Multi Layer Perceptron Methodology

This script is used for training the a MLP with the training data to find a set of hyperparameters that will be used to build a network that will best generalise to unseen data. The model:

1. Loads the data
2. Splits data into training and test sets
3. Initilises grid search parameters
4. Initialises a results table
5. Trains and validates multiple networks using a grid search
6. Saves the results to table 'MLP_Results'
7. Retrieves best hyperparameters and retrains model using all training data
8. Saves the trained network as 'bestnet' for testing in 'NC_MLP_TestScript'

1. Loading Data

```
%Load the data for training the network

load('Training.mat')
```

2. Splitting training predictors and targets

```
%Split training predictors and targets

train_x = Training(:, 1:42)';
train_y = Training(:, 43)';
```

3. Neural Net Grid Search Parameters

```
%Define parameters for grid search

hiddenLayerSize = [10, 30, 50];
momentums = [0.01, 0.1 0.5, 0.9];
learningrates = [0.01, 0.1 0.5, 0.9];
```

4. Results table

```
%Initialise results table for storage

mlp_results = {'Structure', 'Momentum', 'Learning_Rate', 'Avg_Acc', 'Avg_Time'};
```

5. Training and Grid search

```
%Build a network using each of the grid search configurations. Carry out
%10-fold cross validation on each combination and save results to table

for i = hiddenLayerSize
    for j = momentums
        for k = learningrates

            list_acc = [];
            list_time = [];

            %print paramaters in run
            fprintf('\nHidden layer structure: %2f. ', i);
            fprintf('Momentum: %2f. ', j);
            fprintf('Learning Rate: %2f. \n', k);
            %fprintf('ValFailures (Early Stopping): %2f. \n', n);

            %10-fold cross validation
            n=size(Training,1); %size of dataset
            c = cvpartition(n,'kfold',10); % 10-folds

            %for each fold
            for m = 1:10

                tic
                train_idx=training(c,m); %training indexes of fold-m
                train_data= Training(train_idx,:); %get data using indexes of fold-m
                x = train_data(:,1:42)'; %fold-m training predictors
                t = train_data(:,43)'; %fold-m training targets

                % Choose a Training Function
                trainFcn = 'traingdx';  %Gradient descent with momentum and adaptive learning rate backp

                % Create a Pattern Recognition Network
                net = patternnet([i, i, i], trainFcn);

                % Choose Input and Output Pre/Post-Processing Functions
                % For a list of all processing functions type: help nnprocess
                net.input.processFcns = {'removeconstantrows','mapminmax'};

                %define parameters for training
                net.trainParam.mc = j; %momentum
                net.trainParam.lr = k; %learning rate
                net.trainParam.max_fail = 15;
                net.trainParam.epochs = 500;

                % Setup Division of Data for Training, Validation, Testing
                % For a list of all data division functions type: help nndivision
                net.divideFcn = 'dividerand';  % Divide data randomly
                net.divideMode = 'sample';  % Divide up every sample
                net.divideParam.trainRatio = 70/100;
                net.divideParam.valRatio = 15/100;
                net.divideParam.testRatio = 15/100;

                % Choose a Performance Function
                % For a list of all performance functions type: help nnperformance
                net.performFcn = 'crossentropy';  % Cross-Entropy

                % Choose Plot Functions
```

```
                % For a list of all plot functions type: help nnplot
                net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
                    'plotconfusion', 'plotroc'};

                % Train the Network
                [net,tr] = train(net,x,t);

                % Test the Network
                y = net(x);
                e = gsubtract(t,y);
                performance = perform(net,t,y);
                tind = vec2ind(t);
                yind = vec2ind(y);
                percentErrors = sum(tind ~= yind)/numel(tind);

                % Recalculate Training, Validation and Test Performance
                trainTargets = t .* tr.trainMask{1};
                valTargets = t .* tr.valMask{1};
                testTargets = t .* tr.testMask{1};
                trainPerformance = perform(net,trainTargets,y);
                valPerformance = perform(net,valTargets,y);
                testPerformance = perform(net,testTargets,y);
                time = toc;

                C = confusionmat(t, round(y));
                accuracy = (C(1,1) + C(2,2)) / (C(1,1) + C(1, 2) + +C(2, 1) + C(2,2));
                precision = C(1,1) / (C(1,1) + C(1,2));
                sensitivity = (C(1,1)) / (C(1, 1)+C(2, 1));
                specificity = (C(2,2)) / (C(2, 2)+C(1, 2));

                fprintf('fold %d... ', m)

                list_acc = [list_acc, accuracy];
                list_time = [list_time, time];

                fprintf('Accuracy: %2f \n',accuracy) %proportion of correct predictions
                %fprintf('Precision: %2f \n',precision) %proportion of positive results that were correc
                %fprintf('Sensitivity: %2f \n',sensitivity) %amount of customers leaving that were corre
                %fprintf('Specificity: %2f \n',specificity) %amount of customers staying that were corre
                %fprintf('AUC: %2f \n',AUC) %amount of customers staying that were correctly identified
            end

            average_acc = mean(list_acc)
            average_time = mean(list_time)

            mlp_results = vertcat(mlp_results, {[i, i, i], j, k, average_acc, average_time});
        end
    end
end
```

4. SVM Results Table

The results are saved to a .mat file so experiments only have to be run once.

```
%compile recorded results into table
MLP_Results = cell2table(mlp_results(2:end, :));
MLP_Results.Properties.VariableNames = mlp_results(1,:);
```

```
% save results to .mat file
save('MLP_Results.mat','MLP_Results');
```

