```
%{
This file is the same as nn.m except for that the test data is
 manually
taken from the array before training is done, then is given to the
 trained
model to test it manually.
%}


function []=nn_manual(inputLabelledData, hiddenLayerSize,
 testDataPercentage)

    % Define number of neurons for the hidden layer of the NN
    hiddenLayerSize = hiddenLayerSize;

    % Create a Pattern Recognition Network with the defined number of
 hidden layers.
    % `patternnet` is specific for pattern-recognition NNs
    net = patternnet(hiddenLayerSize);
    %{
    patternnet() is specialized for pattern recognition problems.
    - Default training algo: Scaled conjugate gradient backpropagation
 (trainscg).
        * trainscg's goal: minimize a cost function.
    - Default loss cost function: Cross-entropy.
        * This function measures the performance of a classification
 model whose
        output varies between 0 and 1.
        * Cross-entropy loss increases as the prediction probability
 diverges
        from the output value.
        * Therefore, small values -> good performance, large values ->
 bad performance.
    %}

    % Set up Division of Data for Training, Validation, Testing
 Subsets
    net.divideParam.trainRatio = 85/100;
    net.divideParam.valRatio = 15/100;
    test_percent = 20;

    % before splitting into inputs and targets shuffle the rows
    random_final_labelled_data =
 inputLabelledData(randperm(size(inputLabelledData, 1)), :);

    % set some percentage of it aside for testing
    test_element_count =
 uint32((test_percent/100)*length(random_final_labelled_data));

    % Define which features to include in the input set.
```

```
    train_inputs = random_final_labelled_data(1:end-
test_element_count,1:end-5)';    % Take all the rows, and all the 10
 features as inputs. Could also use: inputs = dataSet(:,1:end-2).
    test_inputs= random_final_labelled_data(end-test_element_count
+1:end,1:end-5)';

    % Define the target set
    train_targets = random_final_labelled_data(1:end-
test_element_count, end-4:end)';
    test_targets = random_final_labelled_data(end-test_element_count
+1:end, end-4:end)';

    % Standardise and normalise the input data.
    % normalize() normalises the data such that the center is 0 and
 the
    % standard deviation is 1. Function normalises each column by
 default.
    % 'range' makes all the values be between 0 and 1.
    train_inputs = normalize(train_inputs, 'range');
    test_inputs = normalize(test_inputs, 'range');

    % Train the Network
    [net, tr] = train(net, train_inputs, train_targets);

    % ------------------------
    % Test the Network with the test subset from the current dataset
    % ------------------------
    actualTstOutputs = net(test_inputs);

    %  compare the NN's predictions against the training set
    idealTstOutputs = test_targets;
    tstPerform = perform(net, idealTstOutputs, actualTstOutputs);

    sets_for_labels = [{'LGW'} {'RA'} {'RD'} {'SiS'} {'StS'}];

    % we need to convert the targets from Nx5 boolean values into a
 single
    % string row/column to be able to run confusionchart
    for yy=1 : size(idealTstOutputs,2)
        % get the 5 1/0 values representing the class label
        current_ideal_class = idealTstOutputs(:,yy);
        current_actual_class = actualTstOutputs(:,yy);
        % find where the '1' is
        [~,I_ideal] = max(current_ideal_class);
        [~,I_actual] = max(current_actual_class);
        % get the corresponding string value of the class label
        idealTstOutputsSimplified(:,yy) = sets_for_labels(I_ideal);
        actualTstOutputsSimplified(:,yy) = sets_for_labels(I_actual);
    end
    % create the confusion matrix object to show and retrieve
    % classification accuracy from
    plotTitle = sprintf('ANN Confusion Matrix for %i
 features',size(inputLabelledData,2)-5);
```

```
    cm =
confusionchart(idealTstOutputsSimplified,actualTstOutputsSimplified,...
        'Title', plotTitle,...
        'RowSummary', 'absolute',...
        'ColumnSummary', 'absolute');

    %  plot the confusion matrix
    % numFeatures = sprintf('%i
features',size(inputLabelledData,2)-5);
    % hold off
    % plotconfusion(idealTstOutputs,actualTstOutputs,numFeatures);

    % Calculate the classification accuracy from the confusion matrix
    % Need to first obtain the number of correct classifications, this
will
    % be equal to the sum of the values in the diagonal of the CM
    confusionMatrixResults = cm.NormalizedValues;
    correct_predictions = 0;
    for ii=1 : length(confusionMatrixResults)
        correct_predictions = correct_predictions +
confusionMatrixResults(ii,ii);
    end
    accuracy = (correct_predictions/length(test_targets))*100;

    fprintf("\n-------------\nSummary:\n    Test data: %i Percent\n
Hidden layer neurons: %i\n", testDataPercentage, hiddenLayerSize)
    fprintf("    Number of features: %i \n",
size(inputLabelledData,2)-5)
    fprintf("    ANN classification accuracy %f\n", accuracy)
    fprintf('    Patternnet performance: %f \n', tstPerform);
    fprintf('    num_epochs: %d, stop: %s\n-------------\n\n',
tr.num_epochs, tr.stop);

end
```

Published with MATLAB® R2020a