Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Industriales

Máster en Automática y Robótica

Applied Artificial Intelligence

Assignment 2.3: Linear Discriminant Analysis

Josep María Barberá Civera (17048)

March 9, 2024

ASSIGNMENT 2.3

Linear Discriminant Analysis

Similarly to the PCA the steps followed for the LDA are described.

1. First load the synthetically created "data_D2_C2" data and then proceed to normalise the "p.value" dataset.
2. Next, the scatter matrices are computed (namely S1, S2, Sw, Sb, and St).
3. Then the Multidimensional Fisher Discriminant is maximised thanks to the $S_W^{-1} S_B$ eigen vectors
4. Finally, the data is projected and plotted next to the original data set (raw data).

1.1 Methodology

Based on the Supervised Linear Processing technique called Linear Discriminant Analysis (or LDA shortly), we are looking for the classification of our labelled data. The Fisher Discriminant states that the best line in which to project the data is the one that links the media centres of each class weighted by the inverse of the dispersion within each class. In the Multidimensional Fisher Discriminant the projection line is computed directly as the eigen vector of the product $S_W^{-1} S_B$, where S_W is the scatter matrix within the classes and the S_B is the scatter matrix between the classes.

With the help of the code developed for the scatter matrix assignment we have extended it to include the LDA.

Let us denote our raw data by X, where X corresponds to a lengthy matrix with two rows and numerous columns. The first row signifies the position along the x-axis, while the second row denotes positions along the y-axis. Each column pertains to a point, resulting in a total of 300 data points.

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & \dots & x_{1,300} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & \dots & x_{2,300} \end{bmatrix}$$

Additionally, there exists a row vector with 300 entries, each assigning a class label to every point, with only two distinct classes present.

It is important to normalise the data, we can achieve this subtracting the mean (μ) and dividing by the standard deviation (σ) to each element in the data,

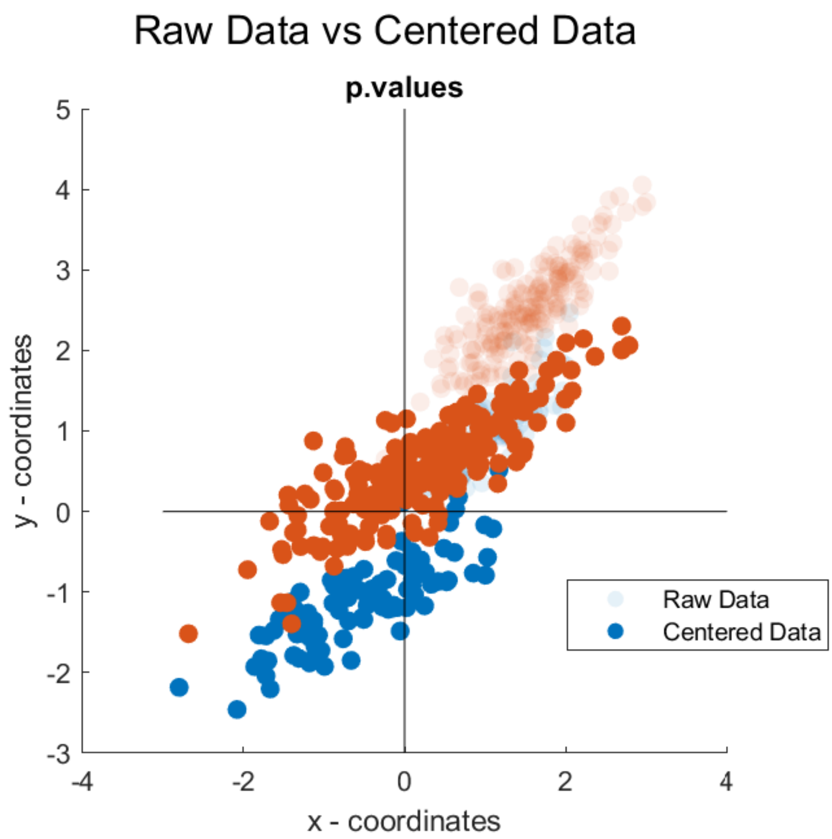

Figure 1: *Raw data vs Normalised data*

$$x_{ni} = \frac{x_i - \mu}{\sigma}$$

If we plot the raw data along with the normalised one we will observe how the data has centred, as seen in Figure 1.

Then the scatter matrices should be computed. As we have two groups we should split the initial data into two groups called X_1 and X_2. For each group it is easily to compute the **scatter matrix inside the group** as

$$S_c = \sum_{i \in c} (x_{ni} - \mu_c)(x_{ni} - \mu_c)^T = cov(X_c^T) \cdot (n_c - 1)$$

where $c = \{X_1, X_2\}$ refers to each group or class, so μ_c is the mean and σ_c is the standard deviation within the X_1 or X_2 class data points. Also, n_c stands for the number of data points in each class. Then the **scatter matrix within the groups** can be easily obtain as

