Coding Assignment 1
Computational Intelligence
Dr. S. Hajipour

Mohammad Hossein Shafizadegan
99104781

October 31, 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.

```
    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:

```
    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.

```
    % 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.

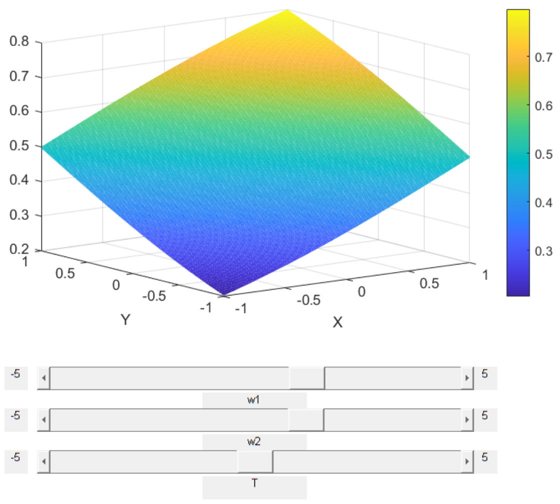
```
    % Slide bar for w1
    s_w1 = uicontrol('Parent',f,'Style','slider','Position',[81,110,419,23],...
                     'value',w1, 'min',-5, 'max',5);
    bgcolor = f.Color;
    sl1 = uicontrol('Parent',f,'Style','text','Position',[50,110,23,23],...
                    'String','-5','BackgroundColor',bgcolor);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

```
    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;
```

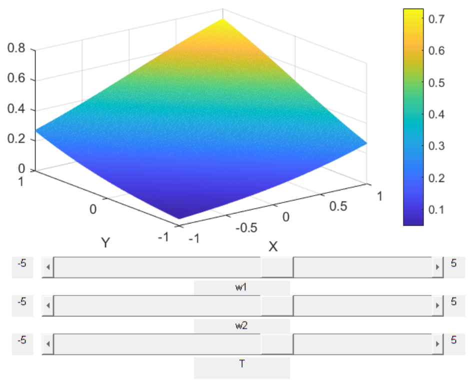
```
        [X,Y] = meshgrid(x,y);

        w1 = get(b_w1, 'Value'); % get the current slider value for w1 using the get function
        w2 = get(b_w2, 'Value'); % get the current slider value for w2 using the get function
        T = get(b_T, 'Value'); % get the current slider value for T using the get function
        f_act = 1./(1 + exp(-1*beta.*(w1*X+w2*Y-T))); % calculate the new f_act values
        p.ZData = f_act; % update the plot

        fprintf("w1= %d ,  w2 = %d  ,   T = %d \n", w1, w2, T);

    end
```

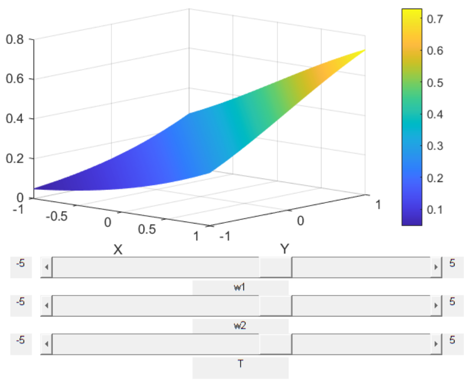
The result can be seen below:



b

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



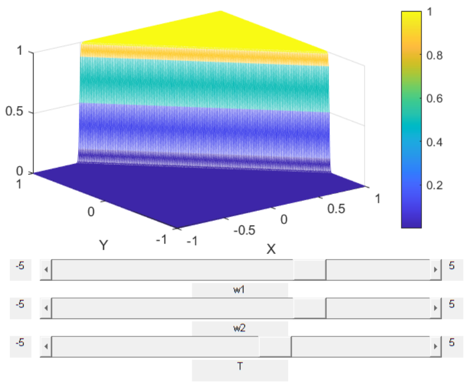


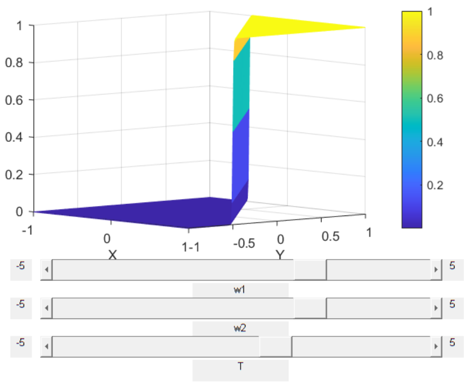
c

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.

```
%% part C

% OR function
w1 = 2;
w2 = 2;
T = 1;

beta = 100;

plot_fact(w1, w2, T, beta)
```

The results can be seen here.

