2nd step Create feedforward NN and train, test, data set

When I created the merged data set files in step 01 and next step need to be a crated feedforward NN and train data set and testing dataset and get their training and testing accuracy.

First need to load the meagre data file before creating that file in 1st step. Then doing normalization dataset and need defined the neural network and hidden layer size, and feedforward NN like Levenberg-Marquardt as the training algorithm. Then Divide data into training, validation, and test sets and Set training parameters. After training the neural network needs to check intra and inter variance value and calculate the training features and sample. When I used the loop, I checked the variance values and displayed the variance results. Then get the tarin accuracy test accuracy and finally save the result in trained\_results.mat file.

The code is given below.

% Load the merged dataset

load('mergedData.mat');

% The merged data should be in the variable `merged\_data`.

% Separate features and targets(labels)

% Assuming the last column is the user identifier (target/labels)

% And other all columns correspond to the Features

inputs = merged\_data(:, 1:end-1)'; % Features (transpose to match NN input format)

targets = merged\_data(:, end)'; % User identifiers(targets/labels) (transpose for NN format)

% Normalize the input data (optional, improves NN performance)

inputs = normalize(inputs, 'range'); % Define Normalize to [0, 1] range

% Create and configure the neural network

hiddenLayerSize = [20,10]; % Define the size of the hidden layer

net = feedforwardnet(hiddenLayerSize, 'trainlm'); % Feedforward NN with Levenberg-Marquardt as the training algorithm

% Divide data into training, validation, and test sets

net.divideParam.trainRatio = 0.7; % 70% training

net.divideParam.valRatio = 0.15; % 15% validation

net.divideParam.testRatio = 0.15; % 15% testing

% Set training parameters (optional, can tweak to improve performance)

net.trainParam.epochs = 1000; % Maximum number of training iterations

net.trainParam.goal = 1e-6; % Performance goal (MSE)

net.trainParam.min\_grad = 1e-7; % Minimum gradient

% Train the neural network

[net, tr] = train(net, inputs, targets);

% Evaluate intra-variance and inter-variance for the training set

% Extract training set inputs and targets

train\_inputs = inputs(:, tr.trainInd); % Features of training set

train\_targets = targets(tr.trainInd); % Targets of training set

% Get unique classes in the training set

unique\_classes = unique(train\_targets);

% Initialize intra-variance and inter-variance

intra\_variance = 0;

inter\_variance = 0;

% Calculate overall mean of the training features

overall\_mean = mean(train\_inputs, 2); % Overall mean across all features (column-wise)

% Total number of samples in the training set

total\_samples = size(train\_inputs, 2);

% use loop calculate variance metrics for each class

for i = 1:length(unique\_classes)

% Find the indices of samples belonging to the current class

class\_idx = find(train\_targets == unique\_classes(i));

% Extract features of the current class

class\_samples = train\_inputs(:, class\_idx);

% Number of samples in this class

num\_samples\_in\_class = size(class\_samples, 2);

% Calculate the mean of the current class

class\_mean = mean(class\_samples, 2); % Mean across features for this class

% Compute intra-variance (variance within the class)

intra\_variance = intra\_variance + ...

sum(sum((class\_samples - class\_mean).^2)) / total\_samples;

% Compute inter-variance (variance of class means relative to the overall mean)

inter\_variance = inter\_variance + ...

num\_samples\_in\_class \* sum((class\_mean - overall\_mean).^2) / total\_samples;

end

% Display inter and intra variance results

fprintf('Intra-variance (Training Set): %.4f\n', intra\_variance);

fprintf('Inter-variance (Training Set): %.4f\n', inter\_variance);

% Evaluate training accuracy for the training set

train\_outputs = net(inputs(:, tr.trainInd)); % Network predictions for training set

train\_predicted\_classes = round(train\_outputs); % Round predictions to nearest integer

train\_actual\_classes = targets(tr.trainInd); % Actual training targets

% Calculate training accuracy

train\_correct\_predictions = sum(train\_predicted\_classes == train\_actual\_classes);

train\_total\_samples = length(train\_actual\_classes);

train\_accuracy = (train\_correct\_predictions / train\_total\_samples) \* 100;

fprintf('Training Accuracy: %.2f%%\n', train\_accuracy);

% Evaluate test accuracy

test\_inputs = inputs(:, tr.testInd); % Inputs for the test set

test\_targets = targets(tr.testInd); % Targets for the test set

% Generate predictions for the test set

test\_outputs = net(test\_inputs);

test\_predicted\_classes = round(test\_outputs);

test\_actual\_classes = test\_targets;