Rank	Structure	Momentum	Learning Rate	10-fold Accuracy	10-fold time
1	[50,50,50]	0.5	0.9	0.805	17.638
2	[50,50,50]	0.5	0.01	0.800	15.384
3	[50,50,50]	0.5	0.5	0.799	14.395
4	[50,50,50]	0.5	0.1	0.792	14.338
5	[30,30,30]	0.5	0.9	0.788	7.093
6	[50,50,50]	0.9	0.5	0.787	8.359
7	[50,50,50]	0.9	0.01	0.784	8.518
8	[30,30,30]	0.5	0.5	0.780	5.452
9	[50,50,50]	0.9	0.9	0.778	6.955
10	[30,30,30]	0.5	0.1	0.778	6.321
21	[10, 10,10]	0.9	0.5	0.757	1.763
22	[10, 10,10]	0.5	0.9	0.755	2.071
23	[10, 10,10]	0.5	0.1	0.752	2.483
24	[30,30,30]	0.1	0.01	0.745	3.395
25	[10, 10,10]	0.1	0.1	0.742	1.419
26	[50,50,50]	0.1	0.01	0.742	5.205
27	[10, 10,10]	0.1	0.01	0.742	2.247
28	[50,50,50]	0.01	0.01	0.740	5.591
29	[30,30,30]	0.01	0.01	0.740	3.903
30	[10, 10,10]	0.01	0.01	0.738	9.670
31	[10, 10,10]	0.9	0.1	0.736	1.866
32	[10, 10,10]	0.01	0.1	0.734	7.308
40	[50,50,50]	0.01	0.9	0.720	4.215
41	[10, 10,10]	0.1	0.9	0.718	1.079
42	[50,50,50]	0.01	0.1	0.716	3.076
43	[10, 10,10]	0.01	0.5	0.714	1.227
44	[50,50,50]	0.1	0.9	0.713	2.971
45	[50,50,50]	0.1	0.5	0.708	2.978
46	[30,30,30]	0.1	0.9	0.701	2.120
47	[30,30,30]	0.1	0.5	0.699	1.372
48	[30,30,30]	0.01	0.5	0.697	2.378

5. Parameter Selection

```
%The best parameters are retrieved from the results table
load('MLP_Results')

%find row with best accuracy, and the index
[best_acc, best_acc_idx] = max(MLP_Results.Avg_Acc);

%get row using the index
best_parameters = MLP_Results{best_acc_idx,:};
best_struct = [best_parameters(1), best_parameters(1), best_parameters(1)];
best_momentum = best_parameters(4);
best_learningrate = best_parameters(5);
```

Train MLP with optimal hyperparameters and save network for testing

```
%A new model is fitted according to the optimum hyperparameters in the
%previous section. This will be tested on the test data in 'NC_MLP_TestScript' and final accuracy
%will be calculated to assess model generalisability

load('Training.mat')
load('Testing.mat')

train_x = Training(:, 1:42)';
train_y = Training(:, 43)';

test_x = Testing(:, 1:42)';
test_y = Testing(:, 43)';


%print paramaters in run
fprintf('\n Best Hidden layer structure: %2f. ', best_struct);
fprintf('Best Momentum: %2f. ', best_momentum);
fprintf('Best Learning Rate: %2f. \n', best_learningrate);
%fprintf('ValFailures (Early Stopping): %2f. \n', n);

tic

% Choose a Training Function
trainFcn = 'traingdx';  %Gradient descent with momentum and adaptive learning rate backpropagation

% Create a Pattern Recognition Network
best_net = patternnet([50, 50, 50], trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
best_net.input.processFcns = {'removeconstantrows','mapminmax'};

%define parameters for training
best_net.trainParam.mc = best_momentum; %momentum
best_net.trainParam.lr = best_learningrate; %learning rate
best_net.trainParam.max_fail = 15;
best_net.trainParam.epochs = 500;

% Setup Division of Data for Training, Validation, Testing
```

```
% For a list of all data division functions type: help nndivision
best_net.divideFcn = 'dividerand';  % Divide data randomly
best_net.divideMode = 'sample';  % Divide up every sample
best_net.divideParam.trainRatio = 70/100;
best_net.divideParam.valRatio = 15/100;
best_net.divideParam.testRatio = 15/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
best_net.performFcn = 'crossentropy';  % Cross-Entropy

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
best_net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotconfusion', 'plotroc'};

% Train the Network
[best_net,tr] = train(best_net,train_x,train_y);

% Test the Network
y = best_net(test_x);
time = toc;

C = confusionmat(test_y, round(y));
accuracy = (C(1,1) + C(2,2)) / (C(1,1) + C(1, 2) + +C(2, 1) + C(2,2));
precision = C(1,1) / (C(1,1) + C(1,2));
sensitivity = (C(1,1)) / (C(1, 1)+C(2, 1));
specificity = (C(2,2)) / (C(2, 2)+C(1, 2));

fprintf('Accuracy: %2f \n',accuracy) %proportion of correct predictions
fprintf('Precision: %2f \n',precision) %proportion of positive results that were correctly classified
fprintf('Sensitivity: %2f \n',sensitivity) %amount of customers leaving that were correctly idenified
fprintf('Specificity: %2f \n',specificity) %amount of customers staying that were correctly identified
```

Save Best Neural Network 'best_net'

```
%Save the network with the best hyperparameters as ;bestnet'. This is used
%to predict test data in 'NC_MLP_TestScript'

save best_net
```