Machine Learning Assignment 2

1. Code

```
%% Implementing Perceptron

% 1. Read Data and Divide into Trg and Testing Data

% 2. Perform Perceptron Trg till all trg samples are
correctly classfied

% 3. Perform Testing using the Updated Weights

% 4. Plot Decision Boundary on scatter plot

% 5. Check performance

clc;

close all;

clear all;

%% Reading Dataset and Separating Trg and Test Data

dataset = load('data3.mat'); % Reading the CSV File

% Splitting the Data

[trg_data, test_data, trg_class, test_class] =
split_data(dataset.data,75);

% Performing Training

[predicted_class, final_weights] =
per_trg(trg_data,trg_class,0.5);

% Performing Classification on Test Data
```

```
bias = ones(size(test_data,1),1); % Addition of Bias
Feature

test_data = horzcat(bias,test_data);

for i = 1 : size(test_data,1)

    z = sum(final_weights.*test_data(i,:));

    % Applying Step Activation Function

    if z >= 0

        y_hat = 1;

    else

        y_hat = -1;

    end

    y(i) = y_hat;

end

%% Create the Decision Boundary

b = final_weights(1);

w1 = final_weights(2);

w2 = final_weights(3);

x = max(trg_data(:,1));

for i = 1 : size(trg_data,1)

    yline = -(b + w1*x)/w2;

    x = x - 0.1;

    xline(i) = x;

    y_line(i) = yline;
```

```
end

%% Plotting Data

figure(1)

set(gcf, 'Position', get(0,'Screensize')); % Fullscreen
Plot

gscatter(trg_data(:,1),trg_data(:,2),trg_class);

grid on;

hold on

%plot decision Boundary

plot(xline, y_line, 'linewidth', 2);

legend('Class - 1',' Class - 2','Decision Boundary');

xlabel('F1');

ylabel('F2');

title('PERCEPTRON IMPLEMENTATION - DECISION BOUNDARY');

%% Performance Evaluation

conf_matrix = confusionmat(test_class , y');

Accuracy =
sum(diag(conf_matrix))/sum(sum(conf_matrix))*100;

%% FUNCTIONS USED (APPENDED BELOW)

% DATASET SPLITTING INTO TRG AND TESTING DATA

% PERCEPTRON TRAINING

%% DATASET SPLITTING INTO TRG AND TESTING DATA

function [trg_data,test_data, trg_class, test_class] =
split_data(data,trg_ratio)
```

```
%SPLIT_DATA Summary of this function goes here
data = lower(data); % Lowercase Data
%data = table2array(data);
total_size = length(data);
trg_size = round(total_size * (trg_ratio/100));
test_size = total_size - trg_size;
% Shuffling the Data
index_random = randperm(length(data),length(data));
dataset = data(index_random,:);
% Separating Trg Data and Trg Class
trg_data = dataset(1 : trg_size, 1: end - 1);
trg_class = dataset(1 : trg_size, end);
% Separating Test Data and Test Class
test_data = dataset(trg_size + 1 : total_size, 1: end-1);
test_class = dataset(trg_size + 1 : total_size, end);
end
%% PERCEPTRON TRAINING
function [y, w] = per_trg(trg_data,trg_class, eta)
% Step-1 (initialize)
bias = ones(size(trg_data,1),1); % Addition of Bias Feature
data = horzcat(bias,trg_data);
[row, colm] = size(data);
w = zeros(colm,1);
```

```
w = transpose(w);

% Step - 2 (Perform Trg)

% Perform Trg till Convergence

for k = 1 : 10000 % No of Iterations

    for i = 1 : row % For all Samples

        % Compute the Weighted Sum

        z = sum(w.* data(i,:)); %

        % Applying Step Activation Function

        if z >= 0

            y_hat = 1;

        else

            y_hat = -1;

        end

        % Checking Result (if update for weights required
or not)

        delta_w = 0; % Initialize

        if trg_class(i) ~= y_hat % Not Equal Case

            % Perform Updation of weights

            delta_w = eta*(trg_class(i) -
y_hat).*data(i,:);

            w = w + delta_w;

        else % do nothing

            w = w;
```

```
        end
%         delta_w_vec(i,1) = delta_w;
        y(i) = y_hat; % Store in Array for Comparison
    end
    % (CHECKED)
    if (trg_class == y') % Until all the samples are
correctly classified
            break % for breaking k Loop
    end
end
% Testing Trg Accuracy (NOT DISPLAYED)
conf_matrix = confusionmat(trg_class , y);
accuracy =
sum(diag(conf_matrix))/sum(sum(conf_matrix))*100;
end % Function END
```

Graph:

