Coding Assignment 1
Computational Intelligence
Dr. S. Hajipour

Mohammad Hossein Shafizadegan
99104781

November 2, 2023

Contents

1   Question 1

a

In order to obey clean code rules, we have a developed a function called "plot_fact" for this section. The initial values of the parameters will be provided to this function as input arguments.

```matlab
function plot_fact(init_w1, init_w2, init_T, beta)
```

In this function, we have to define and create a *meshgrid* for X and Y as follows:

```matlab
global p % define a global variable for the plot handle
dx = 0.01;
dy = 0.01;
beta = 0.5;

x = -1:dx:1;
y = -1:dy:1;
[X,Y] = meshgrid(x,y);
```

Then we assign random initial values to w_1, w_2, and T and form the activation function.

```matlab
% initialize some initial values for w1, w2, and T
w1 = 0.5;
w2 = 0.5;
T = 0.5;

f_act = 1./(1 + exp(-1*beta.*(w1*X+w2*Y-T))); % calculate the initial f_act values
```

After that we have to deal with interactive sliders for each of the parameters. Using MATLAB builtin *uicontrol* command, we create three sliders. The code used for creating one of these sliders can be seen below.

```matlab
% Slide bar for w1
s_w1 = uicontrol('Parent',f,'Style','slider','Position',[81,110,419,23],...
                 'value',w1, 'min',-5, 'max',5);
bgcolor = f.Color;
sl1 = uicontrol('Parent',f,'Style','text','Position',[50,110,23,23],...
                   'String','-5','BackgroundColor',bgcolor);
sl2 = uicontrol('Parent',f,'Style','text','Position',[500,110,23,23],...
                   'String','5','BackgroundColor',bgcolor);
sl3 = uicontrol('Parent',f,'Style','text','Position',[240,85,100,23],...
                   'String','w1','BackgroundColor',bgcolor);
```

In order to interactively change the parameters and observe the results, we have to develop a callback function for the sliders and assign the callback function to them. In the callback function, we first parse the new value of the parameters, then we generate the activation function again regarding new values. The code for the callback function is as follows:

```matlab
function update (src, event, b_w1, b_w2, b_T)
   global p % access the global variable p
   dx = 0.01;
   dy = 0.01;
   beta = 0.5;

   x = -1:dx:1;
   y = -1:dy:1;
```

```
9        [X,Y] = meshgrid(x,y);
10
11       w1 = get(b_w1, 'Value'); % get the current slider value for w1 using the get function
12       w2 = get(b_w2, 'Value'); % get the current slider value for w2 using the get function
13       T = get(b_T, 'Value'); % get the current slider value for T using the get function
14       f_act = 1./(1 + exp(-1*beta.*(w1*X+w2*Y-T))); % calculate the new f_act values
15       p.ZData = f_act; % update the plot
16
17       fprintf("w1= %d ,  w2 = %d ,   T = %d \n", w1, w2, T);
18
19   end
```
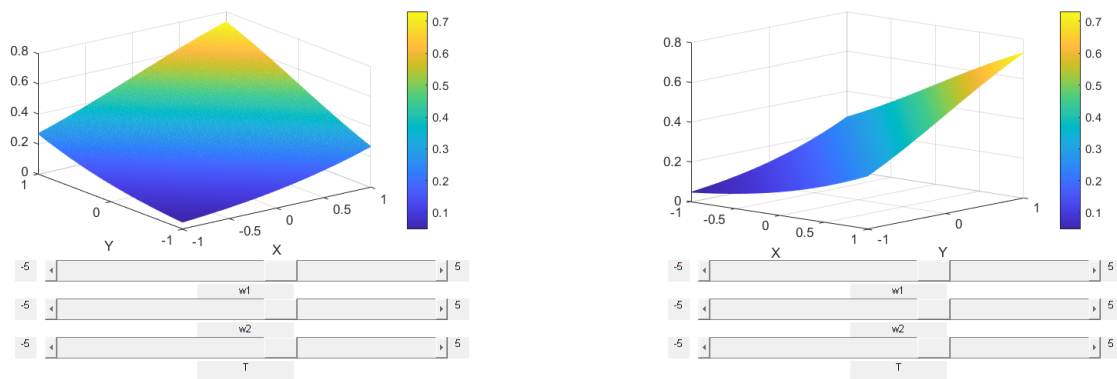
The result can be seen below:
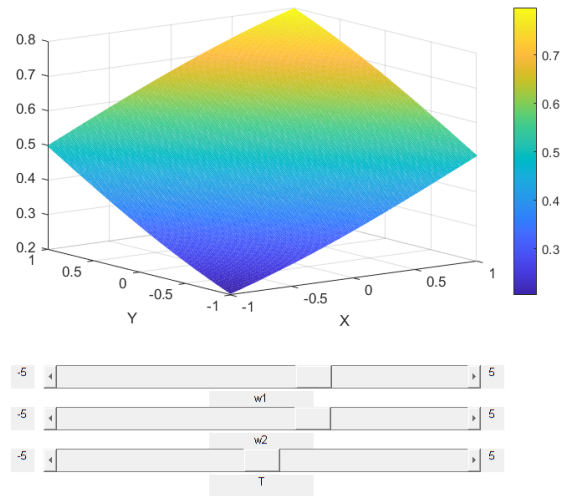
b

