Coursework 2: Machine Learning & Neural Computation

Course Coordinator: Dr. Aldo Faisal

Report Prepared By: Jenna Luchak, MSc HBR
Email: jkl17@ic.ac.uk
CID: 01429938

Date Submitted: December 4, 2017

Jenna Luchak, MSc HBR jkl17@ic.ac.uk CID: 01429938

Question 1

a) The effect of learning rate on the learning behaviour of a multi-layer perceptron (MLP) was analyzed. A plot of resulting learning curves from this analysis can be seen in Figure 1 below.

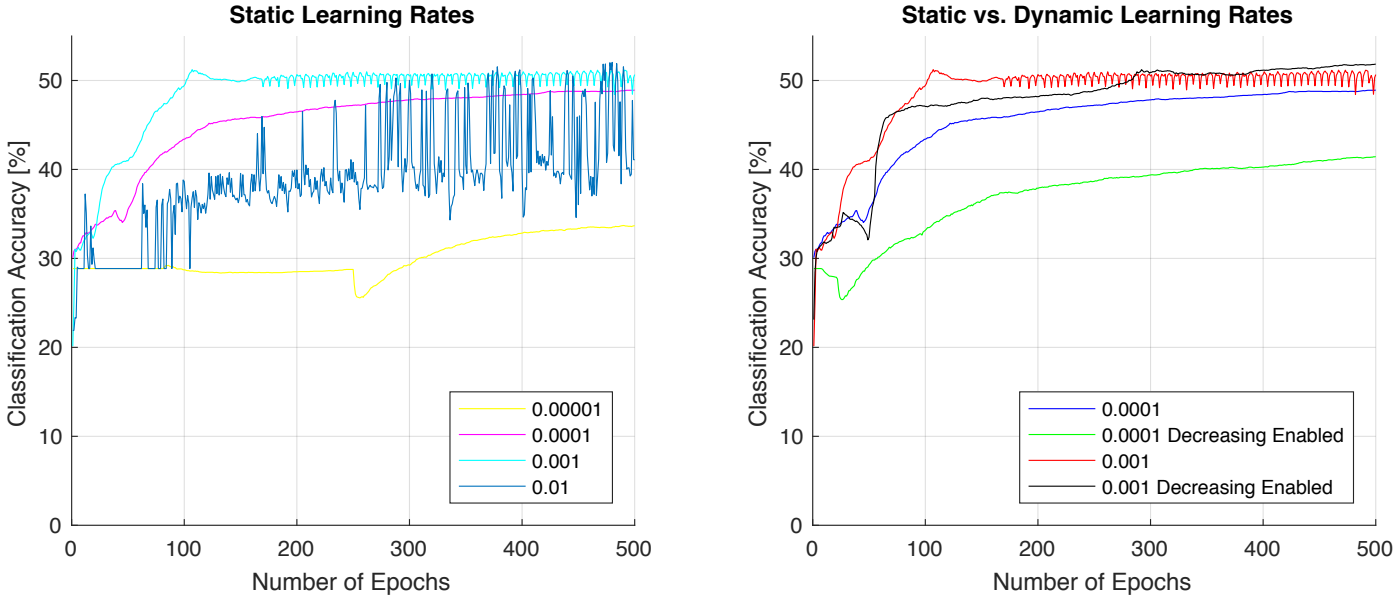

Figure 1: Multi-layer perceptron (MLP) learning curves tested with different learning rates. [Left] A comparison of different static learning rates. [Right] The same initial learning rates tested using static and dynamic strategies.

When a static learning rate strategy is executed, as seen in the left image of Figure 1, the MLP performs optimally between the learning rates of 0.0001 and 0.001. As the rate increases from 0.0001, the training speed of the MLP improves; its learning curve becomes steeper and reaches maximum accuracy sooner. However, the system also begins to shown signs of instability and over fitting, as represented by the curves fluctuation from 150-500 epochs. When the learning rate is set to a larger value than 0.001, such as 0.01, the learning curves performance greatly reduces. It lacks stability across all epochs and fluctuates significantly. When the learning rate is decreased to a value smaller than 0.0001, such as 0.00001 the training speed is slower, as is evident by the constant classification accuracy across 0-250 epochs, and it's overall accuracy decreases.

