**Code:**

clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The following function performs

% the Mahalanobis distance classification

function [class\_labels] = maha\_dist(m, V, Feature)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Input variables:

% m: mean vector of the normal distribution. (Kx1)

% V: covariance matrix of the normal distribution. (KxK)

% Feature: the feature that will be tested. (NxF)

%

% Output variable:

% class\_labels: the predicted class labels for each feature (1xF)

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[~, K] = size(m); % K: số lượng lớp

[~, F] = size(Feature); % F: số lượng đặc trưng đầu vào

class\_labels = zeros(1, F); % Khởi tạo mảng nhãn lớp đầu ra

for i = 1:F

distances = zeros(1, K); % Khởi tạo khoảng cách Mahalanobis cho từng lớp

for ii = 1:K

% Tính khoảng cách Mahalanobis cho mỗi lớp

distances(ii) = sqrt((Feature(:, i) - m(:, ii))' \* inv(V) \* (Feature(:, i) - m(:, ii)));

end

% Xác định nhãn lớp với khoảng cách nhỏ nhất

[~, class\_labels(i)] = min(distances);

end

end

% Số lượng đặc trưng và lớp

K = 3; % Số lượng lớp

F = 5; % Số lượng đặc trưng

% Dữ liệu giả lập

m = randn(2, K); % Vector trung bình cho 3 lớp (Kx1)

V = cov(randn(100, 2)); % Ma trận hiệp phương sai (2x2)

Feature = randn(2, F); % Đặc trưng đầu vào (NxF)

% Gọi hàm phân loại

class\_labels = maha\_dist(m, V, Feature);

% Hiển thị kết quả

disp('Class labels:');

disp(class\_labels);

**Mô tả:**

**Chức năng của hàm**: Hàm maha\_dist thực hiện phân loại dựa trên khoảng cách Mahalanobis. Đầu vào bao gồm:

* m: vector trung bình của các lớp.
* V: ma trận hiệp phương sai.
* Feature: đặc trưng cần phân loại.

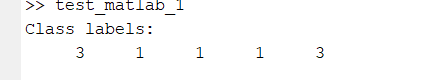
**Cách hoạt động**:

* Tính khoảng cách Mahalanobis từ đặc trưng đến mỗi lớp.
* Chọn lớp có khoảng cách nhỏ nhất làm nhãn lớp đầu ra.

**Đầu ra**:

* class\_labels: danh sách nhãn lớp tương ứng với từng đặc trưng đầu vào.

**Kết quả:**

****

**Code:**

clear

function [class\_labels] = maha\_dist(m, V, Feature)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Input variables:

% m: mean vector of the normal distribution. (Kx1)

% V: covariance matrix of the normal distribution. (KxK)

% Feature: the feature that will be tested. (NxF)

%

% Output variable:

% class\_labels: the predicted class labels for each feature (1xF)

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[~, K] = size(m); % K: số lượng lớp

[~, F] = size(Feature); % F: số lượng đặc trưng đầu vào

class\_labels = zeros(1, F); % Khởi tạo mảng nhãn lớp đầu ra

for i = 1:F

distances = zeros(1, K); % Khởi tạo khoảng cách Mahalanobis cho từng lớp

for ii = 1:K

% Tính khoảng cách Mahalanobis cho mỗi lớp

distances(ii) = sqrt((Feature(:, i) - m(:, ii))' \* inv(V) \* (Feature(:, i) - m(:, ii)));

end

% Xác định nhãn lớp với khoảng cách nhỏ nhất

[~, class\_labels(i)] = min(distances);

end

end

% Extracting two features from two PPG signals

% The first signal is normotensive, and the second is hypertensive

Feature1 = [1.12; 0.11]; % Đặc trưng của tín hiệu normotensive

Feature2 = [0.7; 0.33]; % Đặc trưng của tín hiệu hypertensive

% Mean vectors m1 và m2

m1 = [1.12; 0.11];

m2 = [0.7; 0.33];

% Covariance matrix = identity matrix

V = eye(2); % Ma trận hiệp phương sai (2x2)

% Mean matrix (tập hợp các vector trung bình của các lớp)

m = [m1, m2];

% Sử dụng ma trận hiệp phương sai S = V (giả lập)

S = V;

% Tính khoảng cách Mahalanobis cho từng đặc trưng

J1 = maha\_dist(m, S, Feature1);

J2 = maha\_dist(m, S, Feature2);

% Hiển thị kết quả

disp('Mahalanobis distances for Feature1:');

disp(J1);

disp('Mahalanobis distances for Feature2:');

disp(J2);

**Mô tả:**

**Đoạn mã chính**:

* Feature1 và Feature2 là đặc trưng của hai tín hiệu PPG (normotensive và hypertensive).
* m1 và m2 là vector trung bình của hai lớp.
* V là ma trận hiệp phương sai (ở đây giả sử là ma trận đơn vị).