Here we simply set $w_1 = w_2 = T = \beta = 1$. The results will be as follows:
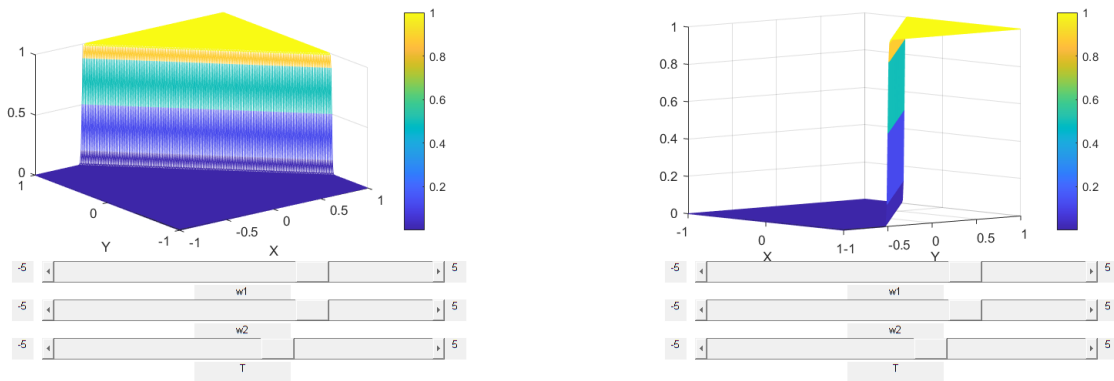

c

In order to implement the OR function, we have to set the value of each weights equal to 2 and the threshold will be 1. As it is said in the instructions, we choose a huge value for β, e.g. 100.

```
1    %% part C
2
3    % OR function
4    w1 = 2;
5    w2 = 2;
6    T = 1;
7
8    beta = 100;
9
10   plot_fact(w1, w2, T, beta)
```
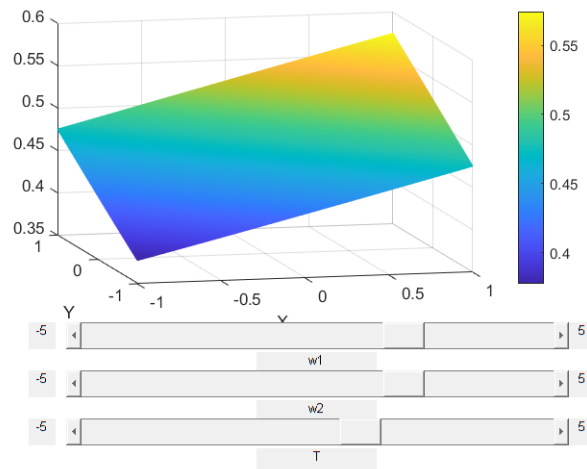
The results can be seen here.

Based on the above figures, it can be concluded that the activation function for huge values of β is quite the same as step threshold function.

d

Now we set the value of β to 0.1 and visualize the results.