Through observation of the right image in Figure 1, it can be seen that when the learning rate is high, like 0.001 a dynamic strategy is recommended over a static approach; the dynamic approach reflects a smoother curve with no fluctuations. Although the dynamic strategies training speed is relatively slower, it's accuracy over 500 epochs is higher, compared to the static approach. In contrast, when the learning rate is low, such as 0.0001, applying a dynamic strategy over a static approach would not be recommended, because while both approach's result in smooth performance curves, the dynamic approach's overall accuracy and training speed is slower.

b) The effect of network architecture on MLP performance was analyzed. Different numbers of neurons and hidden layers were tested and the resulting learning curves can be seen in Figure 2 below.

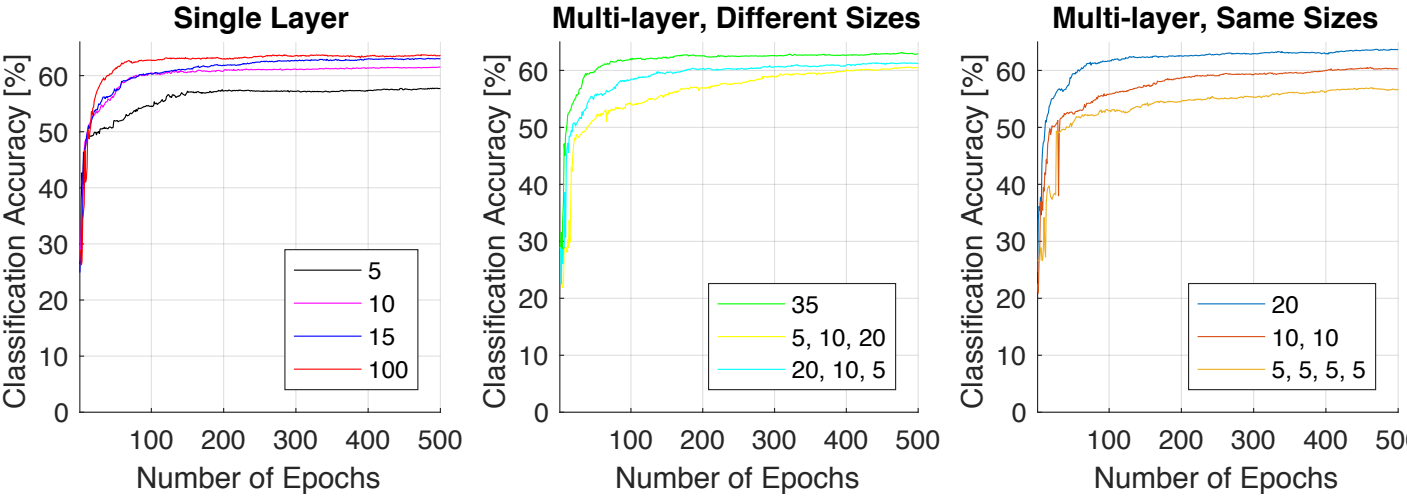

Figure 2: Three graphs depicting Multi-layer perceptron (MLP) learning curves tested with different network architectures. [Left] Different amounts of neurons in a single hidden layer network. [Middle] The same number of total neurons split uniquely into multiple hidden layers. [Right] The same number of total neurons split equally into multiple hidden layers.

As can be seen from the left image in Figure 2, as the number of neurons per layer increases, the learning curves training speed and overall accuracy improves; the curve reaches it's maximum accuracy faster. From the middle image in Figure 2, it can be seen that by splitting the total number of neurons up into multiple layers decreases the performance of the MLP, compared to when the same amount of neurons is in a single hidden layer. Additionally, when splitting up the total number of neurons into multiple layers, the training speed of the MLP is slower when the first layer contains the least amount of neurons and the last layer contains the most, compared to the contrast configuration. Lastly, the more hidden layers present in the architecture, the slower the training speed and performance. This conclusion is evident in the right image in Figure 2, as the learning curve with 2 layers of 10 neurons per layer learns faster than the curve with 4 layers for 5 neurons per layers. Based on these results, it can be concluded that a single layers of neurons is the recommended configuration, however, if a system had to use multiple hidden layers, it would be recommended to use as few as needed and put the largest amount of neurons in the first layer.

c) My optimal MLP configuration yielded classification accuracy, after 500 epochs, of 61.88%. This is an acceptable accuracy because if the data were to be classified randomly, there is a 1 in 4 (25%) chance at predicting the label accurately. The chosen parameters for this MLP system are shown below in Table 1.