$$S_W = \sum_c S_c.$$

Also the **scatter matrix between the groups** is computed as follows,

$$S_B = \sum_c n_c(\mu_c - \mu)(\mu_c - \mu)^T$$

As previously mentioned, the projection matrix for the best classification can be obtained from the eigen vectors of the product between S_W^{-1} and S_B in the following manner:

$$W = eig(S_W^{-1} S_B)$$

Then the projection is easily done selecting the preferred eigen vector and multipliying it element wise with the matrix data (X) resulting in a one dimensional vector:

$$w = W_1^T$$

and multiplying it with the data

$$y = wX$$

Finally, we will desproject the data,

$$\hat{X}_n = w^T y$$

but also desnormalise it as follows,

$$\hat{X} = \hat{X}_n \sigma + \mu$$

1.2 Discussion and Results

The values for some of the aforementioned matrices are compiled here,

$$Sw = \begin{bmatrix} 257.8740 & 144.7179 \\ 144.7179 & 116.0240 \end{bmatrix} \quad Sb = \begin{bmatrix} 41.1260 & 86.7471 \\ 86.7471 & 182.9760 \end{bmatrix}$$

$$W = \begin{bmatrix} -0.9036 & 0.4283 \\ 0.4284 & -0.9036 \end{bmatrix} \qquad D = \begin{bmatrix} 0.0000 & 0 \\ 0 & 2.9911 \end{bmatrix}$$

As we can notice, matrix D stands for the eigen values, particularly, the second one refers to the second eigen vector which we are going to use to project the data.

Once we have the deprojected and denormalised data we can plot it along with the initial data (please refer to Figure 2.

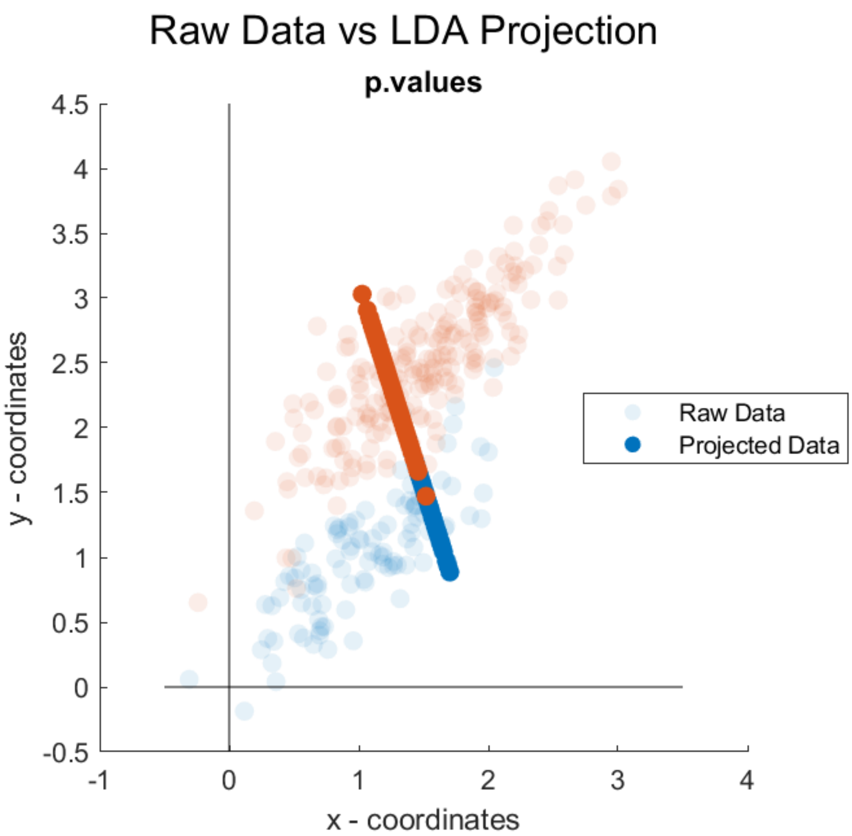

Figure 2: *Raw data vs Projected data*

1.3 Relevant Code

The code prepared for this assignment is shown in the next pages in a wider style.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                           Master in Robotics
%                      Applied Artificial Intelligence
%
% Assinment 2.3: Linear Discriminant Analysis
% Student: Josep Barbera Civera
% ID: 17048
% Date: 09/03/2024
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 0. Load data_D2_C2 and normalize the data (p.value and t.value)
% 1. Compute array Wnlda that maximize the Fisher discriminant
% 2. Compute the 1D coordinates of the test data projected onto Wnlda
% 3. Compute and plot the 2D dimensional coordinates of the above
% projected data once denormalized (see figure for result)
% 4. Compute the reconstruction MSE of the not normalized data

load data_D2_C2.mat

%% Accesing Data
pvalues = p.value;
plabels = p.class;

%% Normalizing data
disp("-------- Normalizing Data ---------------------------");
disp("Normalizing data...");
[Mp, Np] = size(pvalues);
mn_p = mean(pvalues')';
std_p = std(pvalues')';
for i = 1:Np
    pn(:,i) = (pvalues(:,i) - mn_p)./std_p;
end