% Ensure classes are positive integers starting from 1

% Shift classes to be 1-based if they start at 0 or contain negative values

min\_class = min([test\_actual\_classes, test\_predicted\_classes]);

if min\_class <= 0

test\_actual\_classes = test\_actual\_classes - min\_class + 1;

test\_predicted\_classes = test\_predicted\_classes - min\_class + 1;

end

% Calculate test accuracy

test\_correct\_predictions = sum(test\_predicted\_classes == test\_actual\_classes);

test\_total\_samples = length(test\_actual\_classes);

test\_accuracy = (test\_correct\_predictions / test\_total\_samples) \* 100;

fprintf('Test Set Accuracy: %.2f%%\n', test\_accuracy);

% Plot performance and confusion matrix

figure;

plotperform(tr); % Performance plot (training, validation, test errors)

% One-hot encode the test targets and predicted classes for confusion matrix

num\_classes = max([test\_actual\_classes, test\_predicted\_classes]); % Determine the number of unique classes

test\_actual\_classes\_onehot = ind2vec(test\_actual\_classes, num\_classes); % One-hot encode actual classes

test\_predicted\_classes\_onehot = ind2vec(test\_predicted\_classes, num\_classes); % One-hot encode predicted classes

% Plot confusion matrix using one-hot encoded labels

figure;

plotconfusion(test\_actual\_classes\_onehot, test\_predicted\_classes\_onehot); % Confusion matrix

% Save the trained results and model

results.train\_accuracy = train\_accuracy; % Training accuracy

results.test\_accuracy = test\_accuracy; % Test accuracy

results.intra\_variance = intra\_variance; % Intra-variance

results.inter\_variance = inter\_variance; % Inter-variance

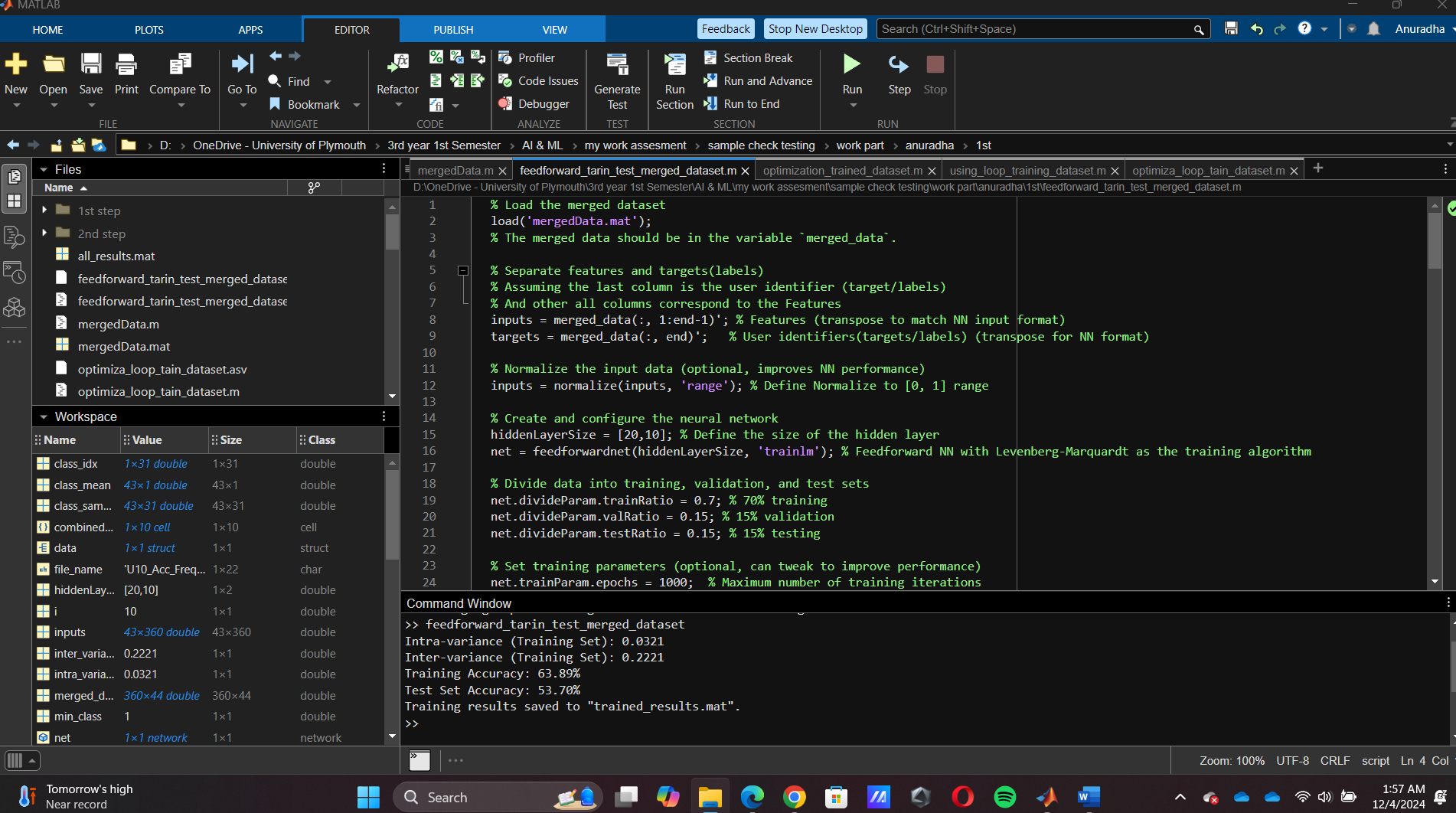
results.net = net; % Trained neural network

