CS303A Homework 3

> 11811721 庄湛

Q.1

> Consider the following data set comprised of three binary input attributes (A1, A2 and A3) and one binary output:
>
> | Example | A_1 | A_2 | A_3 | Output y |
> | --- | --- | --- | --- | --- |
> | x_1 | 1 | 0 | 0 | 0 |
> | x_2 | 1 | 0 | 1 | 0 |
> | x_3 | 0 | 1 | 0 | 0 |
> | x_4 | 1 | 1 | 1 | 1 |
> | x_5 | 1 | 1 | 0 | 1 |
>
> **Use the algorithm (as below) to learn a decision tree for these data. Show the computations made to determine the attribute to split at each node.**
>
> **function** DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree
>
> **if** *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
> **else if** all *examples* have the same classification **then return** the classification
> **else if** *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
> **else**
> $A \leftarrow \operatorname{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
> *tree* ← a new decision tree with root test A
> **for each** value v_k of A **do**
> $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
> *subtree* ← DECISION-TREE-LEARNING(*exs*, *attributes* − A, *examples*)
> add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
> **return** *tree*
>
> `PLURALITY-VALUE` selects the most common output value among a set of examples, breaking ties randomly.
> `IMPORTANCE` is information gain, based on entropy.

$\mathrm{H}(P) = -\sum_{i=1}^{n} p_i log p_i$

$IG(T,a) = \mathrm{H}(T) - \mathrm{H}(T|a)$

$H(y) = -0.6log0.6 - 0.4log0.4 = 0.97095$

$H(y|A_1) = \frac{4}{5}(-0.5log0.5 - 0.5log0.5) + \frac{1}{5}(-1log1 - 0log0) = 0.8$

$H(y|A_2) = \frac{3}{5}(-\frac{2}{3}log\frac{2}{3} - \frac{1}{3}log\frac{1}{3}) + \frac{2}{5}(-1log1 - 0log0) = 0.55098$

$H(y|A_3) = \frac{2}{5}(-0.5log0.5 - 0.5log0.5) + \frac{3}{5}(-\frac{2}{3}log\frac{2}{3} - \frac{1}{3}log\frac{1}{3}) = 0.95098$

$IMPOTANCE(A_1, examples) = H(y) - H(y|A_1) = 0.17$

$IMPOTANCE(A_2, examples) = H(y) - H(y|A_2) = 0.43$

$IMPOTANCE(A_3, examples) = H(y) - H(y|A_3) = 0.02$

$argmax_{a \in attributes} IMPOTANCE(a, examples) = A_2$

So there will be a new decision tree with root test A_2,

For $A_2 = 1$:

Example	A_1	A_3	**Output y**
x_3	0	0	0
x_4	1	1	1
x_5	1	0	1

$H(y_2) = -\frac{2}{3}log\frac{2}{3} - \frac{1}{3}log\frac{1}{3} = 0.97095$

$H(y_2|A_1) = \frac{2}{3}(-1log1 - 0log0) + \frac{1}{3}(-1log1 - 0log0) = 0$

$H(y_2|A_3) = \frac{1}{3}(-1log1 - 0log0) + \frac{2}{3}(-0.5log0.5 - 0.5log0.5) = 0.66667$

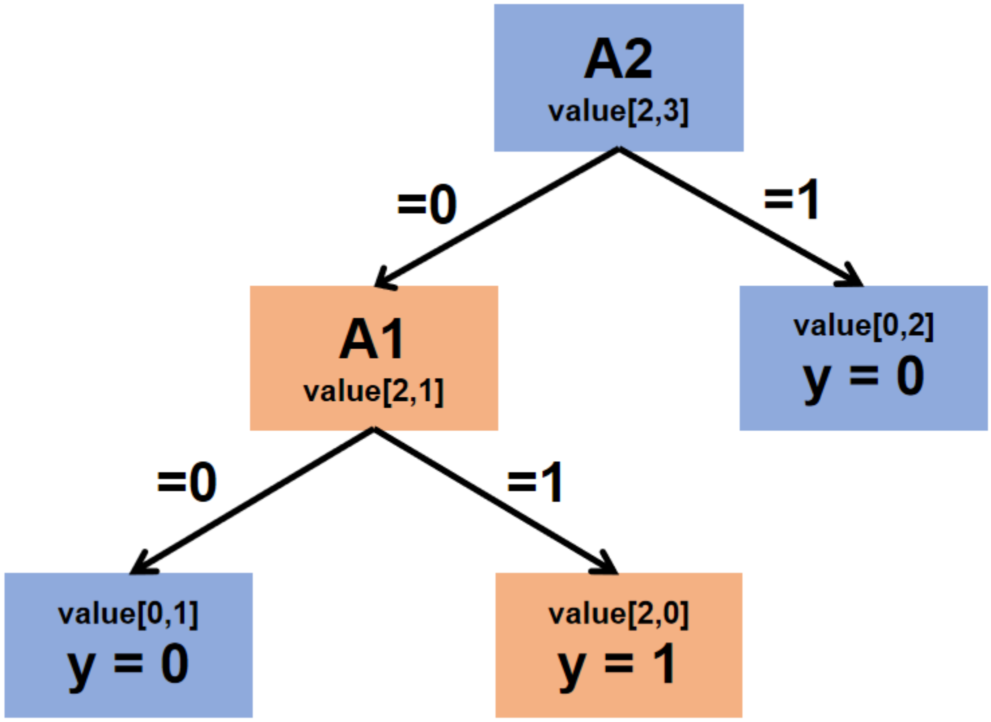
Obviosly, in this case, $argmax_{a \in attributes} IMPOTANCE(a, examples) = A_1$, and then all child examples have the same classification.

