EECS 545 HW3 Chien-Wei Lin

1.

(1) Since arriving safely is the children of Snowy, arriving safely is not independs of snowy weather and no other property observed

(2)

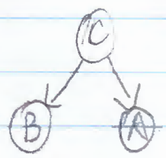
Since safe driver is not observed, "properly maintained is not independent of repair bill"

(3)

Yes., as "Arrive Safely" is not observe. "Properly Maintained Car" is conditionally independent of snow weather given a safe driver

(4)

Yes, "repair bill" is conditionally independent of arriving safely, given a car accident

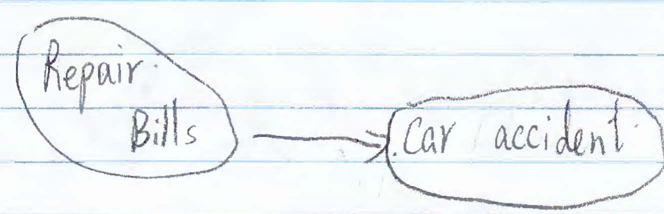


$$P(A,B|C) = \frac{P(A,B,C)}{P(C)} = \frac{P(C)P(B|C)P(A|C)}{P(C)} = P(B|C)P(A|C)$$

(5)

Yes, it's possible

$\Rightarrow$



2.

(1)

Let's have an one dimension example 1, 2, 3, 4, 5, 6, 7, 8, 9, 10000

$\Rightarrow$ mean is 1004.5, but medoids is 6

In this case, 6 can provide more information about the set rather then 1004.5

(2)

k-means must use distances that consistent with mean however, k-medoid is more flexible. we can use any similarity measure. K means minimize the Euclidean distances, but not minimize arbitrary other distances. Therefore, k means is not good at optimizing pearson correlation. On the other hand, k medoids can.

3.

(1)

$$q_c^{(i)} = P(y^{(i)}=c \mid x^{(i)}) = \frac{P(x^{(i)} \mid y^{(i)}=c)\,P(y^{(i)}=c)}{P(x^{(i)})} = \frac{\pi_c \prod_{j=1}^{m} \theta_{jc}^{x_j^{(i)}} (1-\theta_{jc})^{(1-x_j^{(i)})}}{\prod_{j=1}^{m} P(x_j^{(i)})}$$

(2)

$$Q(\lambda, \theta) = \sum_{i=1}^{l} \log P(x^{(i)}, y^{(i)}) + \lambda \sum_{i=l+1}^{l+u} \log P(x^{(i)})$$

$$= \sum_{i=1}^{l} \left[\sum_{c=1}^{k} \left(1_{\{y^{(i)}=c\}}\right) \log\left(\pi_c \prod_{j=1}^{m} \theta_{jc}^{x_j^{(i)}} (1-\theta_{jc})^{1-x_j^{(i)}}\right) \right] + \lambda \sum_{i=l+1}^{l+u} \log \prod_{c=1}^{k} \frac{P(y^{(i)}=c, x^{(i)})}{P(y^{(i)}=c \mid x^{(i)})}$$

$$= \sum_{i=1}^{l} \left[\sum_{c=1}^{k} \left(1_{\{y^{(i)}=c\}}\right) \log \pi_c \prod_{j=1}^{m} \theta_{jc}^{x_j^{(i)}} (1-\theta_{jc})^{1-x_j^{(i)}} \right] + \lambda \sum_{i=l+1}^{l+u} \sum_{c=1}^{k} \log \frac{P(y^{(i)}=c, x^{(i)})}{q_c^{(i)}}$$

(3)

$$\pi_c = \frac{\sum_{i=1}^{l} 1_{\{y^{(i)}==c\}}}{l} + \lambda \frac{\sum_{i=l+1}^{l+u} q_c^{(i)}}{u}$$

$$\theta_{jc} = \frac{\sum_{i=1}^{l} 1_{\{y^{(i)}=c\}} x_j^{(i)}}{\sum_{i=1}^{l} 1_{\{y^{(i)}=c\}} \sum_{j=1}^{m} x_j^{(i)}} + \lambda \frac{\sum_{i=l+1}^{l+u} q_c^{(i)} x_j^{(i)}}{\sum_{i=l+1}^{l+u} \sum_{j=1}^{m} q_c^{(i)} x_j^{(i)}}$$

5.

Let $\Sigma = Q \Lambda Q^T$ as the eigendecomposition of Σ

$\Rightarrow$ the objective function $= \arg\max_{|c|=1} c^T \Sigma c = \arg\max_{|c|=1} c^T Q \Lambda Q^T c = \arg\max_{|c|=1} \| \Lambda^{\frac{1}{2}} Q^T c \|_2^2$

$= \arg\max_{|c|=1} \sum_{i=1}^{n} \lambda_i (q_i^T c)^2$

since $\sum_{i=1}^{n} \lambda_i (q_i^T c)^2 \leq \lambda_1 \sum_{i=1}^{n} (q_i^T c)^2 \leq \lambda_1$, subject to $\|c\| = 1$

In order to maximize $c^T \Sigma c$, c should be first eigenvector w.r.t. the largest eigenvalue.