**Hoạt động**:

* Tính khoảng cách Mahalanobis giữa Feature1/Feature2 và các lớp xác định bởi vector trung bình (m) và ma trận hiệp phương sai (V).
* Kết quả J1 và J2 biểu diễn nhãn lớp tương ứng với từng đặc trưng dựa trên khoảng cách Mahalanobis.

**Ý nghĩa**:

* Khoảng cách Mahalanobis đo độ tương đồng giữa một đặc trưng với một phân phối (xác định bởi trung bình và hiệp phương sai).
* Ứng dụng trong phân loại tín hiệu y sinh học (như PPG) để xác định lớp của tín hiệu.

**Kết quả:**

**A white background with black text

Description automatically generated**

**CODE:**

clear

function [p] = Gauss\_posteriori(m, V, Feature)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Input variables:

% m: mean vector of the normal distribution (Kx1)

% V: covariance matrix of the normal distribution (KxK)

% Feature: the feature that will be tested (Kx1)

%

% Output variable:

% p: the value of the normal distribution for the Feature variable

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[K, ~] = size(m); % K: số chiều của vector trung bình

p\_x = (1 / ((2 \* pi)^(K / 2) \* det(V)^0.5)) \* ...

exp(-0.5 \* (Feature - m)' \* inv(V) \* (Feature - m));

p = p\_x; % Kết quả là xác suất của Feature theo phân phối Gauss

end

% Khởi tạo đặc trưng và các thông số

Feature = [1.12; 0.11]; % Đặc trưng cần tính xác suất

m = [1.0; 0.1]; % Vector trung bình của phân phối (2x1)

V = eye(2); % Ma trận hiệp phương sai (2x2)

% Tính xác suất của đặc trưng

p = Gauss\_posteriori(m, V, Feature);

% Hiển thị kết quả

disp('The posterior probability density for the feature is:');

disp(p);

**MÔ TẢ:**

**Hàm Gauss\_posteriori**:

* Tính giá trị mật độ xác suất của một điểm (Feature) theo phân phối chuẩn đa chiều được xác định bởi:
  + m: vector trung bình (mean vector).
  + V: ma trận hiệp phương sai (covariance matrix).

**Đầu vào và đầu ra**:

* **Đầu vào**:
  + m: Vector trung bình (kích thước Kx1).
  + V: Ma trận hiệp phương sai (kích thước KxK).
  + Feature: Đặc trưng (kích thước Kx1).
* **Đầu ra**:
  + p: Xác suất mật độ tại điểm đặc trưng Feature.

**KẾT QUẢ:**

**A black text on a white background

Description automatically generated**

**CODE:**

clear

function [p] = Gauss\_posteriori(m, V, Feature)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Input variables:

% m: mean vector of the normal distribution (Kx1)

% V: covariance matrix of the normal distribution (KxK)

% Feature: the feature that will be tested (Kx1)

%

% Output variable:

% p: the value of the normal distribution for the Feature variable

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[K, ~] = size(m); % K: số chiều của vector trung bình

p\_x = (1 / ((2 \* pi)^(K / 2) \* det(V)^0.5)) \* ...

exp(-0.5 \* (Feature - m)' \* inv(V) \* (Feature - m));

p = p\_x; % Kết quả là xác suất của Feature theo phân phối Gauss

end

% Extracting two features from two PPG signals

% The first signal is normotensive, and the second signal is hypertensive

% Đặc trưng của tín hiệu normotensive và hypertensive

Feature1 = [1.12; 0.11];

Feature2 = [0.7; 0.33];

% Xác suất tiên nghiệm của các lớp

P1 = 0.5; % a priori probability of class 1

P2 = 0.5; % a priori probability of class 2

% Giá trị trung bình (mean) của hai lớp

m1 = [1.12; 0.11];

m2 = [0.7; 0.33];

% Ma trận hiệp phương sai là ma trận đơn vị

V = eye(2);

% Tính xác suất hậu nghiệm (posterior probability)

p1 = P1 \* Gauss\_posteriori(m1, V, Feature1); % Xác suất Feature1 thuộc class 1

p2 = P2 \* Gauss\_posteriori(m2, V, Feature1); % Xác suất Feature1 thuộc class 2

p3 = P1 \* Gauss\_posteriori(m1, V, Feature2); % Xác suất Feature2 thuộc class 1

p4 = P2 \* Gauss\_posteriori(m2, V, Feature2); % Xác suất Feature2 thuộc class 2

% Hiển thị kết quả

disp('Posterior probabilities:');

fprintf('Feature1 belongs to class 1: %.4f\n', p1);

fprintf('Feature1 belongs to class 2: %.4f\n', p2);

fprintf('Feature2 belongs to class 1: %.4f\n', p3);

fprintf('Feature2 belongs to class 2: %.4f\n', p4);

**MÔ TẢ:**

**Mục đích**:

* Đoạn mã này tính xác suất hậu nghiệm (posterior probability) của hai đặc trưng Feature1 và Feature2 thuộc về hai lớp khác nhau (class 1 và class 2).
* Xác suất hậu nghiệm được tính dựa trên phân phối Gauss đa chiều (multivariate Gaussian distribution).