For $A_2 = 0$:

Example	A_1	A_3	**Output y**
x_1	1	0	0
x_2	1	1	0

All examples have the same classification.

So the dicision tree is like as the following figure:



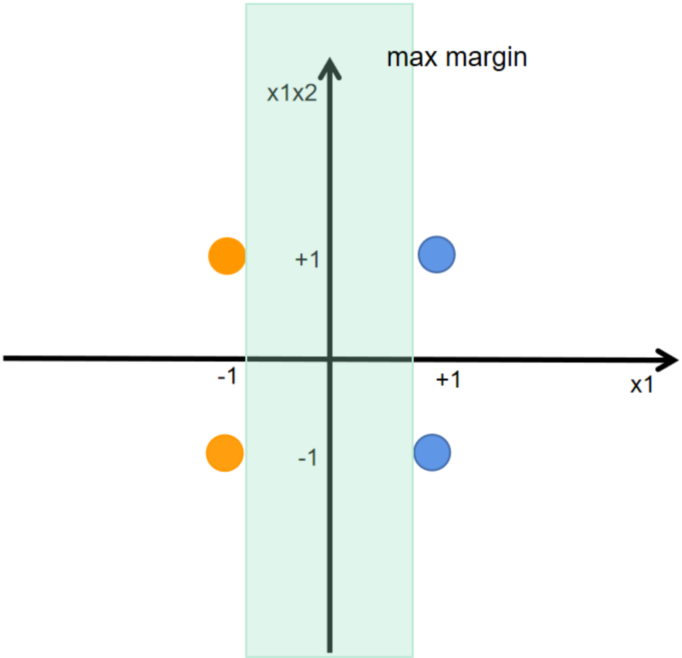
Q.2

> Construct a support vector machine that computes the XOR function. Use values of +1 and -1 (instead of 1 and 0) for both inputs and outputs, so that an example looks like ([−1, 1], 1) or ([−1, −1], −1). Map the input [x1, x2] into a space consisting of x1 and x1x2.
>
> a. Draw the four input points in this space, and the maximal margin separator. What is the margin?
>
> b. Now draw the separating line back in the original Euclidian input space.

a.

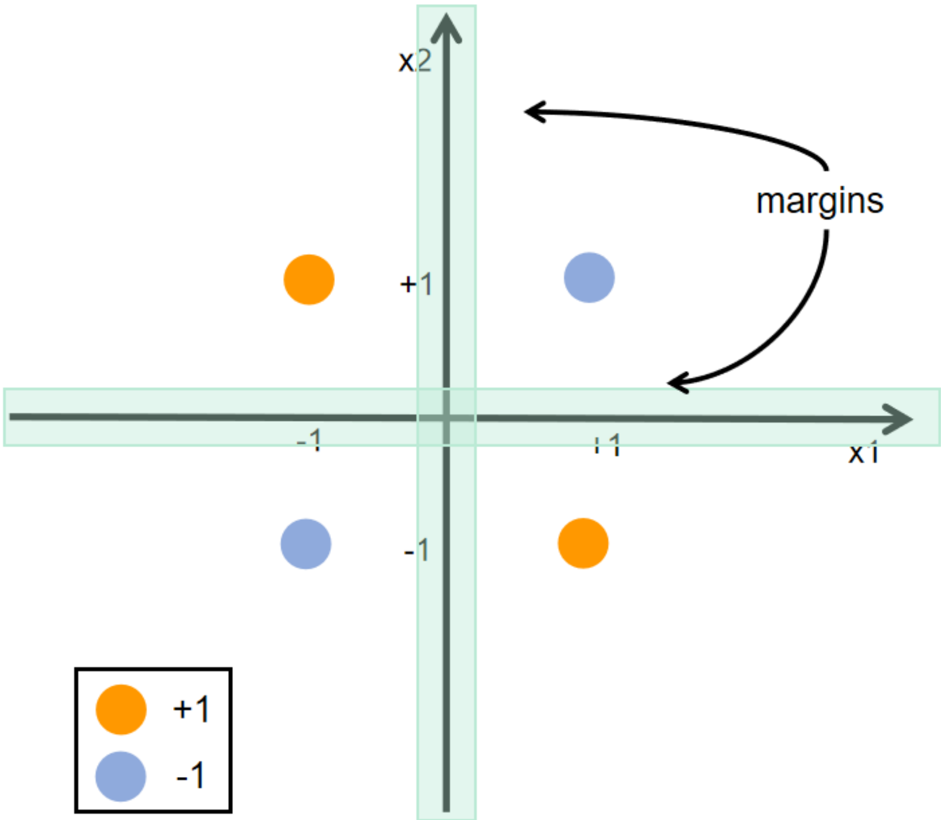
[x1, x2]	x1 xor x2	[x1, x1x2]
[-1, -1]	-1	[-1, +1]
[-1, +1]	+1	[-1, -1]
[+1, -1]	+1	[+1, -1]
[+1, +1]	-1	[+1, +1]

