AMATH 482 Homework 5

Yiping Li

March 17, 2021

Abstract

In this study, we are asked to separate the video stream to both the foreground video and a background, using the Dynamic Mode Decomposition method (DMD).

1 Introduction and Overview

1.1 Problem Setting

The first video contains several running Formula 1 cars and one waving flag. The background is almost the same at every frame except that the camera was slightly shaking. The second video contains a person skiing down from a mountain. Again, our task is to separate the foreground and the background video. In other words, we need to extract the motion out of the background.

1.2 Data Format

All video inputs are given in `MP4` file. The first video is about 6 seconds long and has the resolution of 1872 * 1112. Its frame per second (fps) is around 60. The second video is about 7 seconds long and has the resolution of 1466 * 1604. Its fps is also around 60. Notice that for each frame we need to store 2 million numbers and each video has over 300 frames, which give us around 600 million numbers. We tried to process those original videos, but it turns that the memory of our device is not enough to hold such amount of data. Thus, we used the videos with lower resolution (960 * 540).

2 Theoretical Background

The most significant trait of DMD is that it linearizes non-linear things. Suppose $U(x_i, t_j)$ describes the state of the i^{th} element at time t_j, if we use rows to code elements and columns to code time, we would get a matrix $\mathbf{X}$ such that,

$$\mathbf{X} = \begin{bmatrix} U(x_1,t_1) & U(x_1,t_2) & \dots & U(x_1,t_M) \\ U(x_2,t_1) & U(x_2,t_2) & \dots & U(x_2,t_M) \\ \vdots & \dots & \ddots & \vdots \\ U(x_N,t_1) & U(x_N,t_2) & \dots & U(x_N,t_M) \end{bmatrix}$$

Now, let

$$\mathbf{X_{jk}} = \begin{bmatrix} U(\mathbf{x},t_j) & U(\mathbf{x},t_{j+1}) & \dots & U(x_1,t_k) \end{bmatrix}$$

The DMD method approximates the modes of the Koopman operator, which is a linear, time-independent operator that

$$\mathbf{X}_{j+1} = \mathbf{A}\mathbf{X}_j$$

Find such linear operator allowed us to predict states at any time, but the actual valid prediction range is bounded.

To do such decomposition, we find matrices $\mathbf{X_1}$ and $\mathbf{X_2}$ such that,

$$\mathbf{X_1} = \mathbf{X_1^{M-1}} = [\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_{M-1}}] = [\mathbf{x_1}, \mathbf{A}\mathbf{x_1}, \dots, \mathbf{A^{M-2}}\mathbf{x_1}]$$

$$\mathbf{X_2} = \mathbf{X_2^M} = [\mathbf{x_2}, \mathbf{x_3}, \ldots, \mathbf{x_k}] = [\mathbf{Ax_1}, \mathbf{A^2x_1}, \ldots, \mathbf{A^{M-1}x_1}]$$

Therefore, we can rewrite $\mathbf{X_2}$ as

$$\mathbf{X_2} = \mathbf{AX_1} + \mathbf{r}e_{M-1}^T.$$

where the second term is the error of $\mathbf{x_M}$ since it is not include in the Krylox space of $\mathbf{X_1}$.

Again, our goal is to find the operator $\mathbf{A}$, therefore let's apply SVD on $\mathbf{X_1}$, and we have

$$\mathbf{X_2} = \mathbf{AU\Sigma V}^* + \mathbf{r}e_{M-1}^T.$$

Now, we are going to find $\mathbf{A}$ when $\mathbf{X_2}$ is projected to the space with the basis of $\mathbf{U}$. In other words, we want to find the matrix $\mathbf{A}$ in the POD modes. Therefore, the error must be orthogonal to the POD basis, which leads to $\mathbf{U}^*\mathbf{r} = \mathbf{0}$.

$$\mathbf{U}^*\mathbf{X_2} = \mathbf{U}^*\mathbf{AU\Sigma V}^*.$$

By moving variables around, the final equation will be

$$\mathbf{U}^*\mathbf{AU} = \mathbf{U}^*\mathbf{X_2\Sigma V}^* = \tilde{\mathbf{S}}.$$

Next, suppose $\mathbf{y_k}$ and μ_k are the eigen vector and eigenvalues of $\tilde{\mathbf{S}}$. Therefore,

$$\begin{aligned} \tilde{\mathbf{S}}\mathbf{y_k} &= \mu_k \mathbf{y_k} \\ \mathbf{U}^*\mathbf{AU}\mathbf{y_k} &= \mu_k \mathbf{y_k} \\ \mathbf{AU}\mathbf{y_k} &= \mu_k \mathbf{U}\mathbf{y_k} \end{aligned} \tag{1}$$