**Ý nghĩa các bước**:

* **Dữ liệu đầu vào**:
  + Feature1, Feature2: Các đặc trưng cần được phân loại.
  + P1, P2: Xác suất tiên nghiệm (prior probability) của các lớp (giả định bằng nhau là 0.5).
  + m1, m2: Vector trung bình của hai lớp.
  + V: Ma trận hiệp phương sai (giả định là ma trận đơn vị để đơn giản hóa tính toán).
* **Hàm Gauss\_posteriori**:
  + Tính xác suất mật độ của một đặc trưng theo phân phối Gauss đa chiều.
* **Tính xác suất hậu nghiệm**:
  + Kết hợp xác suất tiên nghiệm và xác suất mật độ để tính xác suất hậu nghiệm: P(class∣Feature)=P(class)⋅LikelihoodP(\text{class}|\text{Feature}) = P(\text{class}) \cdot \text{Likelihood}P(class∣Feature)=P(class)⋅Likelihood

**KẾT QUẢ:**

**A white text with black text

Description automatically generated**

**CODE:**

clear

% Extracting two features from two PPG signals

% The first signal is normotensive, and the second signal is hypertensive

% Đặc trưng của hai tín hiệu

Feature1 = [1.12; 0.11];

Feature2 = [0.7; 0.33];

% Tầng cạnh tranh với hai lớp

net1 = competlayer(2);

% Kết hợp các đặc trưng thành tập đặc trưng

feature = [Feature1, Feature2]; % Định dạng (num\_features x num\_samples)

% Cấu hình mạng

net1 = configure(net1, feature);

% Huấn luyện mạng với tập đặc trưng

net1 = train(net1, feature);

% Kiểm tra mạng với tập đặc trưng đã sử dụng để huấn luyện

outputs = net1(feature);

% Chuyển đổi kết quả đầu ra thành nhãn lớp

classes = vec2ind(outputs);

% Hiển thị kết quả

disp('Output classes:');

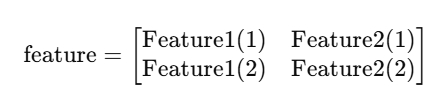
disp(classes);

**MÔ TẢ:**

**Mục đích**:

* Đoạn mã huấn luyện một mạng nơ-ron cạnh tranh (competitive layer) để phân loại hai tập dữ liệu đặc trưng Feature1 và Feature2.

**Ý nghĩa các bước**:

* **competlayer(2)**:
  + Tạo một tầng cạnh tranh với 2 lớp. Tầng này hoạt động dựa trên cơ chế cạnh tranh giữa các neuron để xác định lớp.
* **Kết hợp đặc trưng**:
  + Dữ liệu đặc trưng Feature1 và Feature2 được kết hợp thành ma trận feature với định dạng: 
* **Cấu hình và huấn luyện mạng**:
  + configure: Thiết lập mạng dựa trên dữ liệu đầu vào.
  + train: Huấn luyện mạng nơ-ron trên tập dữ liệu đặc trưng.
* **Dự đoán và phân loại**:
  + Đầu ra outputs là các giá trị từ mạng nơ-ron.
  + vec2ind(outputs): Chuyển đổi đầu ra thành nhãn lớp, giúp xác định đặc trưng thuộc về lớp nào.

**KẾT QUẢ:**

**A white background with black text

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**CODE:**

clear

Feature1=[1.12 0.11];

Feature2=[0.7 0.33];

Feature3=[1.01 0.10];

Feature4=[0.6 0.3];

feature=[Feature1;Feature2;Feature3;Feature4];

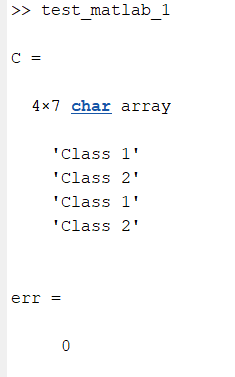
Group=['Class 1';'Class 2';'Class 1';'Class 2'];

[C,err,P,logp,coeff] = classify(feature,feature,Group,'linear')

**MÔ TẢ:**

Đoạn mã thực hiện phân loại tuyến tính (Linear Discriminant Analysis - LDA) để gán nhãn Class 1 hoặc Class 2 cho các đặc trưng (features). Mục đích là đánh giá khả năng phân biệt của mô hình dựa trên dữ liệu huấn luyện và kiểm tra. Kết quả bao gồm nhãn dự đoán, tỷ lệ lỗi, và xác suất cho từng nhãn.

