When I use same code And PCA method it shows some warning in command window. But it is not a hindrance to obtaining optimization accuracy values. Because that warning mentions PCA is detected, that dataset have the some features are highly collinear or even linearly dependent. This means some columns are linear combinations of others, and PCA automatically handles this by discarding redundant components.

Code and command window given below.

% Load the dataset

load('mergedData.mat');

% Separate features and targets

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

targets = merged\_data(:, end); % Targets

% Handle multicollinearity by removing highly correlated features

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

% Find indices of redundant features (absolute correlation > 0.99)

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); % Remove duplicate indices

% Remove redundant features

if ~isempty(redundant\_columns)

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

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

end

% Apply PCA to reduce dimensions

[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 neural network

inputs = optimized\_features'; % Transpose features

targets = targets'; % Transpose targets

% Initialize arrays to store validation and test accuracy for multiple iterations

val\_accuracies = zeros(10, 1);

test\_accuracies = zeros(10, 1);

all\_results = cell(10, 1); % Store results for each iteration

% Loop for training and testing 10 times

for iteration = 1:10

fprintf('Iteration %d of 10\n', iteration);

% Create and configure the neural network

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; % Max epochs

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

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;

% 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;

val\_accuracies(iteration) = val\_accuracy; % Store validation accuracy

% Evaluate 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;

% 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;

test\_accuracies(iteration) = test\_accuracy; % Store test accuracy

% Store the results for each iteration

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

results.training\_record = tr; % Training record

all\_results{iteration} = results;

% Save the results for the current iteration

save(sprintf('optimized\_trained\_results\_%d.mat', iteration), 'results');

fprintf('Iteration %d: Validation Accuracy = %.2f%%, Test Accuracy = %.2f%%\n', iteration, val\_accuracy, test\_accuracy);

end

% Save all results

save('optimized\_all\_results.mat', 'val\_accuracies', 'test\_accuracies', 'all\_results');

% Display summary results

fprintf('Average Validation Accuracy: %.2f%%\n', mean(val\_accuracies));

fprintf('Average Test Accuracy: %.2f%%\n', mean(test\_accuracies));

% Visualize PCA Explained Variance

figure;

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

title('PCA Explained Variance');

