Distributed Multimedia Systems Assignment

CHEN SHANG (G2203629G)

Feb 2023

1.

(a)

$$\left(\begin{array}{cc:cc:cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 2 \\ \hdashline 1 & 1 & 2 & 2 & 1 & 2 \\ 1 & 1 & 2 & 4 & 4 & 4 \\ \hdashline 1 & 2 & 2 & 3 & 4 & 4 \\ 1 & 2 & 3 & 3 & 4 & 4 \end{array}\right)$$

$$\mathbf{x_1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \ \mathbf{x_2} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \ \mathbf{x_3} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \end{bmatrix}, \ \mathbf{x_4} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \ \mathbf{x_5} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 4 \end{bmatrix}$$

$$\mathbf{x_6} = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 4 \end{bmatrix}, \ \mathbf{x_7} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \end{bmatrix}, \ \mathbf{x_8} = \begin{bmatrix} 2 \\ 3 \\ 3 \\ 3 \end{bmatrix}, \ \mathbf{x_9} = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}$$

$$\overline{\mathbf{x}} = \frac{1}{9}\sum_{i=1}^{9}\mathbf{x_i} = \begin{bmatrix} 2.75 \\ 3.5 \\ 4.25 \\ 5.25 \end{bmatrix}$$

$$\mathbf{C} = \frac{1}{8}\sum_{i=1}^{9}(\mathbf{x_i} - \overline{\mathbf{x}})(\mathbf{x_i} - \overline{\mathbf{x}})^T$$

Use the following codes to calculate covariance matrix:

```
clear;clc;
x1 = [0;0;0;0];
x2 = [0;0;1;1];
x3 = [0;0;1;2];
x4 = [1;1;1;1];
x5 = [2;2;2;4];
x6 = [1;2;4;4];
x7 = [1;2;1;2];
x8 = [2;3;3;3];
x9 = [4;4;4;4];
xbar = zeros(4,1);
for i=1:1:9
    xbar = xbar + eval('x' + string(i));
end
xbar = xbar ./ 9;
csum = zeros(4,4);
for i=1:1:9
    temp = eval('x' + string(i)) - xbar;
    csum = csum + temp * temp';
end
c = csum ./ 8;
disp(c);
```

Therefore, $\mathbf{C} = \begin{bmatrix} 1.6944 & 1.7361 & 1.4028 & 1.4167 \\ 1.7361 & 2.0278 & 1.6944 & 1.6667 \\ 1.4028 & 1.6944 & 2.1111 & 1.9167 \\ 1.4167 & 1.6667 & 1.9167 & 2.2500 \end{bmatrix}$

(b)

$\det(\mathbf{C} - \lambda\mathbf{I}) = \lambda^4 - \frac{97\lambda^3}{12} + \frac{2335\lambda^2}{288} - \frac{543\lambda}{256} + \frac{641}{4608} = 0$

$\lambda_1 = 0.1001,\ \lambda_2 = 0.2634,\ \lambda_3 = 0.7577,\ \lambda_4 = 6.9621$

$v_1 = \begin{bmatrix} 0.6585 \\ -0.7251 \\ 0.1958 \\ -0.0464 \end{bmatrix},\ v_2 = \begin{bmatrix} -0.1715 \\ 0.0837 \\ 0.7319 \\ -0.6541 \end{bmatrix},\ v_3 = \begin{bmatrix} 0.5811 \\ 0.4541 \\ -0.4013 \\ -0.5433 \end{bmatrix},\ v_4 = \begin{bmatrix} 0.4464 \\ 0.5109 \\ 0.5147 \\ 0.5242 \end{bmatrix}$

(c)

$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} => b = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$

Because λ_1 is smaller than other eigen values, we choose to use 3 principal components.

$\hat{b} = \sum_{i=2}^{4}(b^T v_i)v_i = \begin{bmatrix} 1.0439 \\ 0.9517 \\ 0.0130 \\ -0.0031 \end{bmatrix} => \begin{bmatrix} 1.0439 & 0.9517 \\ 0.0130 & -0.0031 \end{bmatrix}$

2.