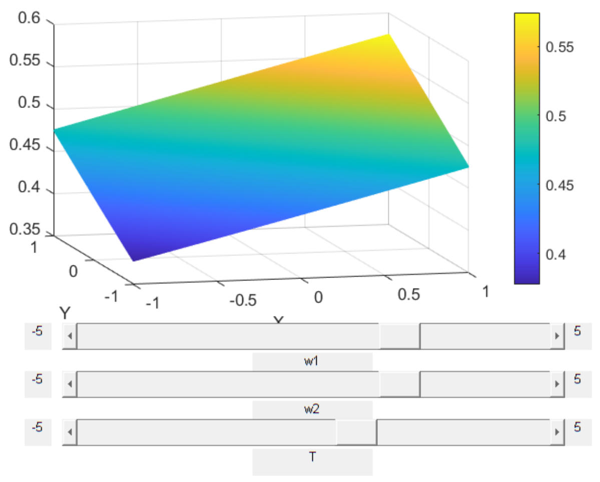




Based on the above figures, it can be concluded that the activation function for huge values of β is quite the same as step threshold function.

d

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



2 Question 2

a

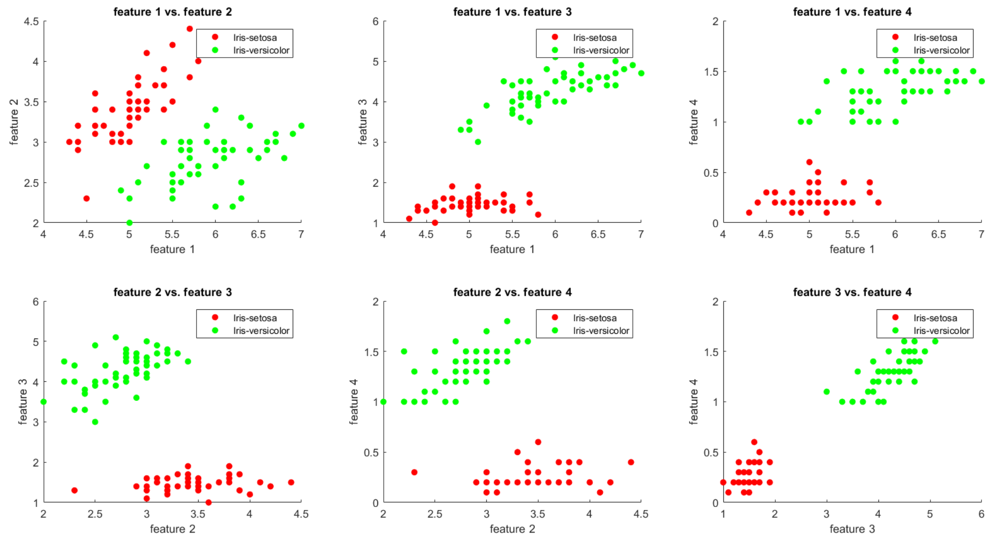
First we properly load the dataset into the MATLAB workspace. Then we separate the data corresponding to the first two classes after extracting their indices. The code for this section is as follows:

```
    % load dataset
    irisTable = readtable('../iris.csv');
    irisData = table2array(irisTable(:,1:4));

    % Extract data for Iris-setosa and Iris-versicolor classes
    setosa_idx = find(strcmp(irisTable.Var5, 'Iris-setosa'));
    versicolor_idx = find(strcmp(irisTable.Var5, 'Iris-versicolor'));

    setosa_data = irisData(setosa_idx, :);
    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:

```
    function scatter_features(data1, data2, f1,f2)
      % Plotting features: sepal length vs sepal width
      scatter(data1(:,f1), data1(:,f2), 'filled', 'MarkerFaceColor', 'r');
      hold on;
      scatter(data2(:,f1), data2(:,f2), 'filled', 'MarkerFaceColor', 'g');
      xlabel(sprintf('feature %d', f1));
      ylabel(sprintf('feature %d', f2));
      title(sprintf('feature %d vs. feature %d', f1, f2));
      legend('Iris-setosa', 'Iris-versicolor');
      hold off;
    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:

```
P = randperm (50);
train_data = [setosa_data(P(1:40), 3:4) ; versicolor_data(P(1:40), 3:4)];

% all data labels within an array (first class 1 then class 0)
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.

