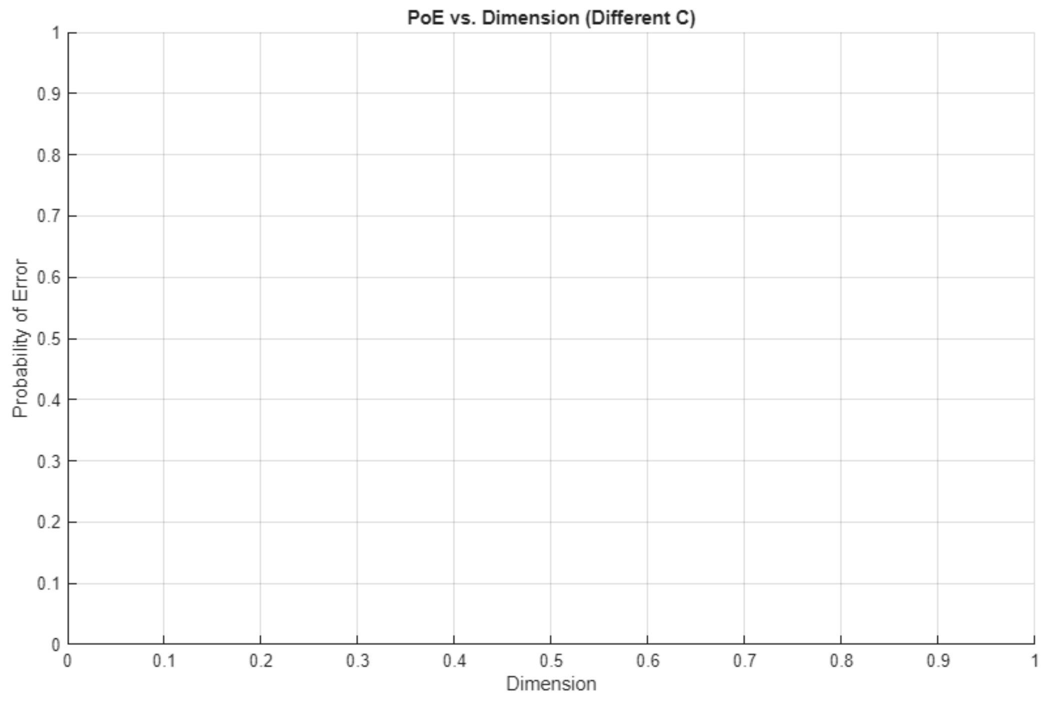

Probability of Error vs. Dimension BG 3
FG #1
FG #2
FG #3
FG #4
FG #5
Probability of Error
Dimension
0.055
0.05
0.045
0.04
0.035
0.03
0
10
20
30
40
50
60
70


Probability of Error vs. Dimension BG 4
FG #1
FG #2
FG #3
FG #4
FG #5
Probability of Error
Dimension
0.06
0.055
0.05
0.045
0.04
0.035
0.03
0
10
20
30
40
50
60
70



(b) C ∈ {1,2,4,8,16,32}

```
C_list = [1, 2, 4, 8, 16, 32];
dims = [1, 2, 4, 8, 16, 24, 32, 40, 48, 56, 64];

figure;
hold on;
grid on;
title('PoE vs. Dimension (Different C)');
xlabel('Dimension');
ylabel('Probability of Error');

colors = {'k', 'b', 'c', 'g', 'm', 'r'};

for i = 1:length(C_list)
    C = C_list(i);

    [pi_fg, mu_fg, sig_fg] = EM(TrainsampleDCT_FG, C);
    [pi_bg, mu_bg, sig_bg] = EM(TrainsampleDCT_BG, C);

    current_errors = zeros(1, length(dims));

    for k = 1:length(dims)
        d = dims(k);

        X = Test_Features(:, 1:d);

        prob_FG = zeros(size(X, 1), 1);
        for c = 1:C

            pdf_val = multv_npdf(X, mu_fg(c, 1:d), sig_fg(c, 1:d));
            prob_FG = prob_FG + pi_fg(c) * pdf_val;
        end

        total_score_FG = prob_FG * Py_cheetah;

        prob_BG = zeros(size(X, 1), 1);
        for c = 1:C
            pdf_val = multv_npdf(X, mu_bg(c, 1:d), sig_bg(c, 1:d));
            prob_BG = prob_BG + pi_bg(c) * pdf_val;
        end

        total_score_BG = prob_BG * Py_grass;

        prediction = total_score_FG > total_score_BG;

        current_errors(k) = sum(prediction ~= Test_Labels) /
length(Test_Labels);
    end

    plot(dims, current_errors, '-o', 'Color', colors{i}, 'LineWidth', 1.2)

end

legend('C=1', 'C=2', 'C=4', 'C=8', 'C=16', 'C=32');
```



EM

```
function [pi_c, mu_c, sigma_c] = EM(data, C)
    [N, ~] = size(data);

    % 0) Initialization
    rand_idx = randperm(N, C);
    mu_c = data(rand_idx, :);
    sigma_c = repmat(var(data) + 0.001, C, 1);
    pi_c = ones(1, C) / C;

    for iter = 1:50
        % 1) E-Step
        log_prob = zeros(N, C);
        for c = 1:C

             diff = data - mu_c(c, :);
             term1 = sum(log(sigma_c(c, :)));
             term2 = sum((diff.^2) ./ sigma_c(c, :), 2);
             log_prob(:, c) = log(pi_c(c)) - 0.5 * (term1 + term2);
        end

        max_log = max(log_prob, [], 2);
        prob_norm = exp(log_prob - max_log);
        responsibilities = prob_norm ./ sum(prob_norm, 2);

        % 2) M-Step
        Nk = sum(responsibilities, 1);
        for c = 1:C
            pi_c(c) = Nk(c) / N;
            mu_c(c, :) = (responsibilities(:, c)' * data) / Nk(c);
            diff = data - mu_c(c, :);
            sigma_c(c, :) = (responsibilities(:, c)' * (diff.^2)) / Nk(c);
            sigma_c(c, sigma_c(c,:) < 1e-4) = 1e-4;
        end
    end
end

% Multivariate Normal PDF funtion
function y = multv_npdf(X, mu, var_vec)

    [N, D] = size(X);

    det_sigma = prod(var_vec);
    const = 1 / sqrt((2*pi)^D * det_sigma);

    y = zeros(N, 1);

    for i = 1:N

        x_row = X(i, :);
        diff = x_row - mu;
        exponent = -0.5 * sum((diff.^2) ./ var_vec);

        y(i) = const * exp(exponent);
    end
end
```

Published with MATLAB® R2025b