2   Question 2

a

First we properly load the dataset into the MATLAB workspace. Then we separate the data corresponding to the first two classes after extracting their indices. The code for this section is as follows:

```
1    % load dataset
2    irisTable = readtable('../iris.csv');
3    irisData = table2array(irisTable(:,1:4));
4
5    % Extract data for Iris-setosa and Iris-versicolor classes
6    setosa_idx = find(strcmp(irisTable.Var5, 'Iris-setosa'));
7    versicolor_idx = find(strcmp(irisTable.Var5, 'Iris-versicolor'));
8
9    setosa_data = irisData(setosa_idx, :);
10   versicolor_data = irisData(versicolor_idx, :);
```

For visualizing our data in the features domain, we have developed a function named "scatter_features" which receives data corresponding to each class and the number of two desired features as input argument. In this function, we simply visualize data using "scatter" function as follows:

```
1    function scatter_features(data1, data2, f1,f2)
2        % Plotting features: sepal length vs sepal width
3        scatter(data1(:,f1), data1(:,f2), 'filled', 'MarkerFaceColor', 'r');
4        hold on;
5        scatter(data2(:,f1), data2(:,f2), 'filled', 'MarkerFaceColor', 'g');
6        xlabel(sprintf('feature %d', f1));
7        ylabel(sprintf('feature %d', f2));
8        title(sprintf('feature %d vs. feature %d', f1, f2));
9        legend('Iris-setosa', 'Iris-versicolor');
10       hold off;
11   end
```

Now for different combinations of features, we visualize our data to inspect which features separate our data of two classes better.
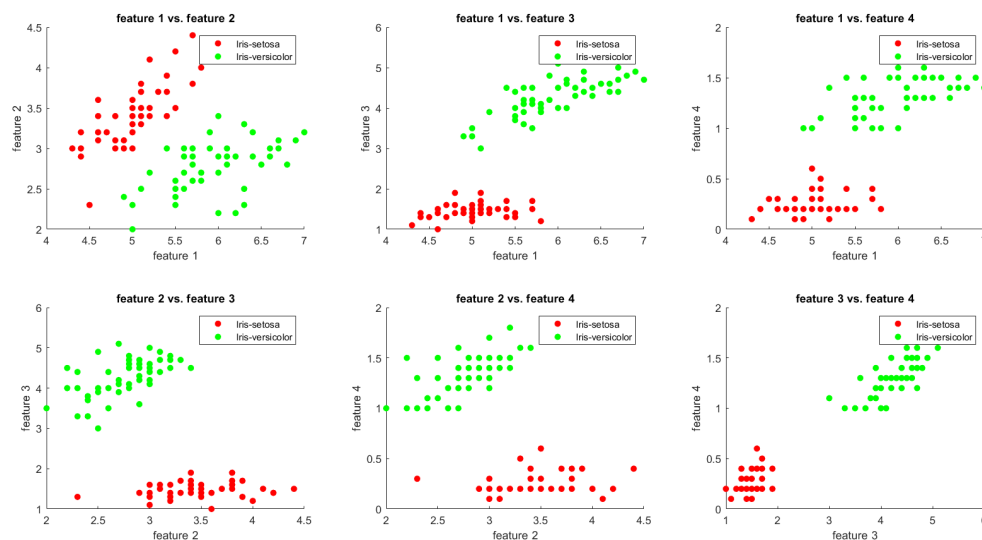
Regarding this figure, we can infer that these two classes of data can be separated quite well using features 3 and 4. For the rest of this question we will use these features for training process.

b

c

First we randomly separate 80% of our data for training and their corresponding labels, as follows:

```
1    P = randperm (50);
2    train_data = [setosa_data(P(1:40), 3:4) ; versicolor_data(P(1:40), 3:4)];
3
4    % all data labels within an array (first class 1 then class 0)
5    out = [ones(1,40), zeros(1,40)];
```

In order to implement the online learning algorithm we have developed a function called "online_learning". The input arguments of this function are the learning rate η, number of iterations, training data and their labels.
This function will find the values of weights and threshold and will return these values alongside with the number of iterations actually took for achieving the convergence.

```
1    function [w_array, theta_array, iter_num] = online_learning(eta, n_iter, x, out)
```

In this function, we first randomly initiate the values of the parameters. Then we have defined two matrices to store the values of weights and threshold for each iteration.

```
1    init_w = [0; 0];
2    init_theta = 5;
3
4    w = init_w;
5    theta = init_theta;
6
7    w_array = [w];
8    theta_array = [theta];
```