**KẾT QUẢ:**

****

**A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generated**

**CODE:**

Feature1=[1.12 0.11];

Feature2=[0.7 0.33];

Feature3=[1.01 0.10];

Feature4=[0.6 0.3];

feature=[Feature1;Feature2;Feature3;Feature4];

Group=['Class 1';'Class 2';'Class 1';'Class 2'];

[C,err,P,logp,coeff] = classify(feature,feature,Group,'diagquadratic')

**MÔ TẢ:**

**Mục đích**: Thực hiện phân loại dựa trên phương pháp "diagonal quadratic discriminant analysis" (phân tích phân biệt bậc hai với giả định ma trận hiệp phương sai là đường chéo).

**Ý nghĩa các lệnh**:

* feature: Ma trận chứa các đặc trưng đầu vào.
* Group: Các nhãn lớp tương ứng cho từng đặc trưng.
* classify: Phân loại các đặc trưng dựa trên phương pháp diagquadratic.
* **Kết quả**:
  + C: Nhãn lớp dự đoán cho từng đặc trưng.
  + err: Tỷ lệ lỗi phân loại.
  + P: Xác suất hậu nghiệm cho từng lớp.
  + logp: Giá trị log của xác suất hậu nghiệm.
  + coeff: Hệ số phân biệt dùng trong phân loại.

**KẾT QUẢ:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**CODE:**

clear

Feature1 = [1.12 0.11];

Feature2 = [0.7 0.33];

Feature3 = [1.01 0.10];

Feature4 = [0.6 0.3];

feature = [Feature1; Feature2; Feature3; Feature4];

data = [feature [1; 2; 1; 2]]; % Tích hợp đặc trưng với nhãn lớp

classificationLearner; % Mở ứng dụng Classification Learner

**MÔ TẢ:**

**Mục đích**: Sử dụng ứng dụng MATLAB **Classification Learner** để thực hiện phân loại dựa trên các đặc trưng đầu vào và nhãn lớp.

**Ý nghĩa các lệnh**:

* feature: Ma trận chứa các đặc trưng đầu vào.
* data: Ma trận tích hợp gồm đặc trưng (feature) và nhãn lớp tương ứng (1 hoặc 2).
* classificationLearner: Lệnh mở ứng dụng MATLAB Classification Learner, nơi bạn có thể:
  + Import dữ liệu (data) để huấn luyện mô hình.
  + Chọn thuật toán phân loại (SVM, k-NN, Decision Trees, v.v.).
  + Đánh giá hiệu suất phân loại.

**KẾT QUẢ:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**CODE:**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Section 12.7.5

% START\_Classification.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Notes: --------- BP label

% Label value: 1 = Norm, 2 = HT

%% Set folder path

clc; clear; close all;

addpath(genpath(pwd));

path.data = './DATA/';

path.output = './OUTPUT/';

%% Load classification dataset

dataset = load([path.data, 'Norm\_vs\_HT.csv']);

target\_Y = dataset(:, 1);

feature\_X = zscore(dataset(:, 2:end));

%% Statistics and Data Visualization for dataset

[row, column] = size(dataset);

count.sample = row;

count.feature = column - 1;

count.Norm = sum(target\_Y == 1);

count.HT = sum(target\_Y == 2);

figure(1);

pie3([count.Norm, count.HT], {'NormBP', 'HT BP'});

legend(num2str(count.Norm), num2str(count.HT));

%% Feature Selection : ReliefF Method

[relieff\_fList, relieff\_weight] = relieff(feature\_X, target\_Y, 10, 'method', 'classification');

figure(2);

bar(relieff\_weight);

xlabel('featureindex');

ylabel('feature weight');

%% Adopt the TopN valuable features to classify

TopN = floor(0.3 \* count.feature);

opt\_feature\_X = feature\_X(:, relieff\_fList(1:TopN));

%% 10-fold Cross-validation

Kfold = 10;

indices = crossvalind('Kfold', target\_Y, Kfold); % 10

sum\_Acc = 0; sum\_Sen = 0; sum\_Spe = 0; sum\_Pre = 0; sum\_Recall = 0; sum\_F1 = 0;

for num\_CV = 1:Kfold

test = (indices == num\_CV);

train = ~test;