```
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.

```
init_w = [0; 0];
init_theta = 5;

w = init_w;
theta = init_theta;

w_array = [w];
theta_array = [theta];
```

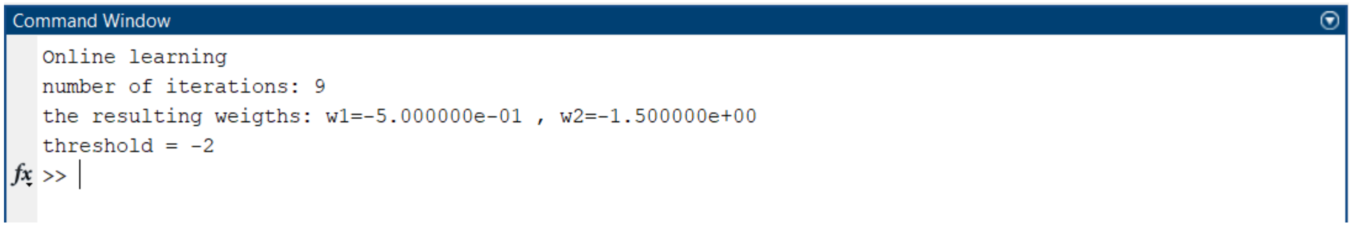
The main part of the algorithm will be discussed here as in a nested for loop, we update the values of the weights and threshold after each iteration using the delta rule. Any further explanation are provided within the code using comments. The code for this section will be as follows:

```
for i = 1:n_iter
  e = 0;
  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 threshold and store its value
      theta = theta - eta * (out(j) - y);
      theta_array = [theta_array theta];

      % Update the weights and store their values
      w = w + eta * (out(j) - y) * x(j,:)';
      w_array = [w_array w];

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

```
            end
        end

        % 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 online algorithm and the final values for the weight and the threshold can be seen below.

```
    [w_online, theta_online, iter_num] = online_learning(eta, n_iter, train_data, out);

    disp('Online learning');
    fprintf('number of iterations: %d \n', iter_num)
    fprintf('the resulting weigths: w1=%d , w2=%d \n', w_online(1, end), w_online(2, end));
    fprintf('threshold = %d \n', theta_online(end));
```

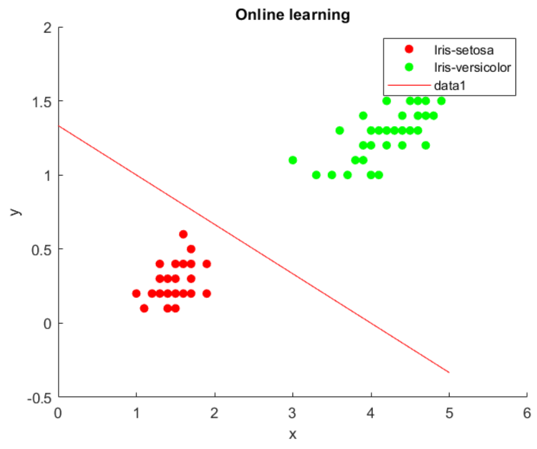


```
Online learning
number of iterations: 9
the resulting weigths: w1=-5.000000e-01 , w2=-1.500000e+00
threshold = -2
>> |
```

In order to plot the achieved line, we have developed a function called ”plot_line”. The code for this simple function is as follows:

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

Using this function, we visualize the results:

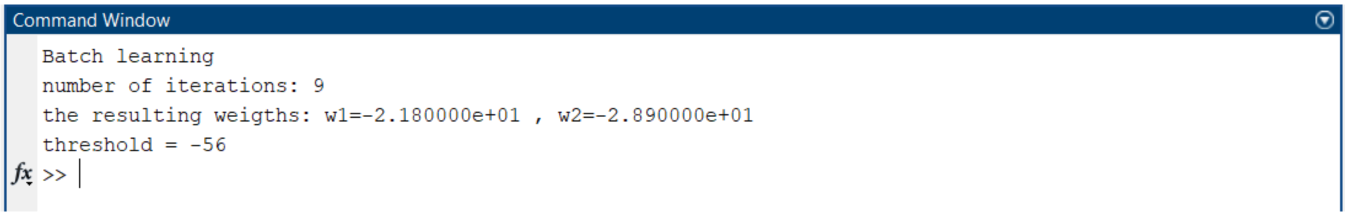


Now for implementing the batch learning algorithm, we also develop a function called ”batch_learning”. The attitude toward developing the this function is quite the same as before except for the algorithm for updating the wights and the threshold which obey the batch learning rule.
The code used for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