The computation above shows that the eigen vector of $\mathbf{A}$ us $\mathbf{Uy_k}$, which is also the DMD modes ϕ_k. The continues multiplication of matrix $\mathbf{A}$ can be given by expanding it in eigen basis, thus the prediction model is described as

$$\mathbf{X_{DMD}}(\mathbf{t}) = \sum_{k=1}^{K} b_k \phi_k e^{w_k t} = \Phi \mathbf{diag}(e^{w_k t})\mathbf{b},$$

where $e^{w_k t}$ is just another way to write the eigenvalues μ_k, and the matrix b is the initial condition, which is given by $\Phi^\dagger x_i$, and x_i is the initial state we want to start with.

3 Algorithm Implementation and Development

3.1 Preparation

Since the videos are provided in `MP4` format, we have to load the videos and reshape it before we actually apply the algorithm.

Algorithm 1: Preparation

Load the video by `VideoReader`
Store information including total frame number, fps, frame height, and frame width
for each frame **do**
  Turn the frame into gray-scale image
  Flat the matrix into a vector, which will be $\mathbf{X_i}$
end for

3.2 Finding the low-rank reconstruction and applying the DMD.

Notice that in the DMD, there is a step which we need to apply the SVM on $\mathbf{X_1}$. By doing this, we are able to find the principle components of the frame at each time t_i. Remember that, the component with the heaviest weight is the one shared in all frames! Therefore, if we select some heavy components, we are probably going to capture the background of the video.

Algorithm 2: Finding the low-rank reconstruction and applying the DMD.

...(normal DMD procedure)
Apply the SVM on $\mathbf{X_1}$
Select first n components for reconstruction, and find the corresponding $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$
...(normal DMD procedure)

3.3 Finding the background and the foreground

Since the DMD reconstruction gives us complex numbers, therefore we have to do moves to get a clear data of the background and the foreground.

Algorithm 3: Finding the background and the foreground

Find $\mathbf{X_{sparse}}$ by $\mathbf{X1} - |\mathbf{X_{DMD}^{low-rank}}|$
Find the real part $\mathbf{R}$ of $\mathbf{X_{sparse}}$
Find the background by $\mathbf{R} + |\mathbf{X_{DMD}^{low-rank}}|$
Find the foreground by $\mathbf{X_{sparse}} - \mathbf{R}$

4 Computational Result

Here are extractions of the foreground and the background

4.1 monte_carlo_low.mp4

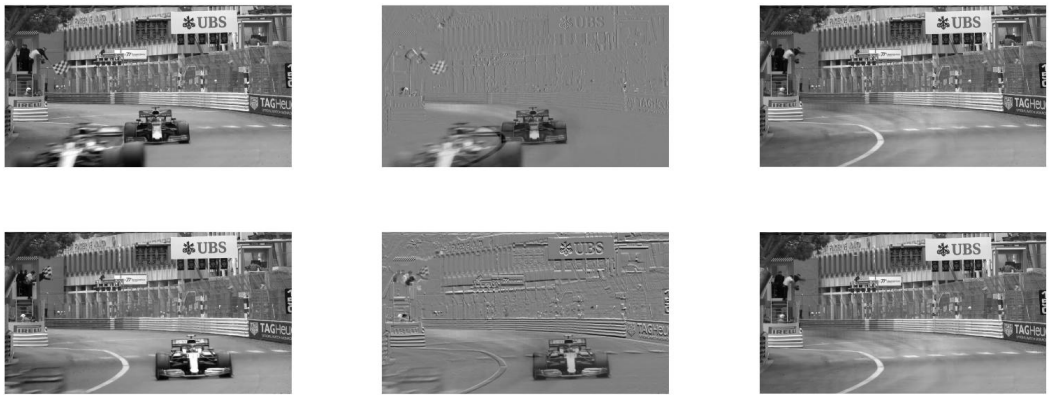

Figure 1: Original (left), Foreground (middle), and Background (right)

Here are extractions of the foreground and the background

4.2 ski_drop_low.mp4

5 Summary and Conclusion

In the study, we applied the SVD to analyze the components, the weights, and the projections of these handwritten digit images. Moreover, we implemented LDA and LDS, creating a binary classier and a trinary