The main part of the algorithm will be discussed here as in a nested for loop, we update the values of the weights and threshold after each iteration using the delta rule. Any further explanation are provided within the code using comments. The code for this section will be as follows:

```
1    for i = 1:n_iter
2        e = 0;
3        for j = 1:length(x)
4            % Compute the output of the threshold logic unit (TLU)
5            y = w' * x(j,:)' >= theta;
6
7            % Check if the output is different from the target
8            if y ~= out(j)
9                % Update the threshold and store its value
10               theta = theta - eta * (out(j) - y);
11               theta_array = [theta_array theta];
12
13               % Update the weights and store their values
14               w = w + eta * (out(j) - y) * x(j,:)';
15               w_array = [w_array w];
16
17               % Update the error count
18               e = e + abs(out(j) - y);
```

```
19            end
20         end
21
22         % Check if the error count is less than or equal to zero
23         % If true, exit the loop as the training is complete
24         if e <= 0
25             break;
26         end
27      end
```

Now we run this online algorithm and the final values for the weight and the threshold can be seen below.

```
1       [w_online, theta_online, iter_num] = online_learning(eta, n_iter, train_data, out);
2
3       disp('Online learning');
4       fprintf('number of iterations: %d \n', iter_num)
5       fprintf('the resulting weigths: w1=%d , w2=%d \n', w_online(1, end), w_online(2, end));
6       fprintf('threshold = %d \n', theta_online(end));
```

```
Command Window                                                                                        ⊙
  Online learning
  number of iterations: 9
  the resulting weigths: w1=-5.000000e-01 , w2=-1.500000e+00
  threshold = -2
fx >> |
```
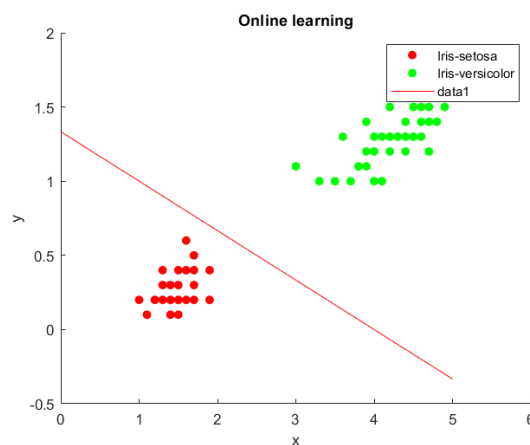
In order to plot the achieved line, we have developed a function called "plot_line". The code for this simple function is as follows:

```
1       function plot_line(w1, w2, theta)
2          slope = -w1/w2;
3          intercept = theta/w2;
4          x = 0:0.1:5;
5          y = slope*x + intercept;
6          plot(x,y,'r'); xlabel('x'); ylabel('y');
7       end
```

Using this function, we visualize the results:

7

Now for implementing the batch learning algorithm, we also develop a function called "batch_learning". The attitude toward developing the this function is quite the same as before except for the algorithm for updating the wights and the threshold which obey the batch learning rule.