Table 1: A List of Optimal Parameters for MLP System

Hidden Layers	Neurons in Each Layer	Learning Rate	Decreasing Enabled?
1	15	0.001	Yes

The optimal MLP configuration was chosen under the criteria of producing justifiably high classification accuracy while maintaining a low computational time. Firstly, a single hidden layer was chosen because not only do multiple hidden layers increase computation time, but they also do not improve the accuracy of the system. As can be seen from the middle and right plots in Figure 2, splitting up the same number of neurons from one hidden layer into multiple hidden layers decreases the classification accuracy of the system. Secondly, a total of 15 neurons were selected because when larger values of neurons, such as 50 or 100, were tested in a single layer, it was found that the resulting classification accuracy improved by no more than 1%. Adding neurons to the MLP's network increases computational training time; therefore, the small improvement to the accuracy was not a justifiable reason to increase the number of neurons anymore than 15. Lastly, although the MLP_REST function does not depend on learning rate because it uses a resilient gradient descent back propagation method, if I had the choice, I would select a learning rate of 0.001 and enable decreasing because, as can be seen from the right plot in Figure 1, it produces a learning curve with the largest overall classification accuracy and smoothest performance with minimal fluctuation.

There are a total of 105 weights in this 15 neuron, single hidden layer neural network. The calculation for this result can be seen in Equation 1 below, where, H = Number of hidden layers, and Ni = Number of neurons in the i^{th} hidden layer. Note that there are 3 input neurons in this method that represent the X, Y and Z components of the accelerometer and 4 output neurons that represent the classification labels.

$$W = \sum_{i=1}^{H} (3N_i) + (4N_i) = (3 * 15) + (4 * 15) = 105 \quad (1)$$

The calculated confusion matrix of this 4-label classifier, where each column represents a predicted class and each row represents the actual class can be seen in Table 2 as follows.

Table 2: Confusion Matrix for a four-way classification using Neural Network Methodology

		Predicted Class			
		1 (walking)	2 (running)	3 (walking upstairs)	4 (walking downstairs)
Actual Class	1 (walking)	977	100	135	121
	2 (running)	220	1006	85	132
	3 (walk upstairs)	213	67	572	277
	4 (walk downstairs)	210	67	275	543

Question 2

Given that the classes 1 and 2 were predicted the most accurately with respect to the confusion matrix in Table 2, these two classes are assumed to be the easiest to distinguish and will thereby be used in the following binary classification. The accuracy achieved over 500 epochs for this binary classification of classes 1 and 2 was 85.30% and the resulting confusion matrix can be seen in Table 3 below.

Table 3: Confusion Matrix for a two-way classification using Neural Network Methodology

		Predicted Class	
		1 (walking)	2 (running)
Actual Class	1 (walking)	1201	132
	2 (running)	274	1169

The same network architecture (specified in Question 1C) performed better on the two-way classification than the four-way classification. The two-way classifications accuracy (85.30%) was over 20% better than the four-way classifications accuracy (61.88%). This result is expected because with half as many classes to predict, it is more likely that you will achieve a higher accuracy.

Question 3/Question 4

K-nearest neighbors (KNN) classification methodology was implemented in my training function, TrainClassifierX, and my testing function, ClassifyX. This classifier was chosen over other classifiers like the probabilistic model because KNN is simple and good at low dimensionality data sets and since this is a binary classification for 3 dimensions, KNN was the optimal choice. KNN was used in the training function to test different values of K and return the optimal value. This was accomplished by first randomly splitting the training data and labels, given in this assignment, into a new training and a new testing data set. The classification accuracy of each K trial was calculated and the optimal K parameter was chosen with respect to the model that resulted in the largest overall classification accuracy. The input arguments of the training function were also passed along as a parameter for the testing function, as they were used as the training data and labels for the test function.

My classifier predicted classes by first calculating the distance of every training data point with respect to each test data point. Next, the labels of the K closest data points (shortest distance between points) were grouped together and then the mode of those points was calculated. To see the annotated script for this assignment see the Appendix. The resulting accuracy of my classifier was 85.99% and the confusion matrix is as follows in Table 4.