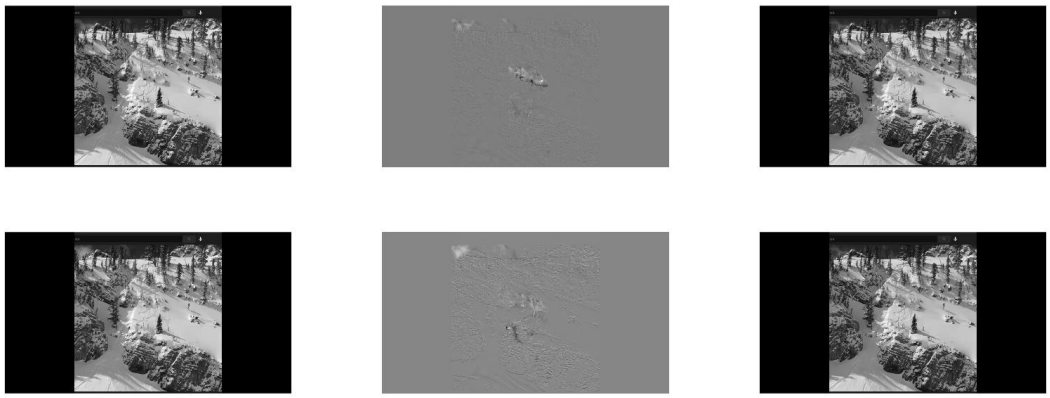

Figure 2: Original (left), Foreground (middle), and Background (right)

classier. Lastly, we compared LDA with DTC and SVM. It turns out the classifier based on LDA is the best among three classier. SVM and DTC are the second and the third respectively.

Appendix A MATLAB Functions

- `tree = fitctree(X,Y)` returns a fitted binary classification decision tree based on the input variables contained in matrix X and output Y. The returned binary tree splits branching nodes based on the values of a column of X.
- `Mdl = fitcsvm(X,Y)` returns an SVM classifier trained using the predictors in the matrix X and the class labels in vector Y for one-class or two-class classification.
- `ypred = predict(mdl,Xnew)` returns the predicted response values of the linear regression model mdl to the points in Xnew.