(a)

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 5 & 9 & 6 \\ 9 & 17 & 10 \\ 6 & 10 & 8 \end{bmatrix}$$

$$\det(\mathbf{A}\mathbf{A}^T - \lambda\mathbf{I}) = -\lambda^3 + 30\lambda^2 - 44\lambda = 0$$

$$\lambda_1 = \sqrt{181} + 15,\ \lambda_2 = -\sqrt{181} + 15,\ \lambda_3 = 0$$

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 6 & 10 \\ 10 & 24 \end{bmatrix}$$

$$\det(\mathbf{A}^T\mathbf{A} - \lambda\mathbf{I}) = \lambda^2 - 30\lambda + 44 = 0$$

$$\lambda_1 = \sqrt{181} + 15,\ \lambda_2 = -\sqrt{181} + 15$$

Sorting the above positive eigen values in the decreasing order and squaring them, we get singular value $5.3342,\ 1.2435$

$$\mathbf{\Sigma} = \begin{bmatrix} 5.3342 & 0 \\ 0 & 1.2435 \\ 0 & 0 \end{bmatrix}$$

(b)

```
clear;clc;
img = imread("1.jpg");
rank = [10, 20, 50, 100, 200, 500];
for i=1:1:6
    subplot(2, 3, i);
```

```
    img_re=svd_compression(img, rank(1,i));
    imshow(img_re);
    title('rank = ' + string(rank(1,i)));
end

function image = svd_compression(img, rank)
    [height, width, channel] = size(img);
    channel_r = double(img(:, :, 1));
    channel_g = double(img(:, :, 2));
    channel_b = double(img(:, :, 3));
    [ur, sr, vr] = svd(channel_r);
    [ug, sg, vg] = svd(channel_g);
    [ub, sb, vb] = svd(channel_b);
    channel_r_re = zeros(height, width);
    channel_g_re = zeros(height, width);
    channel_b_re = zeros(height, width);
    for i=1:1:rank
        channel_r_re = channel_r_re + sr(i, i) * ur(:, i) * vr(:, i
    )';
        channel_g_re = channel_g_re + sg(i, i) * ug(:, i) * vg(:, i
    )';
        channel_b_re = channel_b_re + sb(i, i) * ub(:, i) * vb(:, i
    )';
    end
    img_re = zeros(height, width, channel);
    img_re(:, :, 1) = channel_r_re;
    img_re(:, :, 2) = channel_g_re;
    img_re(:, :, 3) = channel_b_re;
    image = uint8(img_re);
end
```

Output of the above codes is:

rank = 10

rank = 20

rank = 50

rank = 100

rank = 200

rank = 500

3.

I would like to talk about diffusion models, a deep learning concept, which have practical applications in media generation, specifically in image generation.

Generation models, such as the once-popular GAN and the more recent diffusion models, are designed to learn the underlying distribution of real data. Diffusion models achieve this by employing a technique of introducing and removing noise in the learning process.

Diffusion models have numerous practical applications for image generation, including DALLE and Stable Diffusion. These models have the potential to replace the tedious and repetitive work of painters in the gaming industry. By generating high-quality textures for 3D models, diffusion models make games more immersive and realistic.

Diffusion models has the following advantages:

- Diffusion models can benefit from other large multimodal models such as CLIP, offering better text guided control compared with GANs.
- Diffusion models produce more diverse output compared with previous generation models. GANs, for example, commonly suffer from model collapse, which is a problem that diffusion models can mitigate. Unlike GANs, which learn a direct mapping from input to output, diffusion models learn the gradient of the image distribution rather than the distribution of pixels. This approach helps to reduce the occurrence of model collapse.

However, there are also some disadvantages associated with diffusion models:

- Diffusion models typically require a large amount of data to train effectively. For example, the most popular diffusion model for image generation today, Stable Diffusion, utilizes the LAION-5B dataset which contains 5 billion images and has a size of 80 terabytes.
- Diffusion models usually have slower inference speeds compared with networks of similar size. This is due to their unique image generation process, which involves denoising by solving diffusion stochastic differential equations.

(287 words)