Table 4: Confusion Matrix for a two-way classification using K-Nearest Neighbors Methodology

		Predicted Class	
		1 (walking)	2 (running)
Actual Class	1 (walking)	1236	97
	2 (running)	292	1151

My classifier has 2825 parameters in total, and the calculation for this result is as follows.

$$Parameters = size(parameters.k, 1) + size(parameters.NearestNeighbours, 1) = 1 + 2824 = 2825$$

Each one of my training data points is considered a parameter because the classifier requires each training data point to make a prediction on the testing data. My K value for the KNN model is also considered a parameter because it is required by the KNN method to calculate a prediction. Additionally, my training function tests and calculates an optimal K value, therefore, I characterized K as a parameter rather than a hyperparameter because I consider a hyper parameter something that doesn't require any training in the model and cannot be estimated from the data. My classifier did not have any relevant hyperparameter.

Question 5

The multilayer perceptron and my classifier performed similarly. The multilayer perceptron achieved an accuracy of 85.30%, while my classifier achieved an accuracy of 85.99%, therefore, my classifier performed slightly more accurately overall than the multilayer perceptron. By dissecting the perceptron and my classifiers confusion matrices, as shown in Table 3 and 4 respectively, it can be seen that my classify predicted class 1 data points better than the perceptron method, while the perceptron was better at predicting class 2 data points.

Some advantages of the perceptron method are that it is powerful, can be easily updated with new data using back-propagation and it can model complex relationships, while some disadvantages are that it is prone to over fitting, can require significant computing power and has a long computation time. My classifier used the K-nearest neighbors method and some advantages to this model are that it is a relatively simple model, can handle multi-class classification and regression, and is "lazy" because it doesn't need to be trained. Some disadvantages are that it performs poorly if the data set has a high dimensionality, requires a lot of memory power and it requires the user to define a meaningful distance function or k value.

Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Prepared By: Jenna Luchak
% CID: 01429938
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Clear the workspace and command window before begining the script
    clc;
    clear;

%% Load Data
% data - readings from the accelerometer. Each column corresponds to
% respectively X, Y and Z axis.
%
% labels - ID of the activity
% 1 - walking
% 2 - running
% 3 - walking upstairs
% 4 - walking downstairs
    load('Activities.mat');

%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 1A
%%%%%%%%%%%%%%%%%%%%%%%%

% Define the default network's architecture.
    nbrOfNeuronsInEachHiddenLayer = [5]; % 1 hidden layer of 5 neurons

% Define the Maximum number of epochs.
% Epoch - one forward pass and one backward pass of all the training examples.
    nbrOfEpochs_max = 500;

% Create a vector of static learning rates to test
    LR_static = [0.00001 0.0001 0.001 0.01];

% Create a vector of dynamic learning rates to test
    LR_dynamic = [0.0001 0.001];

% Calculate the classification accuracies for static learning rates
for i=1:length(LR_static)
    % Should learning rate decrease with each epoch?
        enable_decrease_learningRateStatic = 0; %1 for enable decreasing, 0 for disable
        learningRate_decreaseValue = 0.0000001; % decrease value
        min_learningRate = 0.00005; % minimum learning rate

    % Define the learning rate
        learningRate = LR_static(i);

    % Call the MLP_1A function to calculate classification accuracy
        [accuracy_1aS, best_prediction_1aS] = MLP_1A(train_data, train_labels, test_data,
test_labels, nbrOfNeuronsInEachHiddenLayer, learningRate, nbrOfEpochs_max,
enable_decrease_learningRateStatic, learningRate_decreaseValue, min_learningRate);

    % Store each learning rates accuracy vector into a matrix
        accuracy_1A_static(i,:) = accuracy_1aS;
end

% Calculate the classification accuracies for static learning rates
for j=1:length(LR_dynamic)
    % Should learning rate decrease with each epoch?
        enable_decrease_learningRateDynamic = 1; %1 for enable decreasing, 0 for disable
        learningRate_decreaseValue = 0.0000001; % decrease value
        min_learningRate = 0.00005; % minimum learning rate

    % Define the learning rate
        learningRate = LR_dynamic(j);
```

```
    % Call the MLP_1A function to calculate classification accuracy
        [accuracy_1aD, best_prediction_1aD] = MLP_1A(train_data, train_labels, test_data,
test_labels, nbrOfNeuronsInEachHiddenLayer, learningRate, nbrOfEpochs_max,
enable_decrease_learningRateDynamic, learningRate_decreaseValue, min_learningRate);