4.

(1)

```
large = 'mandrill-large.tiff';
filename = large;
A = double(imread(filename));
imwrite(uint8(round(A)), 'p4_1.tiff');
```

(2)

```
k = 16; nIters = 40;
filename = small;
[m, kgroup] = myKmeans(filename, k, nIters);
function [ m, kgroup] = myKmeans(filename, k, nIters)
    %%
    color_scale = 256;
    im = double(imread(filename));
    [rows, cols, dim] = size(im);
    m = floor(color_scale * rand(dim, k));
    kgroup = zeros(rows, cols);
    for h = 1:nIters
        kgroup = findClosestCenterOf(kgroup, im, m);
        m = findMeans(kgroup, im, m, k);
        format long
        disp(['percentage: ', num2str(round(h / nIters * 100)) , '%']);
    end
    %%
    function m = findMeans(kgroup, im, m, k)
```

```
        for l = 1:k
            [row, col] = find(kgroup == l);
            if ~isempty(row)
                count = 0;
                for i = 1:length(row)
                    count = count + 1;
                    m(:, l) = m(:, l) + squeeze(im(row(i), col(i),:));
                end
                m(:, l) = m(:, l)/ count;
            end
        end
    end
end

function kgroup = findClosestCenterOf(kgroup, im, m)
    k = size(m, 2);
    for i = 1:size(im, 1)
        for j = 1:size(im, 2)
            tmp = zeros(1, k);
            for l = 1:k
                tmp(l) = norm(squeeze(im(i, j, :)) - m(:, l), 2);
            end
            [~,I] = min(tmp);
            kgroup(i, j) = I;
        end
    end
end
```

(3)

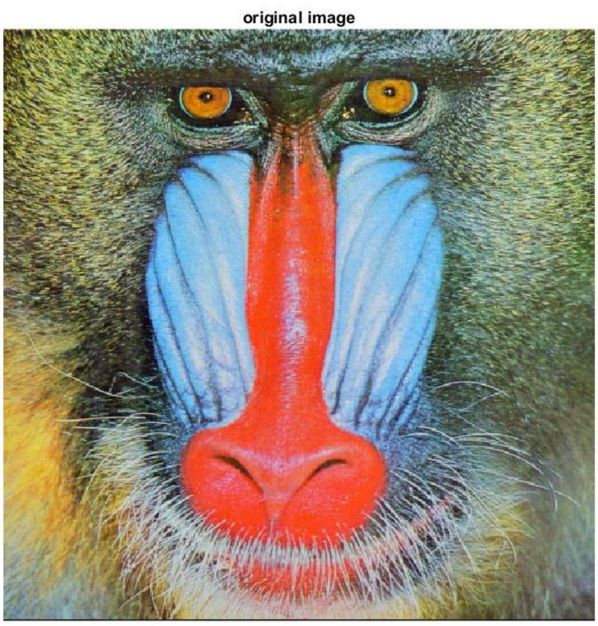

```
filename = large;
A = double(imread(filename));
image = compress(m, A);
figure,
subplot(1,2,1),
imshow(uint8(A));
title('original image');
subplot(1,2,2),
imshow(image);
title('k means compressed image');


function image = compress(m, im)
```

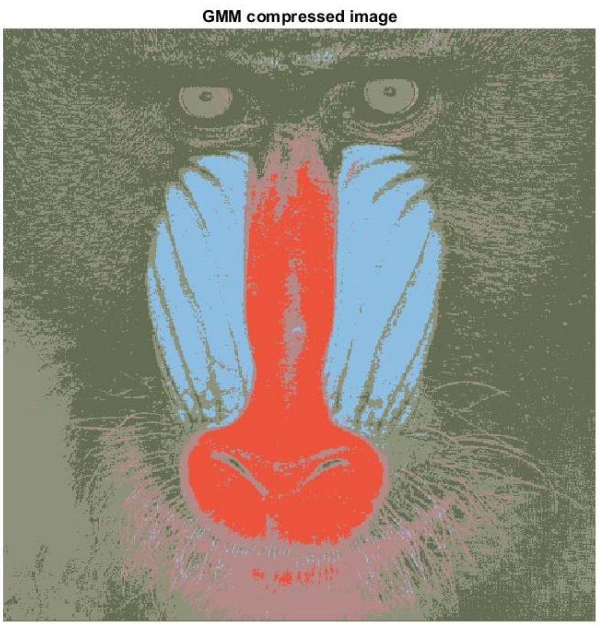
```
    [rows, cols, dim] = size(im);
    image = zeros(rows, cols, dim);
    kgroup = zeros(rows, cols);
    kgroup = findClosestCenterOf(kgroup, im, m);
    for i = 1:rows
        for j = 1:cols
            idx = kgroup(i, j);
            image(i, j, :) = reshape(m(:, idx), [1, 1, dim]);
        end
    end
    image = uint8(image);
end
```