The code used for updating the parameters can be seen below:

```matlab
for i = 1:n_iter
    e = 0;
    theta_c = 0;
    w_c = zeros(2,1);

    for j = 1:length(x)
        % Compute the output of the threshold logic unit (TLU)
        y = w' * x(j,:)' >= theta;

        % Check if the output is different from the target
        if y ~= out(j)
            % Update the temporary threshold and weights
            theta_c = theta_c - eta * (out(j) - y);
            w_c = w_c + eta * (out(j) - y) * x(j,:)';

            % Update the error count
            e = e + abs(out(j) - y);
        end
    end

    % Update the threshold and weights using the temporary values
    theta = theta + theta_c;
    theta_array = [theta_array theta];
    w = w + w_c;
    w_array = [w_array w];

    % Check if the error count is less than or equal to zero
    % If true, exit the loop as the training is complete
    if e <= 0
        break;
    end
end
```
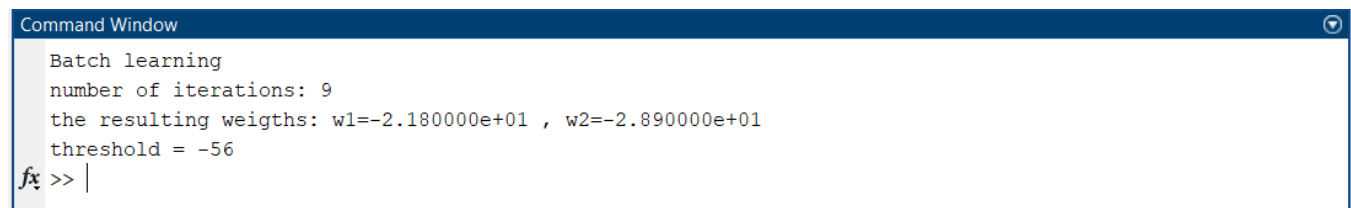
Now we run this algorithm for our data:

```matlab
[w_batch, theta_batch, n_iter_batch] = batch_learning(eta, n_iter, train_data, out);

disp('Batch learning');
fprintf('number of iterations: %d \n', n_iter_batch)
fprintf('the resulting weigths: w1=%d , w2=%d \n', w_batch(1, end), w_batch(2, end));
fprintf('threshold = %d \n', theta_batch(end));
```

```
Command Window
    Batch learning
    number of iterations: 9
    the resulting weigths: w1=-2.180000e+01 , w2=-2.890000e+01
    threshold = -56
fx >> |
```
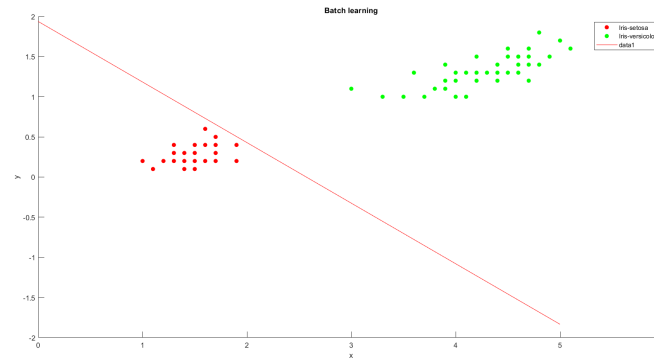
Here we have visualized the results:

```
1     figure;
2     scatter_features(setosa_data, versicolor_data, 3,4); hold on;
3     title('Batch learning')
4     plot_line(w_batch(1, end), w_batch(2, end), theta_batch(end)); hold off;
```


d

Now using the matrices we define for storing the values of weights and threshold for each algorithm, we simply plot the process using subplots. Here are the codes of these sections:
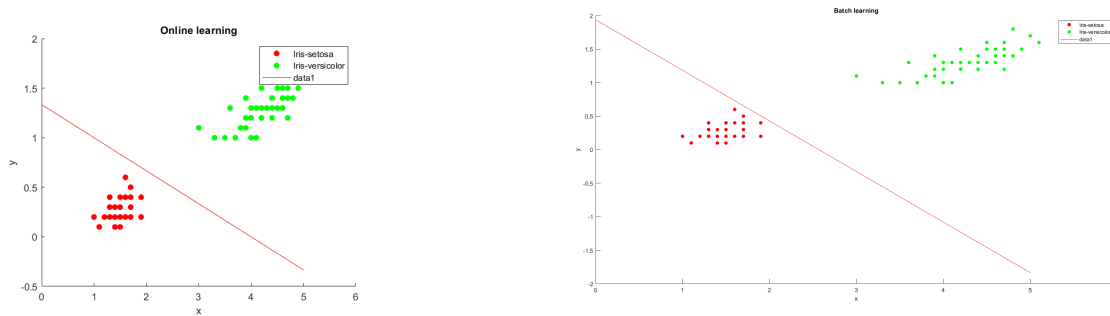
```
1     figure;
2     subplot(3,1,1)
3     plot(w_online(1,:))
4     title('Online learning')
5     xlabel('iteration')
6     ylabel('w1');
7     subplot(3,1,2)
8     plot(w_online(2,:))
9     xlabel('iteration')
10    ylabel('w2');
11    subplot(3,1,3)
12    plot(theta_online)
13    xlabel('iteration')
14    ylabel('theta');
```
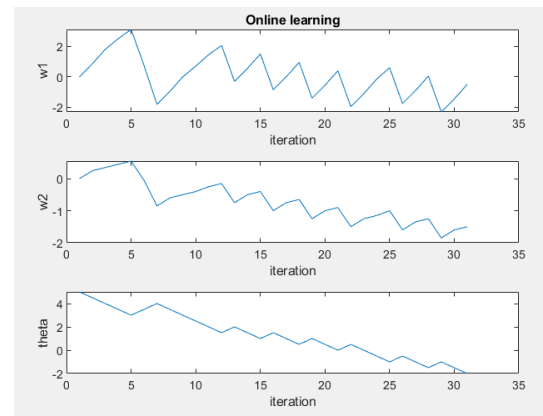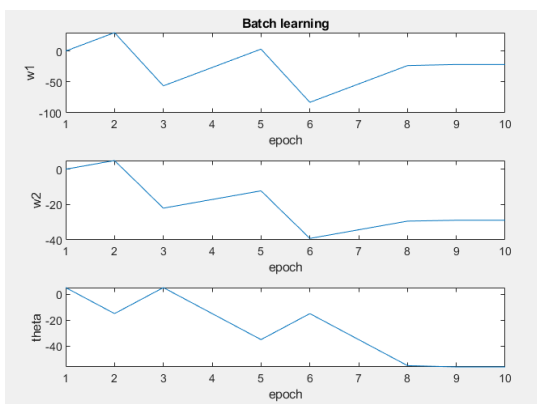
e