    % Store each learning rates accuracy vector into a matrix
        accuracy_1A_dynamic(j,:) = accuracy_1aD;
end

% Plot the Learning Curves in a 1x2 subplot format
% The first subplot is all static learning curves with accuracy calculated in
% percentage
    figure(1)
    x = 1:nbrOfEpochs_max;
    subplot(1,2,1)
    hold on
    grid on
    plot(x,accuracy_1A_static(1,:)*100,'y'); % 0.00001 static
    plot(x,accuracy_1A_static(2,:)*100,'m'); % 0.0001 static
    plot(x,accuracy_1A_static(3,:)*100,'c'); % 0.001 static
    plot(x,accuracy_1A_static(4,:)*100); % 0.01 static
    hold off
    xlabel('Number of Epochs','fontsize',15);
    ylabel('Classification Accuracy [%]','fontsize',15);
    title('Static Learning Rates');
    legend('0.00001','0.0001','0.001','0.01','location','southeast');
    set(gca,'fontsize',15)
    axis([0,500,0,55]);
% The second subplot is all dynammic curves tested along with the static
% curves of the same value
    subplot(1,2,2)
    hold on
    grid on
    plot(x,accuracy_1A_static(2,:)*100,'b'); % 0.0001 static
    plot(x,accuracy_1A_dynamic(1,:)*100,'g'); % 0.0001 dynamic
    plot(x,accuracy_1A_static(3,:)*100,'r'); % 0.001 static
    plot(x,accuracy_1A_dynamic(2,:)*100,'k'); % 0.001 dynamic
    legend('0.0001','0.0001 Decreasing Enabled','0.001','0.001 Decreasing
Enabled','location','southeast');
    xlabel('Number of Epochs','fontsize',15);
    ylabel('Classification Accuracy [%]','fontsize',15);
    title(' Static vs. Dynamic Learning Rates');
    set(gca,'fontsize',15)
    axis([0,500,0,55]);

%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 1B
%%%%%%%%%%%%%%%%%%%%%%%%

% Define a set of different Network architecture's in a structure arrray
% Each element of the array is a vector containing the number of neurons
% in each hidden layer.
% For example:
% [1] - 1 hidden layer with 1 neuron
% [2 3] - 2 hidden layers with 2 and 3 neurons respectively

% Single Hidden Layer with different total neurons
    num(1).a = [5];
    num(2).a = [10];
    num(3).a = [15];
    num(4).a = [20];
    num(5).a = [30];
    num(6).a = [40];
    num(7).a = [50];
    num(8).a = [100];

% Multiple hidden layers for 20 neurons in total
    num(9).a = [10 10];
```

```
    num(10).a = [5 5 5 5];

% Multiple hidden layers for 35 neurons in total
    num(11).a = [35];
    num(12).a = [5 10 20];
    num(13).a = [20 10 5];

for i=1:13
    % Network's architecture.
        nbrOfNeuronsInEachHiddenLayer = num(i).a;

    % Maximum number of epochs.
        nbrOfEpochs_max = 500;

    % Call the MLP_REST function
        [accuracy_1b, best_prediction_1b] = MLP_REST(train_data, train_labels, test_data,
test_labels, nbrOfNeuronsInEachHiddenLayer, nbrOfEpochs_max);

    % Store the MLP_REST vectors into a matrix
        accuracy_1B(i,:) = accuracy_1b; % Classification Accuracy
        best_prediction_1B(i,:) = best_prediction_1b; % Classification Labels
end

% Plot the learning curves for the main report in a 1x3 subplot format
% The first subplot will display various totals of neurons in a single
% hidden layer
    figure(2)
    subplot(1,3,1)
    grid on
    hold on
    plot(x,accuracy_1B(1,:)*100,'k'); % 5
    plot(x,accuracy_1B(2,:)*100,'m'); % 10
    plot(x,accuracy_1B(3,:)*100,'b'); % 15
    plot(x,accuracy_1B(8,:)*100,'r'); % 100
    xlabel('Number of Epochs');
    ylabel('Classification Accuracy [%]');
    title('Single Layer');
    legend('5','10','15','100','location','southeast');
    set(gca,'fontsize',14)
    axis([1,500,0,66]);