(4)

Each pixel in the original image has the value from 0 to 256 on RGB. Therefore, it cost 24 bits per pixel. On the other hand, the kmeans only need to save the value of cluster from 0 to 16, which only costs 4 bits. As a result, if we assume that the means are the same for all the images and ignore them on calculating compressed factor. We use only $\frac{4}{24} = 16.6\%$ of the original image after compression.

(4)





We use 3 bits to store the MAP estimate for each pixel. In GMM, We use $\frac{3}{24} = 12.5\%$ of the original image after compression.

	μ		
K	R	G	B
1	181.577695440685	137.930878010427	135.850026990273
2	141.791300841145	146.007393447149	122.277218588609
3	237.362424634810	84.1546764501801	61.9142384005461
4	141.583117117354	190.533438387129	226.278000572344
5	100.809940463799	110.026302800986	84.8502275241879

Σ_1		
110.94	948.9	32.6
948.9	1156.5	403.1
32.5	403.2	594.4

Σ_2

815.91	644.31	251.73
644.31	658.33	446.36
251.73	446.36	891.47

Σ_3		
195.65	-154.88	321.65
-154.88	334.52	584.75
321.65	584.75	1245.6

Σ_4		
412.39	456.29	278.17
456.29	604.71	417.85
278.17	417.85	391.82

Σ_5		
920.57	13.93	-463.68
13.93	801.55	1047.13
-463.68	1047.13	1693.85

```
k = 5;
nIters = 100;
filename = small;
[m, sigma, prior] = gmm(filename, k, nIters);
filename = large;
A = double(imread(filename));
[rows, cols, dim] = size(A);

W = expectation(reshape(A, [rows * cols, dim]), m, k, sigma, prior);

image = zeros(rows, cols, dim);
for i = 1:rows
    for j = 1:cols
        idx = (i - 1) * cols + j;
        [~, I] = max(W(idx, :));
        image(j, i, :) = reshape(m(I, :), [1, 1, dim]);
    end
end
image = uint8(image);
figure,
subplot(1,2,1),
imshow(uint8(A));
title('original image');
subplot(1,2,2),
imshow(image);
title('GMM compressed image');

function [m, sigma, prior] = gmm( filename, k, nIters )
%GMM Summary of this function goes here
%   Detailed explanation goes here

    im = double(imread(filename));
    [rows, cols, dim] = size(im);
    X = reshape(im, [rows * cols, dim]);

    %% initialization
    indeces = randperm(rows * cols);
    m = X(indeces(1:k), :);
    sigma = zeros(dim, dim, k);
    % Use the overal covariance of the dataset as the initial variance for each
cluster.
    for i = 1:k
        sigma(:, :, i) = cov(X);
```

```
    end
    % Assign equal prior probabilities to each cluster.
    prior = ones(1, k) * (1 / k);
    %% EM
    for h = 1:nIters
        prevM = m;
        W = expectation( X, m, k, sigma, prior);
        [prior, m, sigma ] = maximization(m, W, X, k, prior, sigma);
        if m == prevM
            break
        end
    end
end
function W = expectation( X, m, k, sigma, prior)
    [row, n] = size(X);
    pdf = zeros(row, k);
    for i = 1 : k
        Sigma = sigma(:, :, i);
        meanDiff = bsxfun(@minus, X, m(i, :));
        pdf(:, i) = 1 / sqrt((2*pi)^n * det(Sigma)) * exp(-1/2 * sum((meanDiff /
Sigma .* meanDiff), 2));
    end
    pdf_w = bsxfun(@times, pdf, prior);
    W = bsxfun(@rdivide, pdf_w, sum(pdf_w, 2));
end

function [prior, m, sigma ] = maximization(m, W, X, k, prior, sigma)
    [row, n] = size(X);
    for i = 1 : k
        prior(i) = mean(W(:, i), 1);

        % Divide by the sum of the weights.
        m(i, :) = (W(:, i)' * X) ./ sum(W(:, i), 1);
        sigma_k = zeros(n, n);
        meanDiff = bsxfun(@minus, X, m(i, :));
        for j = 1 : row
            sigma_k = sigma_k + (W(j, i) .* (meanDiff(j, :)' * meanDiff(j, :)));
        end
        sigma(:, :, i) = sigma_k ./ sum(W(:, i));
    end
end
```