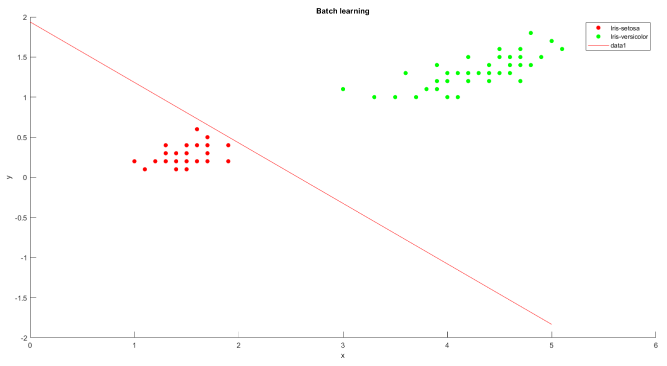
```matlab
[w_batch, theta_batch, n_iter_batch] = batch_learning(eta, n_iter, train_data, out);

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));
```



```
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:

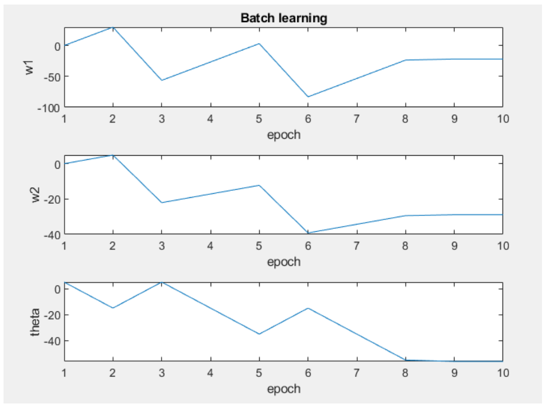
```
    figure;
    scatter_features(setosa_data, versicolor_data, 3,4); hold on;
    title('Batch learning')
    plot_line(w_batch(1, end), w_batch(2, end), theta_batch(end)); hold off;
```

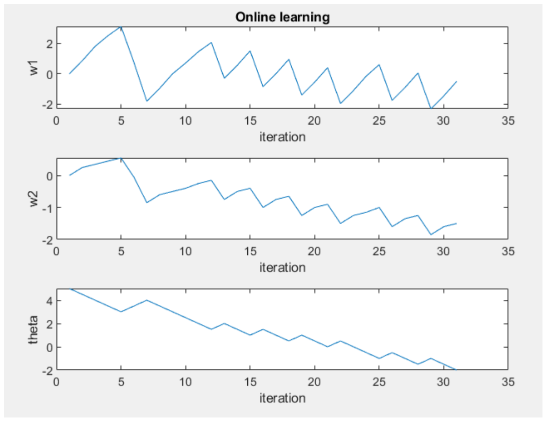


d

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:

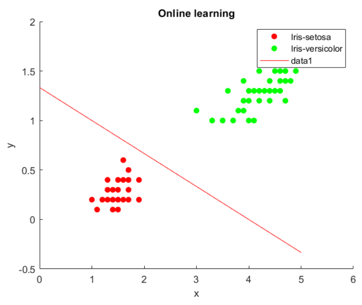
```
    figure;
    subplot(3,1,1)
    plot(w_online(1,:))
    title('Online learning')
    xlabel('iteration')
    ylabel('w1');
    subplot(3,1,2)
    plot(w_online(2,:))
    xlabel('iteration')
    ylabel('w2');
    subplot(3,1,3)
    plot(theta_online)
    xlabel('iteration')
    ylabel('theta');
```

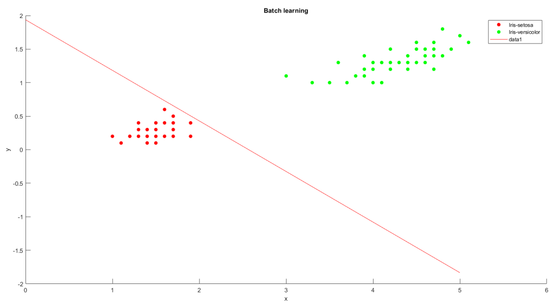




e

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

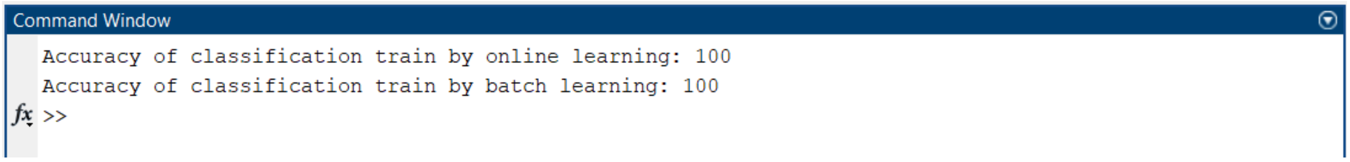




f

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

```
    test_data = [setosa_data(P(41:50), 3:4) ; versicolor_data(P(41:50), 3:4)];
    out_test = [ones(1,10), zeros(1,10)];

    y_online = w_online(:, end)' * test_data' >= theta_online(end);

    y_batch = w_batch(:, end)' * test_data' >= theta_batch(end);

    acc_online = sum(y_online == out_test)/length(out_test) * 100;
    acc_batch = sum(y_batch == out_test)/length(out_test) * 100;

    fprintf('Accuracy of classification train by online learning: %d \n', acc_online);
    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
>>
```