Appendix B MATLAB Code

```
clear all; close all; clc;

[data, train_labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');

data = double(data);
data = data(:)';
train_images = reshape(data(:), [784, 60000]);
% train_images = train_images - repmat(mean(train_images, 1), 784, 1);


[data, test_labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');

data = double(data);
data = data(:)';
test_images = reshape(data(:), [784, 10000]);
% test_images = test_images - repmat(mean(test_images, 1), 784, 1);
```

```
    %% part 1.1-2
    [U, S, V] = svd(train_images, 'econ');

    %% check U
    for k = 1:25
        subplot(5,5,k)
        u = reshape(U(:,k), 28, 28);
        u = rescale(u);
        imshow(u)
    end

%% check S
close all;
s = diag(S);

figure(1)
subplot(2,1,1)
plot(s, 'ko','Linewidth',1)
ylabel('\sigma')
title('\sigma after applying SVD on the training set')

subplot(2,1,2)
semilogy(s,'ko','Linewidth',1)
ylabel('\sigma (log scale)')
title('\sigma after applying SVD on the training set (log scale)')
%% V
basis = [2,3,5];
digits = [];
for i = 1:10
   digit_label = find(train_labels == i-1);
   digits = [digits; digit_label(1:10)'];
end

v = V(:, basis);
for i = 1:10
    plot3(v(digits(i,:), 1), v(digits(i,:),2), v(digits(i,:),3), 'o'); hold on;
    xlabel('second principle component')
    ylabel('third principle component')
    zlabel('fifth principle component')
end

%% part 2.1
close all;
d1 = get_images_by_label(0, train_images, train_labels);
d2 = get_images_by_label(1, train_images, train_labels);

dt1 = get_images_by_label(0, test_images, test_labels);
dt2 = get_images_by_label(1, test_images, test_labels);

[U, w, threshold] = lda2d(d1, d2, 700);
accur = lda2d_test(dt1, dt2, U, w, threshold);

%% part 2.2
```

```
close all; clc;
d1 = get_images_by_label(0, train_images, train_labels);
d2 = get_images_by_label(1, train_images, train_labels);
d3 = get_images_by_label(2, train_images, train_labels);
[U, w, t1, t2] = lda3d(d1, d2, d3, 700);

%% part 2.3-4
% calculation
accur = ones(10);
for i = 1:10
    for j = 1:10
        if i == j
            continue
        end
        d1 = get_images_by_label(i-1, train_images, train_labels);
        d2 = get_images_by_label(j-1, train_images, train_labels);

        dt1 = get_images_by_label(i-1, test_images, test_labels);
        dt2 = get_images_by_label(j-1, test_images, test_labels);

        [U, w, threshold] = lda2d(d1, d2, 700);
        accur(i, j) = lda2d_test(d1, d2, U, w, threshold);
    end
end
%% plot
load('DTC_accur.mat');
load('SVM_accur.mat');

accur = SVM_accur;
[X,Y] = meshgrid(0:9,0:9);
surf(X, Y, accur);
colorbar
xlabel('first digit')
ylabel('second digit')
zlabel('accuracy')

%% find the min and amx
[Min, I] = min(accur(:));
[min_d1, min_d2] = ind2sub(size(accur),I);
accur_d = accur - diag(ones(1,10));
[Max, I] = max(accur_d(:));
[max_d1, max_d2] = ind2sub(size(accur_d),I);

%% part 2.5 DTC and SVM
DTC_accur = ones(10);
SVM_accur = ones(10);
for i = 0:9
    for j = 0:9
        if i == j
            continue
        end
        dtr1 = get_images_by_label(i, train_images, train_labels);
        dtr2 = get_images_by_label(j, train_images, train_labels);
```

```
        dte1 = get_images_by_label(i, test_images, test_labels);
        dte2 = get_images_by_label(j, test_images, test_labels);

        d_train = [dtr1, dtr2];
        l_train = [ones(1, size(dtr1, 2)) * (i), ones(1, size(dtr2, 2)) * (j)];

        d_test = [dte1, dte2];
        l_test = [ones(1, size(dte1, 2)) * (i), ones(1, size(dte2, 2)) * (j)];

        % DTC
        DTC = fitctree(d_train', l_train);
        p = predict(tree, d_test')';
        DTC_accur(i+1, j+1) = sum((p - l_test) == 0) / size(d_test, 2);

        % SVM
        %SVM = fitcsvm(d_train', l_train);
        %p = predict(SVM, d_test')';
        %SVM_accur(i, j) = sum((p - l_test) == 0) / size(d_test, 2);
    end
end



function mat = get_images_by_label(digit,images,labels)
    indices = labels == digit;
    mat = images(:, indices);
end

function [U, w,threshold] = lda2d(d1,d2,feature)
    n1 = size(d1,2);
    n2 = size(d2,2);

    [U,S,V] = svd([d1 d2],'econ');

    d = S*V';
    U = U(:,1:feature);

    d1 = d(1:feature, 1:n1);
    d2 = d(1:feature, n1+1:n1+n2);
    m1 = mean(d1,2);
    m2 = mean(d2,2);

    d1s = d1 - m1;
    d2s = d2 - m2;
    Sw = d1s*d1s' + d2s * d2s';
    Sb = (m1-m2)*(m1-m2)'; % between class

    [V2, D] = eig(Sb,Sw); % linear disciminant analysis
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);

    v1 = w'*d1;
    v2 = w'*d2;
```