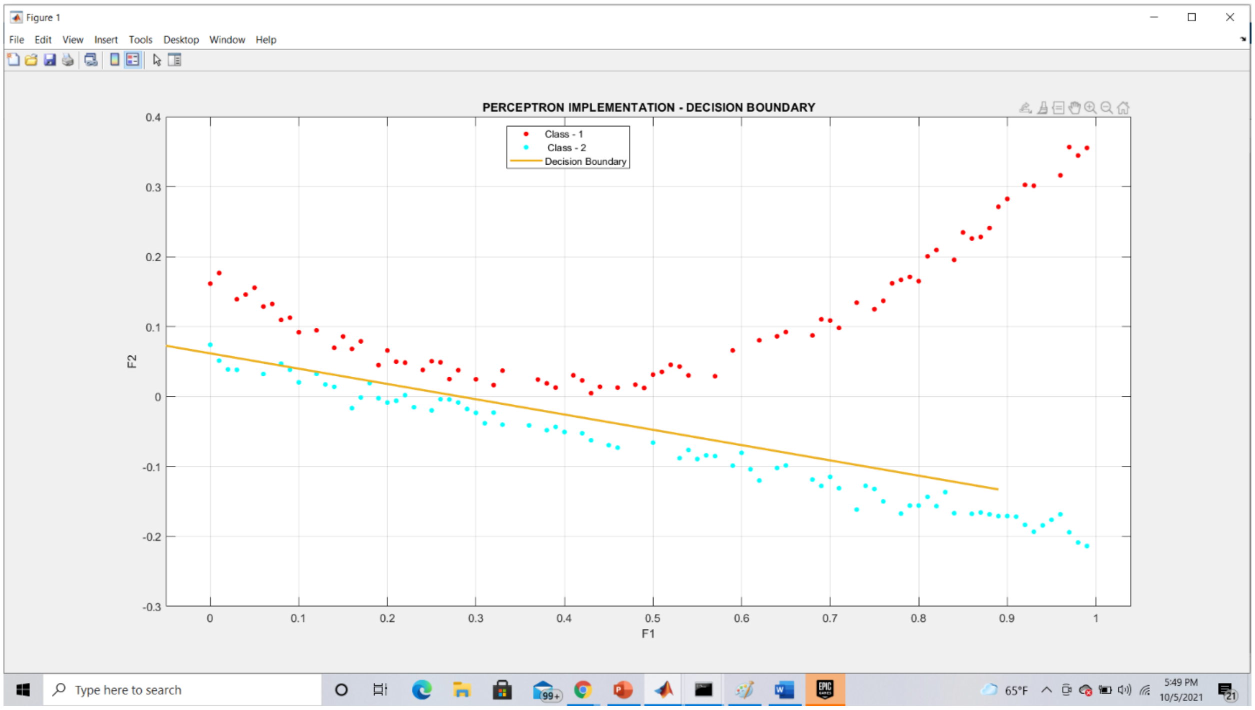


Figure 1
File Edit View Insert Tools Desktop Window Help
PERCEPTRON IMPLEMENTATION - DECISION BOUNDARY
Class - 1
Class - 2
Decision Boundary
F2
F1
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Type here to search
65°F
5:49 PM
10/5/2021

2.

a) $E = -\sum_i \left(t_i \log(x_i) + (1-t_i)\log(1-x_i) \right)$

$$x_i = \frac{1}{1+e^{-s_i}}, \quad s_i = \sum_j y_j w_{ji}$$

We can compute the derivative of the error with respect to each weight connecting the hidden units to the output units using the chain rule.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}}$$

Examining each factor in turn,

$$\frac{\partial E}{\partial x_i} = \frac{-t_i}{x_i} + \frac{1-t_i}{1-x_i}$$

$$= \frac{x_i - t_i}{x_i(1-t_i)}$$

$$\frac{\partial x_i}{\partial s_i} = x_i(1-x_i)$$

$$\frac{\partial s_i}{\partial w_{ji}} = h_j$$

$$\Rightarrow \frac{\partial E}{\partial s_i} = x_i - t_i$$

and $\frac{\partial E}{\partial w_{ji}} = (x_i - t_i) h_j$

Here it is useful to calculate the quantity $\frac{\partial E}{\partial s_j}$ where j indexes the hidden units, s_j is the weighted input sum at hidden unit j, and $h_j = \frac{1}{1+e^{-s_j}}$ is the activation at unit j.

$$\frac{\partial E}{\partial s_j} = \sum_{i=1} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j}$$

$$= \sum_{i=1} (x_i - t_i)(w_{ji})(h_j(1-h_j))$$

$$\frac{\partial E}{\partial h_j} = \sum_{i=1} \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial s_i} \frac{\partial s_i}{\partial y_j}$$

$$= \sum_{i=1} \frac{\partial E}{\partial x_i} x_i (1-x_i) w_{ji}$$

Then a weight w_{kj} connecting input unit k to hidden unit j has gradient

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{kj}}$$

$$= \sum_{i=1} (x_i - t_i)(w_{ji})(h_j(1-h_j))(y_k)$$

By recursively computing the gradient of the error with respect to the activity of each neuron, we can compute the gradients for all weights in a network.

