```
%{
This file is the same as nn_manual.m except for that the test data is
extracted using the built-in `net.divideParam.testRatio` tool.
%}


function []=nn(inputLabelledData)

    % Define number of neurons for the hidden layer of the NN
    hiddenLayerSize = 10;

    % 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 = 70/100;
    net.divideParam.valRatio = 15/100;
    net.divideParam.testRatio = 15/100;

    % ---------------------------------------------
    % Split data into inputs and targets for ML
    % ---------------------------------------------
    % Define which features to include in the input set.
    ann_inputs = inputLabelledData(:,1:end-5)';    % Take all the
 rows, and all the 294 features as inputs.

    % Define the target set
    ann_targets = inputLabelledData(:, end-4:end)'; % Take all the
 rows, and the last 5 columns as outputs.

    % 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.
   ann_inputs = normalize(ann_inputs, 'range');

   % Train the Network
   [net, tr] = train(net, ann_inputs, ann_targets);

   % -----------------------
   % Test the Network with the test subset from the current dataset
   % -----------------------
   %  get the indices/positions randomly selected for the test
dataset (15%)
   tInd = tr.testInd;

   %  obtain the NN's predictions of the test data
   tstInputs = ann_inputs(:,tInd);
   tstOutputs = net(tstInputs);

   %  compare the NN's predictions against the training set
   actualTstOutputs = ann_targets(:,tInd);
   tstPerform = perform(net, actualTstOutputs, tstOutputs);

   idealTstOutputs = ann_targets(:,tInd);
   actualTstOutputs = tstOutputs;

   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');

   % 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(idealTstOutputs))*100;

    fprintf("\n-------------\nANN model accuracy using %i features: %f
\n", size(inputLabelledData,2)-5, accuracy)
    fprintf('Patternnet performance using %i features: %f\n',
 size(inputLabelledData,2)-5, tstPerform);
    fprintf('num_epochs: %d, stop: %s\n-------------\n',
 tr.num_epochs, tr.stop);

end
```

Published with MATLAB® R2020a