As we discussed the results before, we once again demonstrate them here:

f

In this section, we use our model for the remaining test data using this code:

```
1    test_data = [setosa_data(P(41:50), 3:4) ; versicolor_data(P(41:50), 3:4)];
2    out_test = [ones(1,10), zeros(1,10)];
3
4    y_online = w_online(:, end)' * test_data' >= theta_online(end);
5
6    y_batch = w_batch(:, end)' * test_data' >= theta_batch(end);
7
8    acc_online = sum(y_online == out_test)/length(out_test) * 100;
9    acc_batch = sum(y_batch == out_test)/length(out_test) * 100;
10
11   fprintf('Accuracy of classification train by online learning: %d \n', acc_online);
12   fprintf('Accuracy of classification train by batch learning: %d \n', acc_batch);
```

```
Command Window
   Accuracy of classification train by online learning: 100
   Accuracy of classification train by batch learning: 100
fx >>
```

g

First of all we extract data of all three classes the same way we did before.

```
1    % Extract data for Iris-setosa and Iris-versicolor classes
2    setosa_idx = find(strcmp(irisTable.Var5, 'Iris-setosa'));
3    versicolor_idx = find(strcmp(irisTable.Var5, 'Iris-versicolor'));
4    virginica_idx = find(strcmp(irisTable.Var5, 'Iris-virginica'));
5
6    setosa_data = irisData(setosa_idx, :);
7    versicolor_data = irisData(versicolor_idx, :);
8    virginica_data = irisData(virginica_idx, :);
```

For this section, we have modified the code for "scatter_features" in a way to support three classes of data and three features. The result will be a 3D scatter plot with different color for each class. The main modified parts of the "scatter_features" function can be seen below

10

```
1    scatter3(data1(:,f1), data1(:,f2), data1(:,f3), 'filled', 'MarkerFaceColor', 'r');
2    hold on;
3    scatter3(data2(:,f1), data2(:,f2), data2(:,f3), 'filled', 'MarkerFaceColor', 'g');
4    scatter3(data3(:,f1), data3(:,f2), data3(:,f3), 'filled', 'MarkerFaceColor', 'b');
```

For observing our dataset in the features space, we have visualized them for different combinations of features.

```
1    subplot(2,2,1);
2    scatter_features(setosa_data, versicolor_data, virginica_data, 1, 2, 3);
3    subplot(2,2,2);
4    scatter_features(setosa_data, versicolor_data, virginica_data, 1, 2, 4);
5    subplot(2,2,3);
6    scatter_features(setosa_data, versicolor_data, virginica_data, 2, 3, 4);
7    subplot(2,2,4);
8    scatter_features(setosa_data, versicolor_data, virginica_data, 1, 3, 4);
```