%% Divide the dataset into trainset (90%) and testset (10%)

train\_X = opt\_feature\_X(train, :);

train\_Y = target\_Y(train, :);

test\_X = opt\_feature\_X(test, :);

test\_Y = target\_Y(test, :);

%% Training Model

classifier\_type = 'KNN';

switch classifier\_type

case 'LDA' % Classifier 1: Linear Discriminant Analysis (LDA)

trainedModel = fitcdiscr(train\_X, train\_Y);

case 'KNN' % Classifier 2: k-Nearest Neighbor

trainedModel = fitcknn(train\_X, train\_Y);

case 'SVM' % Classifier 3: Support Vector Machine (SVM)

trainedModel = fitcsvm(train\_X, train\_Y);

case 'Tree' % Classifier 4: Decision Tree

trainedModel = fitctree(train\_X, train\_Y);

end

%% Testing Model

[predicted\_Y] = predict(trainedModel, test\_X);

%% Performance Evaluation

[stats] = classification\_evaluation(test\_Y, predicted\_Y, 2); % In Section 12.7.4

sum\_Acc = sum\_Acc + stats.Acc;

sum\_Sen = sum\_Sen + stats.Se;

sum\_Spe = sum\_Spe + stats.Sp;

sum\_Pre = sum\_Pre + stats.P;

sum\_Recall = sum\_Recall + stats.P;

sum\_F1 = sum\_F1 + stats.F1;

end

%% Averaged Performance Result

mean\_Acc = sum\_Acc / Kfold;

mean\_Sen = sum\_Sen / Kfold;

mean\_Spe = sum\_Spe / Kfold;

mean\_Pre = sum\_Pre / Kfold;

mean\_Recall = sum\_Recall / Kfold;

mean\_F1 = sum\_F1 / Kfold;

head\_name = {'Kfold', 'Norm', 'HT', 'Acc', 'Sen', 'Spe', 'Pre', 'Recall', 'F1'};

T = table(Kfold, count.Norm, count.HT, mean\_Acc, mean\_Sen, mean\_Spe, mean\_Pre, mean\_Recall, mean\_F1, ...

'VariableNames', head\_name, 'RowNames', {classifier\_type});

writetable(T, [path.output 'classification\_result.csv'], 'WriteRowNames', true);

**MÔ TẢ:**

**Phần khởi tạo (Lines 10–13):**

* Xóa các biến, đóng tất cả các đồ thị, và thêm đường dẫn chứa dữ liệu cũng như thư mục đầu ra.
* path.data: Đường dẫn đến dữ liệu cần xử lý.
* path.output: Đường dẫn lưu kết quả.

**Tải dữ liệu và chuẩn hóa (Lines 16–18):**

* Norm\_vs\_HT.csv: Dữ liệu chứa nhãn (target\_Y) và đặc trưng (feature\_X).
* zscore: Chuẩn hóa đặc trưng để có trung bình 0 và phương sai 1.

**Thống kê và trực quan hóa dữ liệu (Lines 21–26):**

* Tính số lượng mẫu, đặc trưng, và số lượng thuộc từng lớp (Norm và HT).
* Dùng biểu đồ 3D (pie3) để minh họa tỷ lệ hai lớp.

**Chọn đặc trưng với ReliefF (Lines 28–30):**

* relieff: Phương pháp đánh giá và xếp hạng các đặc trưng dựa trên trọng số.
* Biểu đồ cột thể hiện độ quan trọng của từng đặc trưng.

**Lựa chọn đặc trưng quan trọng (Lines 33–34):**

* Chỉ chọn TopN đặc trưng quan trọng nhất để giảm số chiều dữ liệu.

**K-fold Cross-validation (Lines 36–75):**

* **Chia dữ liệu**: Dữ liệu được chia thành 10 phần (10-fold cross-validation).
* **Huấn luyện và kiểm tra**:
  + Dựa trên thuật toán đã chọn (KNN, LDA, SVM, Tree).
  + Phân loại bằng fitcdiscr, fitcknn, fitcsvm, fitctree.
* **Đánh giá hiệu suất**:
  + Hàm classification\_evaluation: Trả về các chỉ số như:
    - Acc: Độ chính xác.
    - Sen (Sensitivity): Độ nhạy.
    - Spe (Specificity): Độ đặc hiệu.
    - Pre (Precision): Độ chính xác của dự đoán.
    - F1: F1-Score.

**Kết quả trung bình và lưu kết quả (Lines 77–90):**

* Tính giá trị trung bình của các chỉ số hiệu suất qua 10 lần lặp.
* Lưu kết quả vào file CSV (classification\_result.csv).