b) $E = -\sum_i t_i \log(x_i)$

$$x_i = \frac{e^{s_i}}{\sum_{c=1}^{m} e^{s_c}}$$

Thus, computing the gradient yields

$$\frac{\partial E}{\partial x_i} = -\frac{t_i}{x_i}$$

$$\frac{\partial x_i}{\partial s_k} = \begin{cases} \frac{e^{s_i}}{\sum_{c=1}^{m} e^{s_c}} - \left(\frac{e^{s_i}}{\sum_{c=1}^{m} e^{s_c}}\right)^2 & i = k \\ -\frac{e^{s_i} e^{s_k}}{\left(\sum_{c=1}^{m} e^{s_c}\right)^2} & i \neq k \end{cases}$$

$$= \begin{cases} x_i(1-x_i) & i = k \\ -x_i x_k & i \neq k \end{cases}$$

$$\frac{\partial E}{\partial s_i} = \sum_k \frac{\partial E}{\partial x_k} \frac{\partial x_k}{\partial s_i}$$

$$= \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial s_i} - \sum_{k \neq i} \frac{\partial E}{\partial x_k} \frac{\partial x_k}{\partial s_i}$$

$$= -t_i(1-y_i) + \sum_{k \neq i} t_k x_i$$

$$= -t_i(1-x_i) + \sum_{k \neq i} t_k x_i$$

$$= -t_i + x_i \sum_k t_k$$

$$= x_i - t_i$$

$$\Rightarrow \frac{\partial E}{\partial w_{ji}} = \sum_i \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}}$$

$$= (x_i - t_i)\, h_j$$

and for units in the hidden layer, indexed by j,

$$\frac{\partial E}{\partial s_j} = \sum_i \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j}$$

$$= \sum_i (y_i - t_i)(w_{ji})(h_j + (-1))$$

$$\Rightarrow \frac{\partial E}{\partial s_j} = \sum_i (x_i - t_i)\,(w_{ji})\big((1-h_j)\,h_j\big)$$

3. Consider the discrete distribution $\{p_K | K = 1, 2, .., N\}$. The entropy of this distribution is given as $H = -\sum_{K=1}^{N} p_K \log p_K$. What is the distribution that maximizes this entropy? Show formal derivations using the method of Lagrange multipliers.

Ans. $H(P) = -\sum_{K=1}^{n} p_K \ln p_K$

The Lagrangian is

$$L(P, \lambda) = -\sum_{K=1}^{N} p_K \ln p_K + \lambda\left(\sum_{K=1}^{n} p_K - 1\right)$$

Taking partial derivatives with respect to p_K and equating to 0,

$$\frac{\partial}{\partial p_K} L(P, \lambda) = 0$$

$$-\ln p_K - 1 + \lambda = 0$$

$$p_1 = p_2 = \dots = p_n = 1/n$$

Because, all p_K are equal, then the condition $\sum_{K=1}^{n} p_K = 1$ gives

$p_K = 1/n$.

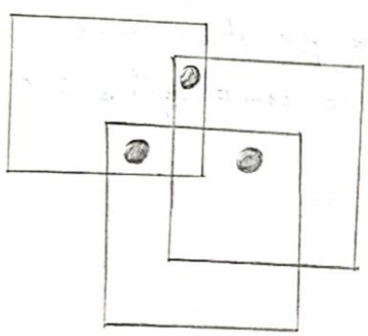
Thus the maximum entropy is

$$H_{max}(P) = -\sum_{K=1}^{n} \frac{1}{n} \ln \frac{1}{n} \Rightarrow H_{max}(P) = \ln n$$

The distribution $p_1 = p_2 = \dots = p_n = 1/n$ maximizes the entropy.

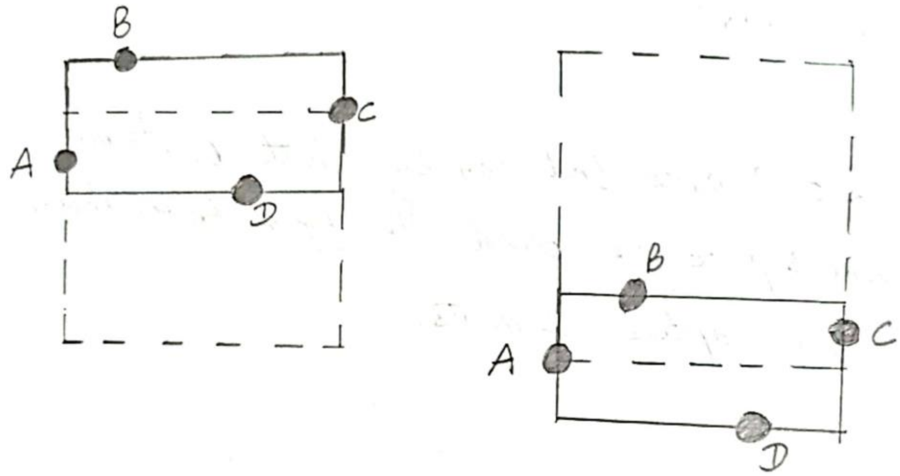
4. What is the VC dimension of axis-aligned squares? Justify your answer.

Ans.
1. There exist 3 points that can be shattered. Again, 1 points and 3 points are trivial. The figure below shows how we can capture 2 points.



So, yes, there exists an arrangement of 3 points that can be shattered.

2. No set of 4 points can be shattered.
Suppose we have four points arranged such that they define a rectangle. Now, suppose we want to select two points (A & C in this case)



The minimum enclosing square for A & C must contain either B or D so we can't capture just two points with a square.