results.training\_record = tr; % Training record (training, validation, test performance)

save('trained\_results.mat', 'results'); % Save results

fprintf('Training results saved to "trained\_results.mat".\n');

The code will shows the that result in command window



In that code it run while it gives tarin result like the popup and it provide confusion matrix graph and NN charts.

A screenshot of a computer

Description automatically generated

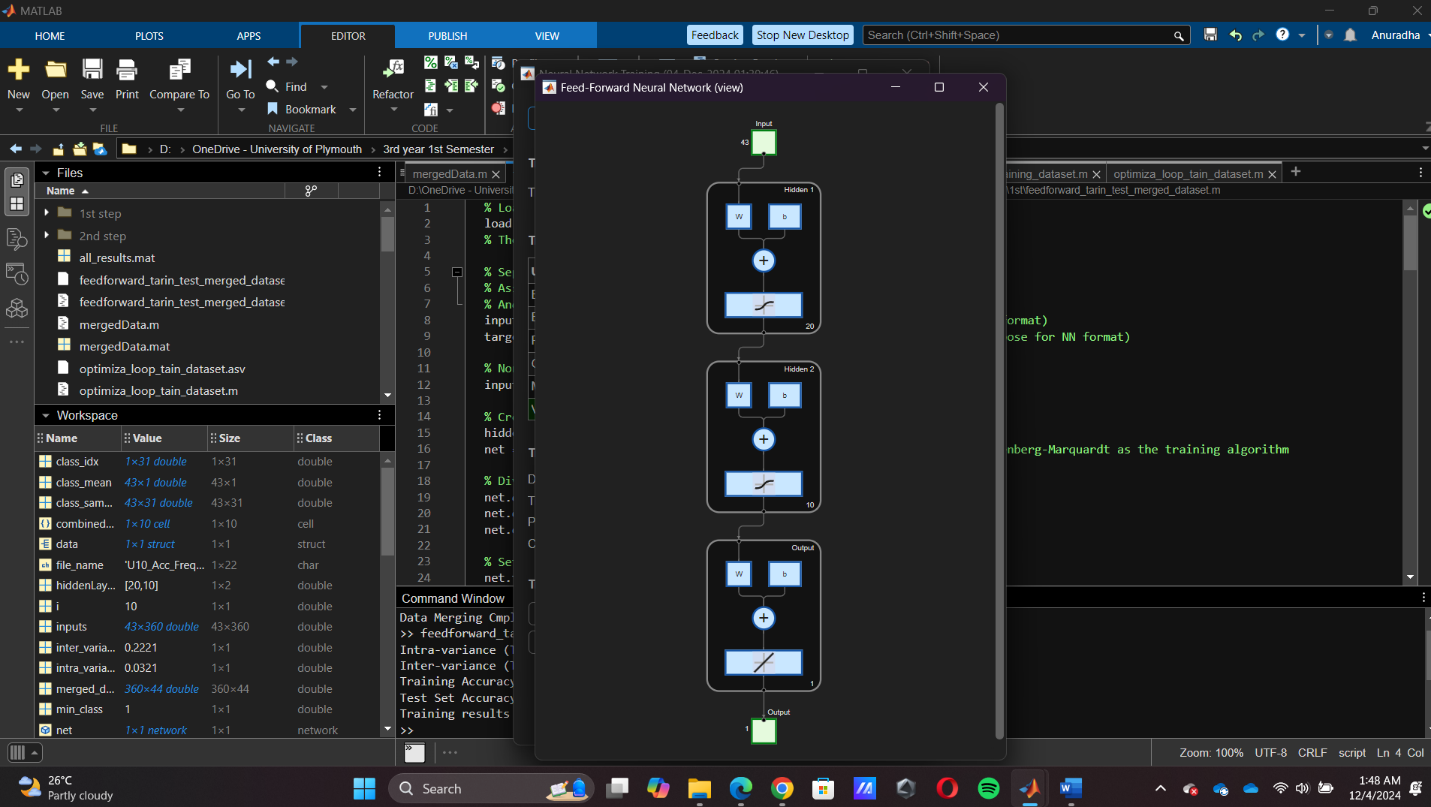
A computer screen shot of a graph

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

When I used 2 hidden layers and defined 20 and 10 neurons in feedforward neural networks