**KẾT QUẢ:**

**CODE:**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Section 12.7.6

% START\_Classification\_DeepLearning.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Notes: --------- BP label

% Label value: 1 = Norm, 2 = HT

% Categories: Norm, HT

%% Set folder path

clc; clear; close all;

addpath(genpath(pwd));

path.data = './DATA/';

path.output = './OUTPUT/';

%% Load image classification dataset into imageDatastore

parentDir = path.data;

dataDir = 'data';

allImages = imageDatastore(fullfile(parentDir, dataDir), 'IncludeSubfolders', true, 'LabelSource', 'foldernames');

%% Statistics and Data Visualization for dataset

count.Norm = sum(str2num(char(allImages.Labels)) == 1);

count.HT = sum(str2num(char(allImages.Labels)) == 2);

figure(1);

pie3([count.Norm, count.HT], {'NormBP', 'HT BP'});

legend(num2str(count.Norm), num2str(count.HT));

%% Divide the dataset into trainset (60%), validation (20%) and testset (20%)

[imgsTrain, imgsValidation, imgsTest] = splitEachLabel(allImages, 0.6, 0.2, 0.2, 'randomized');

count.train = numel(imgsTrain.Files);

count.validation = numel(imgsValidation.Files);

count.test = numel(imgsTest.Files);

%% Build Transfer Learning Model using GoogLeNet

% Notice: GoogLeNet requires RGB images of size 224-by-224-by-3

net = googlenet;

% Extract the layer graph from the network and plot the layer graph

lgraph = layerGraph(net);

%% Modify GoogLeNet Network Parameters

lgraph = removeLayers(lgraph, {'pool5-drop\_7x7\_s1', 'loss3–classifier', 'prob', 'output'});

numClasses = numel(categories(imgsTrain.Labels));

newLayers = [

dropoutLayer(0.6, 'Name', 'newDropout')

fullyConnectedLayer(numClasses, 'Name', 'fc', 'WeightLearnRateFactor', 5, 'BiasLearnRateFactor', 5)

softmaxLayer('Name', 'softmax')

classificationLayer('Name', 'classoutput')

];

lgraph = addLayers(lgraph, newLayers);

lgraph = connectLayers(lgraph, 'pool5–7x7\_s1', 'newDropout');

%% Training Options Setting

options = trainingOptions('sgdm', ...

'MiniBatchSize', 64, ...

'MaxEpochs', 3, ...

'InitialLearnRate', 1e-4, ...

'ValidationData', imgsValidation, ...

'ValidationFrequency', 10, ...

'ValidationPatience', Inf, ...

'Verbose', 1, ...

'ExecutionEnvironment', 'cpu', ...

'Plots', 'training-progress');

%% Training Model

trainedGN = trainNetwork(imgsTrain, lgraph, options);

%% Testing Model

[YPred, probs] = classify(trainedGN, imgsTest);

%% Performance Evaluation

[stats] = classification\_evaluation(str2num(char(imgsTest.Labels)), str2num(char(YPred)), 2);

%% Export Classification Results

head\_name = {'Norm', 'HT', 'train', 'validation', 'test', 'Acc', 'Sen', 'Spe', 'Pre', 'Recall', 'F1'};

T = table(count.Norm, count.HT, count.train, count.validation, count.test, ...

stats.Acc, stats.Se, stats.Sp, stats.P, stats.Recall, stats.F1, ...

'VariableNames', head\_name, 'RowNames', {'GoogLeNet'});

writetable(T, [path.output 'classification\_result.csv'], 'WriteRowNames', true);

**MÔ TẢ:**

* **Thiết lập đường dẫn thư mục**: Đặt các thư mục cho dữ liệu và đầu ra.
* **Tải bộ dữ liệu hình ảnh**: Sử dụng imageDatastore để tải bộ dữ liệu hình ảnh, phân loại hình ảnh theo tên thư mục (nhãn Norm và HT).
* **Thống kê và trực quan hóa dữ liệu**: Đếm số lượng hình ảnh trong các lớp Norm và HT, sau đó hiển thị biểu đồ phân bố của các lớp.
* **Chia bộ dữ liệu thành 3 tập con**: Chia bộ dữ liệu thành 60% tập huấn luyện, 20% tập xác nhận và 20% tập kiểm tra.
* **Xây dựng mô hình học sâu với GoogLeNet**: Sử dụng mô hình chuyển giao học sâu GoogLeNet, loại bỏ một số lớp không cần thiết và thêm các lớp mới cho phân loại.
* **Cài đặt các tùy chọn huấn luyện**: Đặt các tham số huấn luyện như kích thước lô, số epoch, tốc độ học ban đầu và dữ liệu xác nhận.
* **Huấn luyện mô hình**: Sử dụng trainNetwork để huấn luyện mô hình trên bộ dữ liệu huấn luyện.
* **Dự đoán với mô hình**: Dự đoán nhãn cho bộ dữ liệu kiểm tra sử dụng mô hình đã huấn luyện.
* **Đánh giá hiệu suất**: Tính toán các chỉ số đánh giá như độ chính xác (Accuracy), độ nhạy (Sensitivity), độ đặc hiệu (Specificity), độ chính xác (Precision), độ nhớ (Recall) và điểm F1.
* **Lưu kết quả**: Lưu kết quả phân loại và các chỉ số đánh giá vào file CSV.