% The second subplot will display 35 total neurons in one and three hidden
% layer configurations
    subplot(1,3,2)
    hold on
    grid on
    plot(x,accuracy_1B(11,:)*100,'g'); % 35
    plot(x,accuracy_1B(12,:)*100,'y'); % 5 10 20
    plot(x,accuracy_1B(13,:)*100,'c'); % 20 10 5
    xlabel('Number of Epochs');
    ylabel('Classification Accuracy [%]');
    legend('35','5, 10, 20','20, 10, 5','location','southeast');
    title('Multi-layer, Different Sizes');
    set(gca,'fontsize',14)
    axis([1,500,0,65]);

% The third subplot will display 20 total neurons in one, two and three
% hidden layer configurations
    subplot(1,3,3)
    hold on
    grid on
    plot(x,accuracy_1B(4,:)*100); % 20
    plot(x,accuracy_1B(9,:)*100); % 10 10
    plot(x,accuracy_1B(10,:)*100); % 5 5 5 5
    xlabel('Number of Epochs');
    ylabel('Classification Accuracy [%]');
    legend('20','10, 10','5, 5, 5, 5','location','southeast');
    title('Multi-layer, Same Sizes');
    set(gca,'fontsize',14)
```

```
    axis([1,500,0,65]);

% Plot the learning curves from 15 to 50 neurons
    figure(3)
    grid on
    hold on
    plot(x,accuracy_1B(2,:)*100,'m'); % 10
    plot(x,accuracy_1B(3,:)*100,'b'); % 15
    plot(x,accuracy_1B(4,:)*100,'k'); % 20
    plot(x,accuracy_1B(5,:)*100,'r'); % 30
    plot(x,accuracy_1B(6,:)*100,'g'); % 40
    plot(x,accuracy_1B(7,:)*100,'c'); % 50
    hold off
    xlabel('Number of Epochs','fontsize',15);
    ylabel('Classification Accuracy [%]','fontsize',15);
    legend('10','15','20','30','40','50','location','best');
    set(gca,'fontsize',15)

%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 1C
%%%%%%%%%%%%%%%%%%%%%%%%

% Define the optimal MLP configuration
% Network's architecture.
    nbrOfNeuronsInEachHiddenLayer = [15];

% Maximum number of epochs.
    nbrOfEpochs_max = 500;

% Call the MLP_REST function
    [accuracy_1c, best_prediction_1c] = MLP_REST(train_data, train_labels, test_data,
test_labels, nbrOfNeuronsInEachHiddenLayer, nbrOfEpochs_max);

% Calculate the confusion matrix and the classification accuracy for the
% "Neural Networks" four-way classification method.
% i.e. Use a two part for loop to print values of each confusion matrix element,
% such that columns represent the predicted values and the rows represent
% the actual values.
    confusion_matrix_1C = zeros(4,4); % Confusion matrix elements start at zero
    for pred=1:4 % Columns 1 to 4 of confusion matrix
        for actual=1:4 % Rows 1 to 4 of confusion matrix

        for i=1:length(test_labels)
            if test_labels(i) == actual && best_prediction_1c(i) == pred

            % If the if statement is true add 1 to the matrices element value
               confusion_matrix_1C(actual,pred) =  confusion_matrix_1C(actual,pred)+1;
            end
        end
    end
end
% Print a title that reads "Question 1C: The confusion matrix using
% Neural Networks 4 way classification method is as follows." and the
% display the confusion matrix.
    fprintf('Question 1C: The confusion matrix using Neural Networks 4 way classification
method is as follows.\n');
    disp(confusion_matrix_1C);

% Print the achieved accuracy after 500 epochs and a title that reads
% "Question 1C: The achieved accuracy after 500 epochs is"
    fprintf('Question 1C: The achieved accuracy after 500 epochs is
%3.2f\n',accuracy_1c(500)*100);

%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 2
%%%%%%%%%%%%%%%%%%%%%%%%

% Redefine the training and testing data set for a binary classification.
```

```
% Only the data points with the labels 1(walking) or 2(running)
% will be used.

