3rd step Optimization before trained data set

The I need the optimization before I trained data set and first I load the mergedData.mat (it create in 1st step) file and load trained\_results.mat (it already create in 2nd step) then find redundant features remove the that redundant features using for loop and apply PCA method to the data set. Then normalize redundant features and trained the NN with PCA method and get validity and test accuracy. Then calculate and save optimization accuracy and result in a .mat file.

The code has below.

% Load the trained dataset and per-tarined model

load('mergedData.mat'); % Load the dataset

load('trained\_results.mat'); % Load pre-trained model

% Separate features and targets from the loaded data

features = merged\_data(:, 1:end-1); % Features

targets = merged\_data(:, end); % Targets/labels (user identifiers)

% Handle multicollinearity (optional)

corr\_matrix = corrcoef(features); % Correlation matrix

% Find indices of redundant features

redundant\_columns = [];

for i = 1:size(corr\_matrix, 1)

for j = i+1:size(corr\_matrix, 2)

if abs(corr\_matrix(i, j)) > 0.99

redundant\_columns = [redundant\_columns, j]; % Collect the column indices

end

end

end

redundant\_columns = unique(redundant\_columns); % Ensure unique column indices

% Remove redundant features

if ~isempty(redundant\_columns)

fprintf('Removing %d redundant features due to high correlation.\n', length(redundant\_columns));

features(:, redundant\_columns) = [];

end

% Apply PCA to the dataset

[coeff, pca\_features, ~, ~, explained] = pca(features); % Perform PCA

% Retain enough components to explain 95% variance

explained\_variance\_threshold = 95;

cumulative\_explained = cumsum(explained);

num\_components = find(cumulative\_explained >= explained\_variance\_threshold, 1);

% Select reduced features

optimized\_features = pca\_features(:, 1:num\_components);

% Normalize the reduced features

optimized\_features = normalize(optimized\_features, 'range');

% Transpose data for the neural network input format

inputs = optimized\_features'; % Transpose features

targets = targets'; % Transpose targets

% Retrain the Neural Network with Reduced Features

hiddenLayerSize = [20, 10]; % Hidden layer configuration

net = feedforwardnet(hiddenLayerSize, 'trainlm'); % Create a new NN

% 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

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

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

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

% Train the network with PCA-reduced inputs

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

% Evaluate Validation Accuracy

val\_inputs = inputs(:, tr.valInd); % Validation inputs

val\_targets = targets(tr.valInd); % Validation targets

val\_outputs = net(val\_inputs); % Get predictions

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

val\_actual\_classes = val\_targets;% Actual validation labels/targets

% Calculate validation accuracy

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

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

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

% Evaluate Optimization (Test) Accuracy

test\_inputs = inputs(:, tr.testInd); % Test inputs

test\_targets = targets(tr.testInd); % Test targets

test\_outputs = net(test\_inputs); % Get predictions

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

test\_actual\_classes = test\_targets;

% Ensure all class labels are positive integers starting from 1

min\_class = min([test\_actual\_classes, test\_predicted\_classes]); % Get the minimum class value

if min\_class <= 0

% Adjust the class labels to make them positive integers starting from 1

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

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

end

% Calculate optimization 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;

% Save the Optimized Model and Results

results.val\_accuracy = val\_accuracy; % Validation accuracy

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

results.num\_components = num\_components; % Number of PCA components used

results.net = net; % Trained NN with PCA

save('optimized\_trained\_results.mat', 'results'); % Save the results

% Print Accuracy Results

fprintf('Validation Accuracy After PCA: %.2f%%\n', val\_accuracy);

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

% Visualize PCA Results and NN Performance

figure;

pareto(explained); % Plot variance explained by each principal component

title('PCA Explained Variance');

figure;