**KẾT QUẢ:**

**CODE:**

%% Example 2

% Create features.

t = zeros(150,2);

y = zeros(150,2);

t(1:75,1) = 1;

t(76:150,2) = 1;

y(1:44,1) = 1;

y(44:75,2) =1;

y(76:90,1) =1;

y(90:150,2) =1;

% t is the ONE-HOT true labels matrix, for binary classes 150 samples

% y is the ONE-HOT classifier predictions, for binary classes 150 samples

% Using confusionmat. Confusionmat needs label VECTORS, and not One-hot

% matrices. First convert the one-hot labels to label vector. Since we have

% a binary classifier, let us use the first column as the label vector

conmat\_1 = confusionmat(t(:,1),y(:,1));

disp(conmat\_1)

% To plot a fancy looking confusion matrix like the one in the Tutorial,

% use plotconfusion.

% plotconfusion also uses label vectors, but requires each column to be one sample

% , so we transpose the column vector into a row vector

figure;

plotconfusion(t(:,1)',y(:,1)');

**MÔ TẢ:**

**Khởi tạo dữ liệu**:

* t: Ma trận nhãn thực sự (One-Hot), có kích thước 150x2. Nhãn 1 cho lớp đầu tiên trong 75 mẫu đầu tiên, và 1 cho lớp thứ hai từ 76 đến 150.
* y: Ma trận nhãn dự đoán của bộ phân loại (One-Hot), tương tự như t, với các giá trị được điền vào các phần tử để thể hiện kết quả dự đoán.

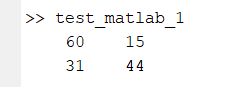
**Sử dụng confusionmat**:

* confusionmat(t(:,1),y(:,1)): Tạo ma trận nhầm lẫn bằng cách so sánh nhãn thực tế (t) với nhãn dự đoán (y). Vì đây là phân loại nhị phân, ta sử dụng cột đầu tiên trong t và y để so sánh.
* disp(conmat\_1): Hiển thị ma trận nhầm lẫn trên màn hình.

**Hiển thị ma trận nhầm lẫn với plotconfusion**:

* plotconfusion(t(:,1)', y(:,1)'): Vẽ đồ thị ma trận nhầm lẫn. Do plotconfusion yêu cầu mỗi cột là một mẫu, nên ta phải chuyển các cột thành hàng (transpose).

**KẾT QUẢ:**

****

**A screenshot of a computer

Description automatically generated**

**CODE:**

clear

% This function calculates the sensitivity and specificity

% and other evaluation measures

function [stats] = classification\_evaluation(TARGET, PREDICTED, positive\_label\_value)

% Input :

% −−−−−−−

% TARGET = Column matrix with TARGET class labels of the training

% examples

% PREDICTED = Column matrix with predicted class labels by the

% classification model

% Output :

% −−−−−−−−

% stats is a structure array

% stats.confusionMat

% Predicted Classes

% p ' n '

% \_\_\_| \_\_\_\_\_\_ | \_\_\_\_\_ |

% Actual p | | |

% Classes n | | |

%

% Four important parameters :

% −−−−−−−−−−−−−−−−−−−−−−−−−−

% TP: true positive

% TN: true negative

% FP: false positive

% FN: false negative

% −−−−−−−−−−−−−−−−−−−−

% Calculating different evaluation measures

% −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

% stats.accuracy = (TP + TN) / (TP + FP + FN + TN)

% stats.precision = TP / (TP + FP)

% stats.sensitivity = TP / (TP + FN)

% stats.specificity = TN / (FP + TN)

% stats.recall = sensitivity

% stats.Fscore = 2 \* TP / (2 \* TP + FP + FN)

index = (TARGET == positive\_label\_value); % index of data with positive class

num\_positive = length(TARGET(index));

num\_negative = length(TARGET(~index));

N = num\_positive + num\_negative;

TP = sum(TARGET(index) == PREDICTED(index));

TN = sum(TARGET(~index) == PREDICTED(~index));

FP = num\_negative - TN;

FN = num\_positive - TP;

TPR = TP / (TP + FN);

FNR = FN / (TP + FN);

FPR = FP / (FP + TN);