%% Plotting Normalized Data vs Raw Data
% one_plot('Raw Data vs Centred Data', 'p.values', ...
%              'x - coordinates', 'y - coordinates', 'Raw Data', ...
%              'Centered Data', pvalues, pn, ...
%              plabels, 'centred_data_vs_raw_data.png');

%% Computing Scatter Matrices
pmatrices = Scatter_matrices(pn, plabels);
S1 = pmatrices{1};
S2 = pmatrices{2};
Sw = pmatrices{3};
Sb = pmatrices{4};
St = pmatrices{5};

%% Computing W matrix: maximizing the Fisher Discriminant
[W, D] = eig(inv(Sw)*Sb);

%% Sort the variances in decreasing order
disp("-------- Sorting variances in decreasing order ---------");
% Extract diagonal of matrix as vector
```

```
D = diag(D);
% Sort W and convert D to a column vector with the eigenvalues
[~, p_rindices] = sort(-1*D);
D = D(p_rindices);
W = W(:, p_rindices);


%% Computing projection
w = W(:,1)';
y = w * pn;

%% Desprojection
x_n = w' * y;
for i=1:Np
    x(:,i) = x_n(:,i) .* std_p + mn_p;
end

%% Plotting Data vs LDA projection
one_plot('Raw Data vs LDA Projection', ...
            'p.values', 'x - coordinates', ...
            'y - coordinates', ...
          'Raw Data', 'Projected Data', pvalues, x, ...
            plabels, 'lda_plot_first.png');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Auxiliar Functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function one_plot(generic_title, title_subplot_1, ...
                    x_1_label, y_1_label, ...
                    legend_1_light, legend_1, ...
                    light_data_1, data_1, ...
                    labels_1, saved_name)
    disp("-------- Plotting ----------------------------------");
    disp(generic_title);
    figure;
    % Generic Title
    sgtitle(generic_title);
    % First Subplot
    subplot(1, 1, 1);
    for i=1:length(labels_1)
        if labels_1(i) == 1
            scatter(light_data_1(1,i), light_data_1(2,i), 50, 'o', ...
                    'MarkerEdgeColor', 'none', 'MarkerFaceColor', [0
                       0.4470 0.7410], ...
                    'MarkerFaceAlpha', 0.1); hold on;
            scatter(data_1(1,i), data_1(2,i), 50, 'o', ...
                    'MarkerEdgeColor', 'none', 'MarkerFaceColor', [0
                       0.4470 0.7410], ...
                    'MarkerFaceAlpha', 1); hold on;
        else
            scatter(light_data_1(1,i), light_data_1(2,i), 50, 'o', ...
                    'MarkerEdgeColor', 'none', 'MarkerFaceColor',
                       [0.8500 0.3250 0.0980], ...
                    'MarkerFaceAlpha', 0.1); hold on;
```

```
                scatter(data_1(1,i), data_1(2,i), 50, 'o', ...
                        'MarkerEdgeColor', 'none', 'MarkerFaceColor',
                           [0.8500 0.3250 0.0980], ...
                        'MarkerFaceAlpha', 1); hold on;
            end
        end
        % Plot vertical lines for x=0 and y=0
        plot([0 0], ylim, 'k-');
        plot(xlim, [0 0], 'k-');
        % Subplot title
        title(title_subplot_1);
        % Axis labels
        xlabel(x_1_label);
        ylabel(y_1_label);
        legend({legend_1_light, legend_1}, 'Location', 'best');
        pbaspect([1 1 1]);
        % pos = get(gcf, 'Position');
        % set(gcf, 'Position',pos+[-900 -300 900 300])
        saveas(gca, saved_name);
end

function matrix_cell = Scatter_matrices(values, labels)
        % first we compute the number of labels
        unique_labels = unique(labels);
        N = length(unique_labels);
        % now we create as many vectors as labels: with cell arrays
        vectors_cell = cell(1, N);
        matrix_cell = cell (1, N+3);

        for i = 1:N
            label = unique_labels(i);
            indices = labels == label; % Find indices corresponding to
               the current label
            vectors_cell{i} = values(:, indices); % Store coordinates
               associated with the label
        end
        % Scatter matrix for each group is computed
        for i = 1:N
            data = vectors_cell{i};
            Sc = cov(data')*(length(data)-1);
            matrix_cell{i} = Sc;
        end
        % Scatter matrix within the groups is computed
        Sw = zeros(N);
        for i = 1:N
            Sw = Sw + matrix_cell{i};
        end
        s = N + 1;
        matrix_cell{s} = Sw;
        % Scatter matrix between the groups is computed
        Sb = zeros(N);
        m_x = mean(values(1,:));
        m_y = mean(values(2,:));
        m = [m_x; m_y];
```

```
        for i = 1:N
            data = vectors_cell{i};
            m_c_x = mean(data(1,:));
            m_c_y = mean(data(2,:));
            m_c = [m_c_x; m_c_y];
            Sb = Sb + length(data)*(m_c-m)*(m_c-m).';
        end
        s = s + 1;
        matrix_cell{s} = Sb;
        % Total Scatter matrix is computed
        St = (values - m)*(values - m).';
        s = s + 1;
        matrix_cell{s} = St;
end
```