The results are as follows:
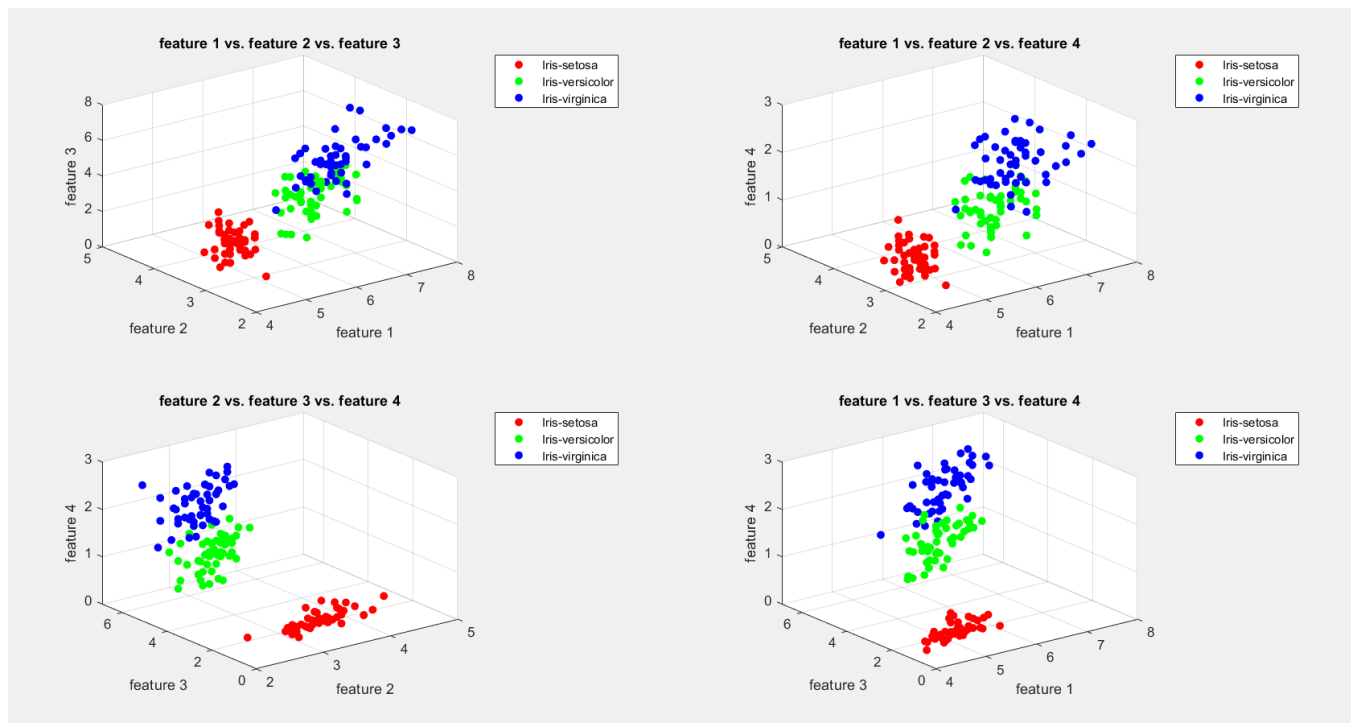
It can be seen vividly that data from classes "iris-versicolor" and "iris-virginica" can't be separated linearly using a plane. These classes are somehow a little mixed with each other which make it impossible to separate them linearly. Therefor, we can't classify these classes using a network of TLUs.

These classes can't be separated via a single TLU obviously. Also as we discussed above, we can't even separate them using a network of TLUs with a 100% accuracy.

In the following we try to separate class 1 (red) and class 2 (green) using a TLU. Then we attempt to do the same for class 2 (green) and class 3 (blue). The resulting planes will divide the 3D space into three sections each for a class. The outputs of these 2 TLUs can be used by TLUs of the next layer which implements AND function and the results will be used by the output neuron to clarify the final output by binary encoding.

11

Regarding the features figure, we have decided to choose features 1, 3, and 4 for classification.