TNR = TN / (TN + FP);

accuracy = (TP + TN) / (TP + FN + FP + TN);

sensitivity = TPR;

specificity = TNR;

precision = TP / (TP + FP);

recall = sensitivity;

F1 = 2 \* ((precision \* recall) / (precision + recall));

stats.positive\_label = positive\_label\_value;

stats.TP = TP;

stats.FP = FP;

stats.FN = FN;

stats.TN = TN;

stats.Acc = accuracy;

stats.Se = sensitivity;

stats.Sp = specificity;

stats.P = precision;

stats.Recall = recall;

stats.F1 = F1;

stats.All = [positive\_label\_value accuracy TP FP FN TN TPR FNR FPR TNR precision sensitivity specificity F1];

% Display the results

disp('Classification Evaluation Results:');

disp(['Accuracy: ', num2str(stats.Acc)]);

disp(['Sensitivity: ', num2str(stats.Se)]);

disp(['Specificity: ', num2str(stats.Sp)]);

disp(['Precision: ', num2str(stats.P)]);

disp(['Recall: ', num2str(stats.Recall)]);

disp(['F1 Score: ', num2str(stats.F1)]);

end

% Sample data (for demonstration purposes)

TARGET = [1; 1; 2; 2; 1; 2; 1; 2; 1; 2]; % Example true class labels (1 or 2)

PREDICTED = [1; 1; 1; 2; 1; 2; 2; 2; 1; 2]; % Example predicted class labels

positive\_label\_value = 2; % Define which class is considered positive (e.g., class 2)

% Call the function with these inputs

stats = classification\_evaluation(TARGET, PREDICTED, positive\_label\_value);

% Display the results

disp(stats);

**MÔ TẢ:**

**Đầu vào**:

* TARGET: Một ma trận cột chứa nhãn lớp của các mẫu huấn luyện (thực tế).
* PREDICTED: Một ma trận cột chứa nhãn lớp được phân loại bởi mô hình (dự đoán).
* positive\_label\_value: Giá trị đại diện cho lớp dương tính (ví dụ: 1).

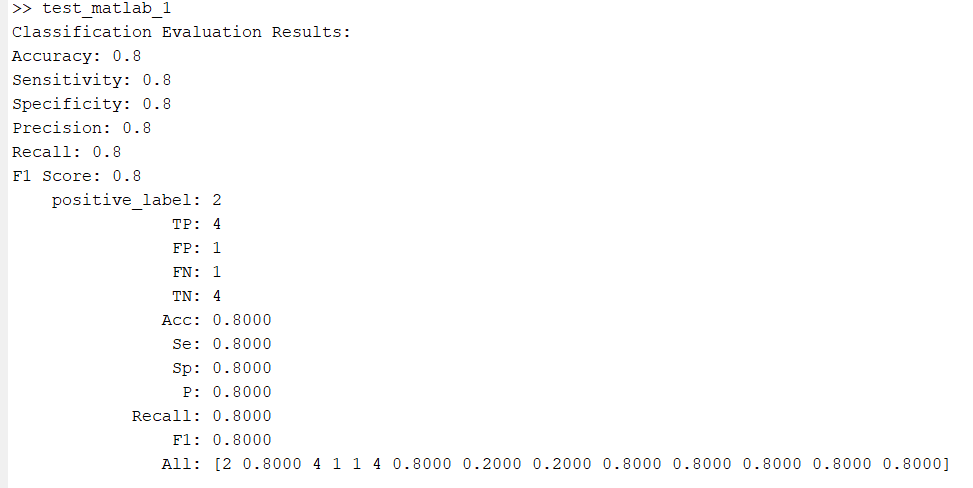
**Các phép toán**:

* Tính toán số lượng mẫu dương tính và âm tính từ TARGET.
* Sử dụng các chỉ số như **TP (True Positive)**, **TN (True Negative)**, **FP (False Positive)** và **FN (False Negative)** để tính các chỉ số đánh giá:
  + **Accuracy**: Độ chính xác của mô hình.
  + **Precision**: Độ chính xác của dự đoán.
  + **Sensitivity (Recall)**: Độ nhạy, khả năng phát hiện lớp dương tính.
  + **Specificity**: Độ đặc hiệu, khả năng phát hiện lớp âm tính.
  + **F1 Score**: Đo lường sự cân bằng giữa Precision và Recall.

**Đầu ra**:

* Một cấu trúc stats chứa tất cả các chỉ số đánh giá được tính toán, bao gồm các chỉ số như accuracy, sensitivity, specificity, precision, recall và F1 score.
* Thông tin về số lượng các giá trị TP, TN, FP, FN, và TPR (True Positive Rate) cũng được lưu trong cấu trúc stats.

**KẾT QUẢ:**

****