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

% One-hot encode targets and predictions for confusion matrix

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

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

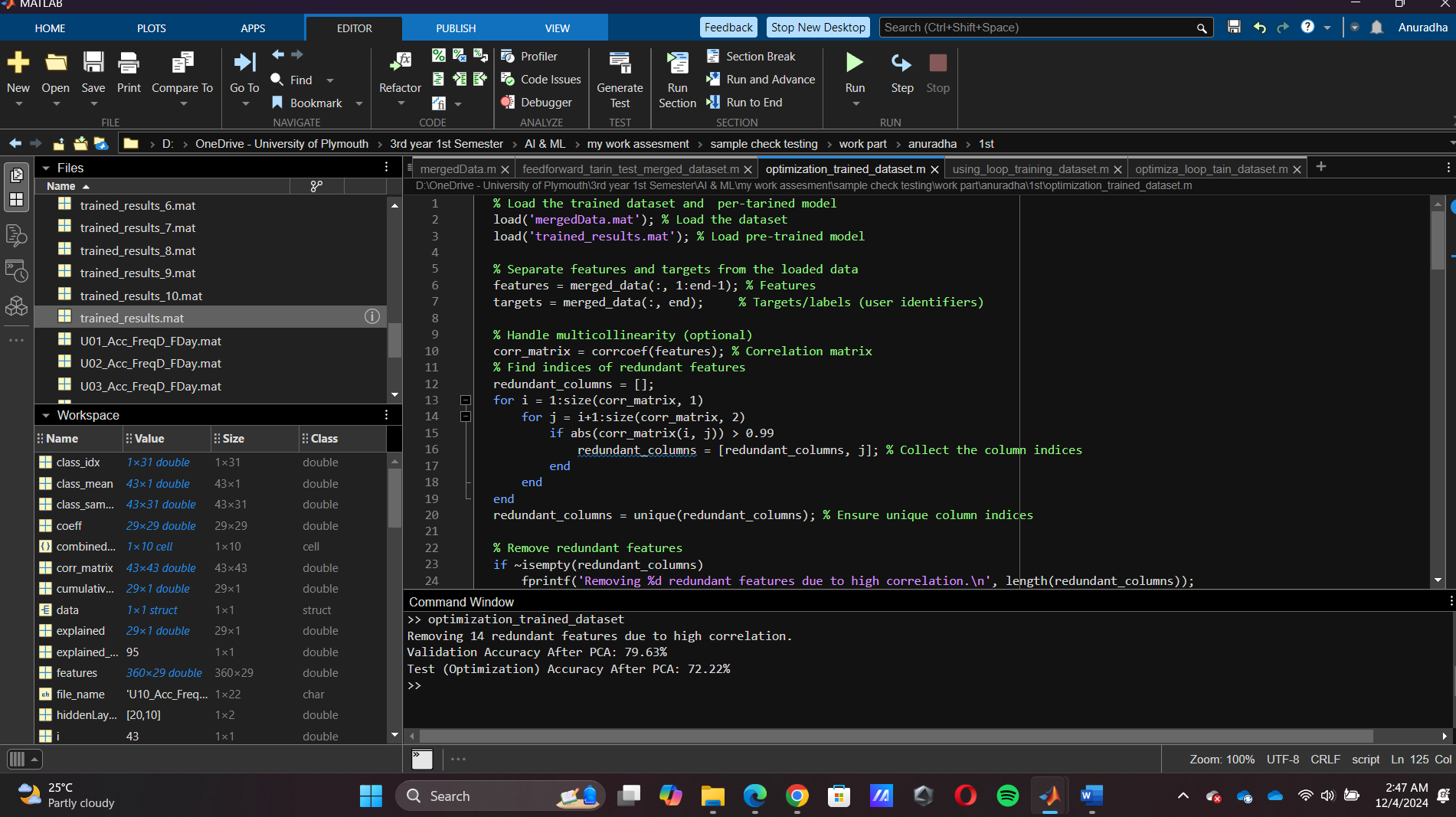
test\_predicted\_onehot = ind2vec(test\_predicted\_classes, num\_classes); % One-hot encode predicted classes