First we train a TLU for separating class 1 and class 2 using 80% of data as training data via the following code. Also we have calculated the accuracy on the train data and test data correspondingly to assess the performance of the classifier.

```
1    selected_features = [1, 3, 4];
2
3    P = randperm (50);
4    train_data = [setosa_data(P(1:40), selected_features) ; ...
5    versicolor_data(P(1:40), selected_features)];
6
7    % all data labels within an array (first class 1 then class 0)
8    out = [ones(1,40), zeros(1,40)];
9
10   n_iter = 100;
11   eta = 0.5;
12
13   [w_12, theta_12, iter_num] = online_learning([0; 0; 0], 0, eta, n_iter, train_data, out);
```
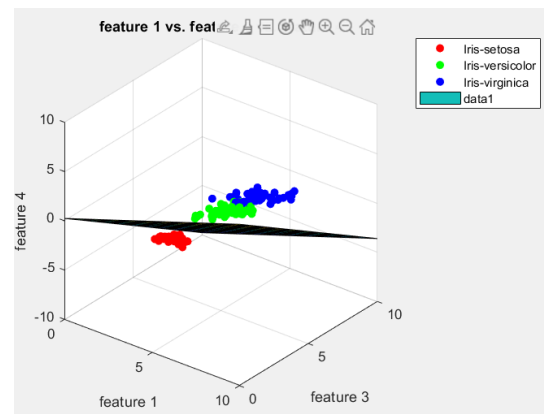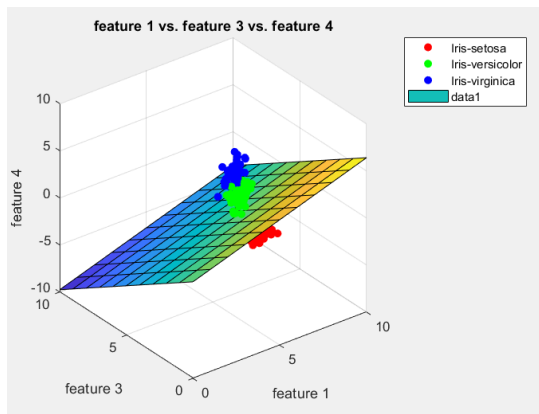
There results for this section can be seen below:

We note that we have converted the function "plt_line" to "plot_plane" to support plotting a plane in 3D space. It can be seen that we can properly separate these two classes using a single TLU.

Now we try to separate class 2 and 3 using a TLU using the same procedure.

```
1    %% train a TLU for separating class 2 and class 3
2
3    P = randperm (50);
4    train_data = [versicolor_data(P(1:40), selected_features) ; ...
5    virginica_data(P(1:40), selected_features)];
```

```
6
7          % all data labels within an array (first class 1 then class 0)
8          out = [ones(1,40), zeros(1,40)];
9
10         eta = 0.01;
11         n_iter = 1000;
12
13         [w_23, theta_23, n_iter_23] = online_learning([0; 0; 0], 0, eta, n_iter, train_data, out);
```

As discussed before, we are not expecting to completely separate these data via a TLU so we have to be more careful here by choosing a smaller value for η and increasing the number of iteration.
The results are as follows:

```
Separating class 2 and 3
number of iterations: 1000
the resulting weigths: w1=9.570000e-01 , w2=-1.205000e+00 , w3=-1.262000e+00
threshold = -1.920000e+00
Accuracy of classification of class 2 and 3 on test data: 85
Accuracy of classification of class 2 and 3 on train data: 9.875000e+01
fx >>
```
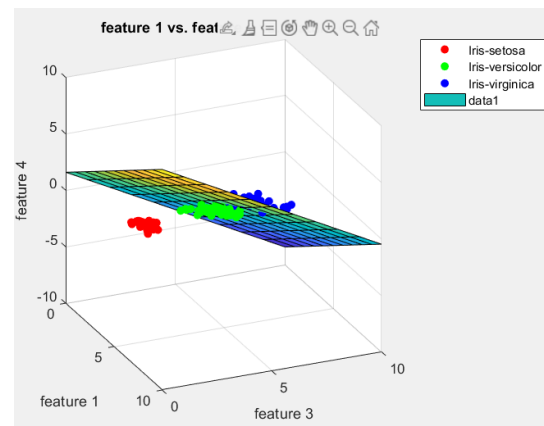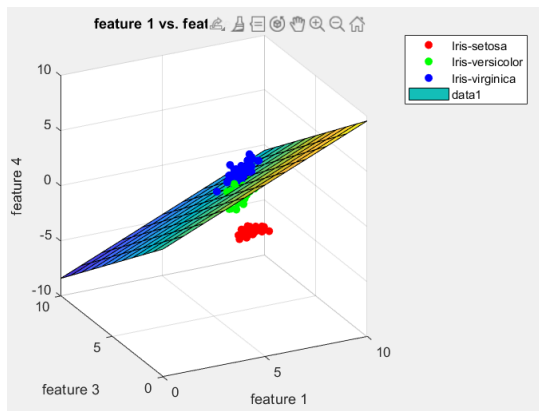

Based on the above results, it can be seen that we couldn't even separate training data after 1000 iterations with a 100% accuracy.

The final result of dividing the 3D space into three sections using 2 TLUs for this problem is as follows:

13