```
    if mean(v1) > mean(v2)
        w = -w;
        v1 = -v1;
        v2 = -v2;
    end
    %figure(4)
    %plot(v1,0,'ob')
    %hold on
    %plot(v2,1,'dr')
    %ylim([0 1])

    % Find the threshold value

    s1 = sort(v1);
    s2 = sort(v2);

    t1 = length(s1);
    t2 = 1;
    while s1(t1) > s2(t2)
        t1 = t1 - 1;
        t2 = t2 + 1;
    end

    threshold = (s1(t1) + s2(t2))/2;

    % Plot histogram of results
    %figure(6)
    %subplot(1,2,1)
    %histogram(s1,30); hold on, plot([threshold threshold], [0 1400],'r')
    %title('Train: digit 0')
    %subplot(1,2,2)
    %histogram(s2,30); hold on, plot([threshold threshold], [0 3000],'r')
    %title('Train: digit 1')
end


function [U, w, t1, t2] = lda3d(d1,d2,d3, feature)
    n1 = size(d1,2);
    n2 = size(d2,2);
    n3 = size(d3,2);


    [U,S,V] = svd([d1 d2 d3],'econ');

    d = S*V';
    U = U(:,1:feature);

    d1 = d(1:feature, 1:n1);
    d2 = d(1:feature, n1+1:n1+n2);
    d3 = d(1:feature, n1+n2+1:n1+n2+n3);

    m1 = mean(d1,2);
    m2 = mean(d2,2);
```

```
m3 = mean(d3,2);
m = mean([d1,d2,d3],2);

Sw = 0;
for k = 1:n1
    Sw = Sw + (d1(:,k) - m1)*(d1(:,k) - m1)';
end
for k = 1:n2
    Sw =  Sw + (d2(:,k) - m2)*(d2(:,k) - m2)';
end
for k = 1:n3
    Sw =  Sw + (d3(:,k) - m3)*(d3(:,k) - m3)';
end

Sb = (m1-m)*(m1-m)' + (m2-m)*(m2-m)' + (m3-m)*(m3-m)';

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

v1 = w'*d1;
v2 = w'*d2;
v3 = w'*d3;


plot(v1,0,'ob','Linewidth',2)
hold on
plot(v2,1,'dr','Linewidth',2)
plot(v3,2,'ok','Linewidth',2)

ylim([0 2.2])

% Find the threshold value

s1 = sort(v1);
s2 = sort(v2);
s3 = sort(v3);


t1 = length(s3);
t2 = 1;
while s2(t1) > s3(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end

th1 = (s2(t1) + s3(t2))/2;

t1 = length(s1);
t2 = 1;
while s3(t1) > s1(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
```

```
    end

    th2 = (s3(t1) + s1(t2))/2;

    figure(5)
    subplot(1,3,1)
    histogram(s1,30); hold on, plot([th2 th2], [0 1000],'r')
    title('Train: digit 0')
    subplot(1,3,2)
    histogram(s2,30); hold on, plot([th1 th1], [0 2500],'r')
    title('Train: digit 1')
    subplot(1,3,3)
    histogram(s3,30); hold on, plot([th2 th2], [0 1000],'r'); plot([th1 th1], [0 1000],'r')
    title('Train: digit 2')

end


function accur = lda2d_test(d1, d2, U, w, threshold)
    feature = size(w, 1);

    n1 = size(d1,2);
    n2 = size(d2,2);

    d = U'*[d1 d2];
    U = U(:,1:feature);

    d1 = d(1:feature, 1:n1);
    d2 = d(1:feature, n1+1:n1+n2);

    s1 = sort(w'*d1);
    s2 = sort(w'*d2);

    count = sum(s1 < threshold) + sum(s2 > threshold);
    accur = count / (n1+n2);



end


function [imgs_train, labels] = mnist_parse(path_to_digits, path_to_labels)

% The function is curtesy of stackoverflow user rayryeng from Sept. 20,
% 2016. Link: https://stackoverflow.com/questions/39580926/how-do-i-load-in-the-mnist-digits-and-label-

% Open files
fid1 = fopen(path_to_digits, 'r');

% The labels file
fid2 = fopen(path_to_labels, 'r');

% Read in magic numbers for both files
A = fread(fid1, 1, 'uint32');
```

```
magicNumber1 = swapbytes(uint32(A)); % Should be 2051
fprintf('Magic Number - imgs_train: %d\n', magicNumber1);

A = fread(fid2, 1, 'uint32');
magicNumber2 = swapbytes(uint32(A)); % Should be 2049
fprintf('Magic Number - Labels: %d\n', magicNumber2);

% Read in total number of imgs_train
% Ensure that this number matches with the labels file
A = fread(fid1, 1, 'uint32');
totalimgs_train = swapbytes(uint32(A));
A = fread(fid2, 1, 'uint32');
if totalimgs_train ~= swapbytes(uint32(A))
    error('Total number of imgs_train read from imgs_train and labels files are not the same');
end
fprintf('Total number of imgs_train: %d\n', totalimgs_train);

% Read in number of rows
A = fread(fid1, 1, 'uint32');
numRows = swapbytes(uint32(A));

% Read in number of columns
A = fread(fid1, 1, 'uint32');
numCols = swapbytes(uint32(A));

fprintf('Dimensions of each digit: %d x %d\n', numRows, numCols);

% For each image, store into an individual slice
imgs_train = zeros(numRows, numCols, totalimgs_train, 'uint8');
for k = 1 : totalimgs_train
    % Read in numRows*numCols pixels at a time
    A = fread(fid1, numRows*numCols, 'uint8');

    % Reshape so that it becomes a matrix
    % We are actually reading this in column major format
    % so we need to transpose this at the end
    imgs_train(:,:,k) = reshape(uint8(A), numCols, numRows).';
end

% Read in the labels
labels = fread(fid2, totalimgs_train, 'uint8');

% Close the files
fclose(fid1);
fclose(fid2);

end
```