% Plot confusion matrix with one-hot encoding

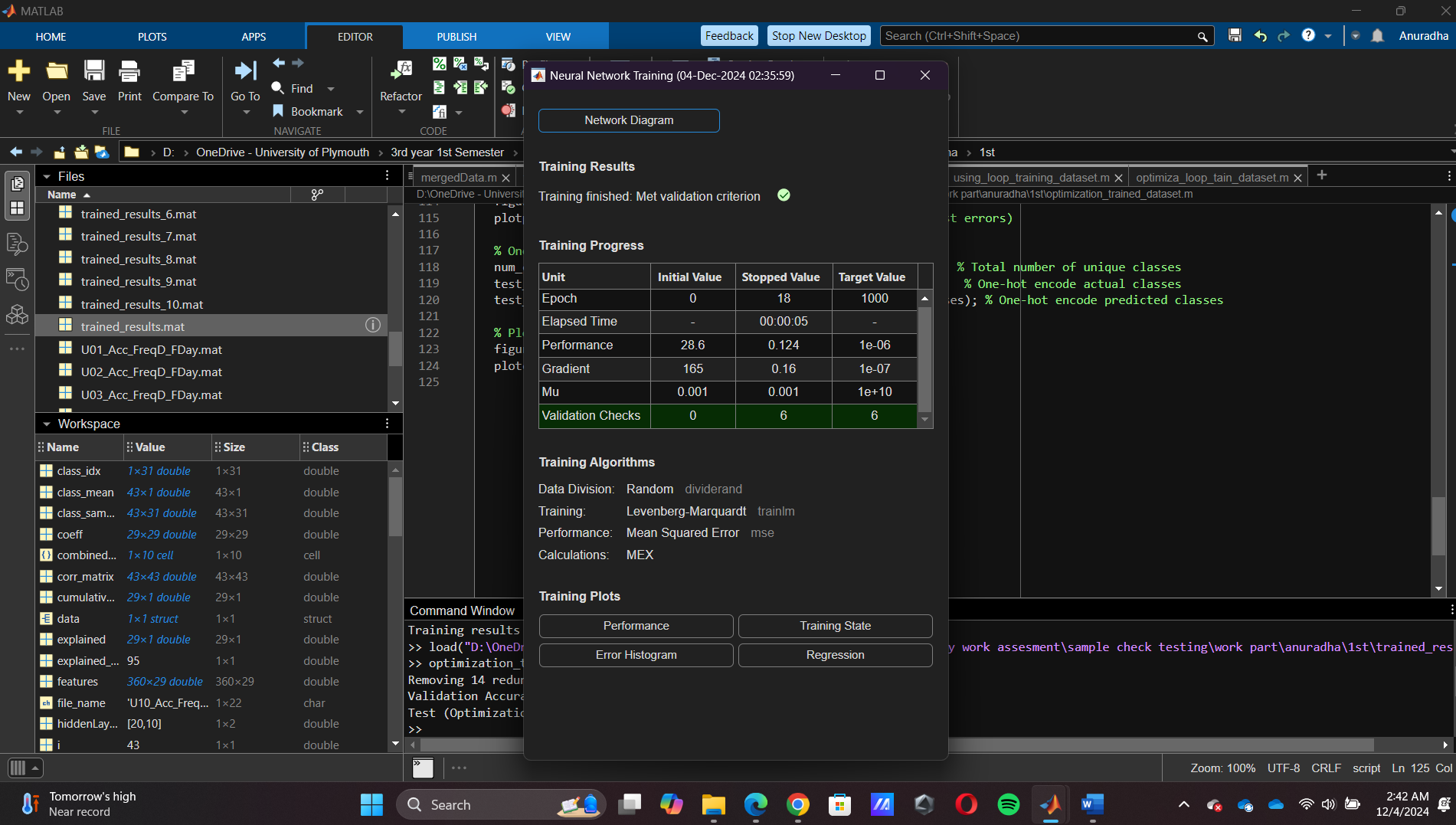
figure;

plotconfusion(test\_actual\_onehot, test\_predicted\_onehot);

run that code command window showing below result



And give the many result and I defined the 2 hidden layers and 20 and 10 neurons in that NN module data set



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