As shown in the figure below, all points with positive (orange) results fall at x1x2 = -1, and vice versa, all points with negative (blue) results fall at x1x2 = +1. So the maximum magin separator is the line x1x2 = 0, and the max margin is 1.



b.

Back in the original Euclidian input space, the magin separator is the line **x1 = 0** and **x2 = 0**.

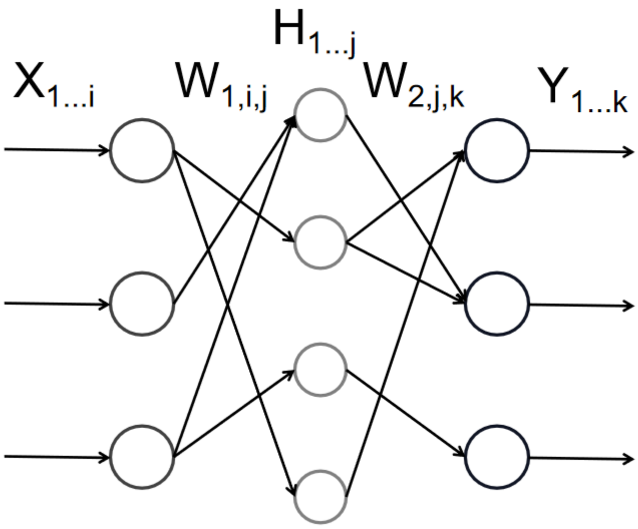


Q.3

Suppose you had a neural network with a linear activation function $g(x) = cx + b$ (c and b are constants):

> a. Assume that the network has one hidden layer. For a given assignment to the weights **w**, write down equations for the value of the units in the output layer as a function of **w** and the input layer **x**, without any explicit mention of the output of the hidden layer. Show that there is a network with no hidden units that computes the same function.
>
> b. Repeat the calculation in part (a), but this time do it for a network with any number of hidden layers.
>
> c. Suppose a network with one hidden layer and linear activation functions has n input and output nodes and h hidden nodes. What effect does the transformation in part (a) to a network with no hidden layers have on the total number of weights? Discuss in particular the case $h \ll n$.

a.



$

$$H_j = g(\sum_i W_{1,i,j}X_i) = c\sum_i W_{1,i,j}X_i + b \tag{1}$$
$$Y_k = g(\sum_j W_{2,j,k}H_j) = c\sum_j W_{2,j,k}H_j + b$$
$$= c\sum_j W_{2,j,k}(c\sum_i W_{1,i,j}X_i + b) + b$$
$$= c^2\sum_j W_{2,j,k}\sum_i W_{1,i,j}X_i + b(c\sum_j W_{2,j,k} + 1)$$

The new network just one layer with $W_{i,k} = \sum_j W_{2,j,k}W_{1,i,j}$ and activation function $g'(x) = c^2x + b(c\sum_j W_{2,j,k} + 1)$.

b.

Using the induction:

For a k-layer network, select the first two layers, which one is input layer I, another is the first hidden layer H_1, after the two layers, the result is O_1. We can use the method in question a to reduce a hidden layer, so that the network becomes a k-1 layer network. After k-1 iteration, the network will be transformed into a single layer neural network.

c.

The weights number will be from $2hn$ to n^2, when h<<n, the network with hidden nodes will store much less weights than reduced network and also can be trained faster.