% Binary set of training data
    train_data_binary = train_data(train_labels==1 | train_labels==2,:);
    train_labels_binary = train_labels(train_labels==1 | train_labels==2,:);

% Binary set of testing data
    test_data_binary = test_data(test_labels==1 | test_labels==2,:);
    test_labels_binary = test_labels(test_labels==1 | test_labels==2,:);

% Define the MLP's optimal Network's architecture.
    nbrOfNeuronsInEachHiddenLayer = [15]; % 1 hidden layer with 15 neurons

% Define the maximum number of epochs.
    nbrOfEpochs_max = 500;

% Call the MLP_REST function to calculate classification accuracy
    [accuracy_2, best_prediction_2] = MLP_REST(train_data_binary, train_labels_binary,
test_data_binary, test_labels_binary, nbrOfNeuronsInEachHiddenLayer, nbrOfEpochs_max);

% Calculate the confusion matrix and the classification accuracy for the
% "Neural Networks" binary classification method.
% i.e. Use a two part for loop to print values of each confusion matrix element,
% such that columns represent the predicted values and the rows represent
% the actual values
    confusion_matrix_2 = zeros(2,2); % Confusion matrix elements start at zero
    for pred=1:2 % Columns 1 and 2 of confusion matrix
        for actual=1:2 % Rows 1 and 2 of confusion matrix

            for i=1:length(test_labels_binary)
                if test_labels_binary(i) == actual && best_prediction_2(i) == pred

                % If the if statement is true add 1 to the matrices element value
                   confusion_matrix_2(actual,pred) =  confusion_matrix_2(actual,pred)+1;
                end
            end
         end
    end
% Print a title that reads "Question 2: The confusion matrix using
% Neural Networks 2-way classification method is as follows." and the resulting confusion
matrix
    fprintf('Question 2: The confusion matrix using Neural Networks 2-way classification
method is as follows.\n');
    disp(confusion_matrix_2);

% Print the acheived accuracy after 500 epochs and a title that reads
% "Question 2: The achieved accuracy after 500 epochs is"
    fprintf('Question 2: The achieved accuracy after 500 epochs is
%3.2f\n',accuracy_2(500)*100);

%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 3
%%%%%%%%%%%%%%%%%%%%%%%%

% Call the TrainClassifierX function to determine the correct parameters to
% pass onto function ClassifyX
    parameters = TrainClassifierX(train_data_binary,train_labels_binary);

% Call the ClassifyX function to predict the labels corresponding to the
% binary test data points
    desired_class_labels = ClassifyX(test_data_binary, parameters);

%%%%%%%%%%%%%%%%%%%%%%%%
%% Question 4
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Calculate the confusion matrix and the classification accuracy for the
% "K Nearest Neighbours" binary classification method

% Use a two part for loop, print values of each confusion matrix element,
% such that columns represent the predicted values and the rows represent
% the actual values
    confusion_matrix_4 = zeros(2,2); % Confusion matrix elements start at zero
    for pred=1:2 % Columns 1 and 2 of confusion matrix
        for actual=1:2 % Rows 1 and 2 of confusion matrix

            for i=1:length(test_labels_binary)
                if test_labels_binary(i) == actual && desired_class_labels(i) == pred

                % If the if statement is true add 1 to the matrices element value
                    confusion_matrix_4(actual,pred) =  confusion_matrix_4(actual,pred)+1;
                end
            end
         end
    end
% Print a title that reads "Question 4: The confusion matrix using K Nearest Neighbours
% method:" and the resulting confusion matrix
    fprintf('Question 4: The confusion matrix using K Nearest Neighbours method:\n');
    disp(confusion_matrix_4);

% Sum the diagonal terms of the confusion matrix
    numerator = confusion_matrix_4(1,1)+confusion_matrix_4(2,2);

% Sum all terms of the confusion matrix
    denominator = sum(sum(confusion_matrix_4));

% Calculate the classification accuracy of the "K Nearest Neighbours
% Method" in units of percent [%], rounded to 2 digits after the decimal
    accuracy_KNN = round(numerator/denominator*100,2);

% Calculate the number of parameters in the classifier
    Num_parameters = size(parameters.k,1)+size(parameters.NearestNeighbours,1);

% Print the achieved accuracy and a title that reads "Question 4: The
% achieved accuracy using KNN methodology is"
    fprintf('Question 4: The achieved accuracy using KNN methodology is
%3.2f\n',accuracy_KNN);

%% Complete the sanity check for this assignment
    SanityCheck

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```