CS5785 Applied Machine Learning

Homework 4

Disheng Zheng, Daniel Terry

11/20/2017

Written Exercises:

1. Maximum-margin classifiers

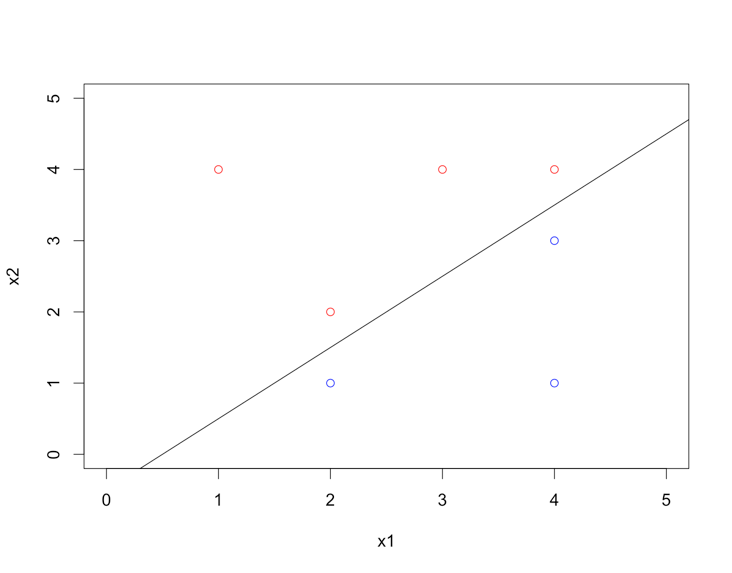


Figure 1. The observations and the maximum-margin separating hyperplane.

The maximum-margin separating hyperplane has to be going exactly through the mid points of blue(2, 1) red(2, 2), and blue(4,3) red(4, 4). The hyperplane has the form: X2 = X1 - 0.5.

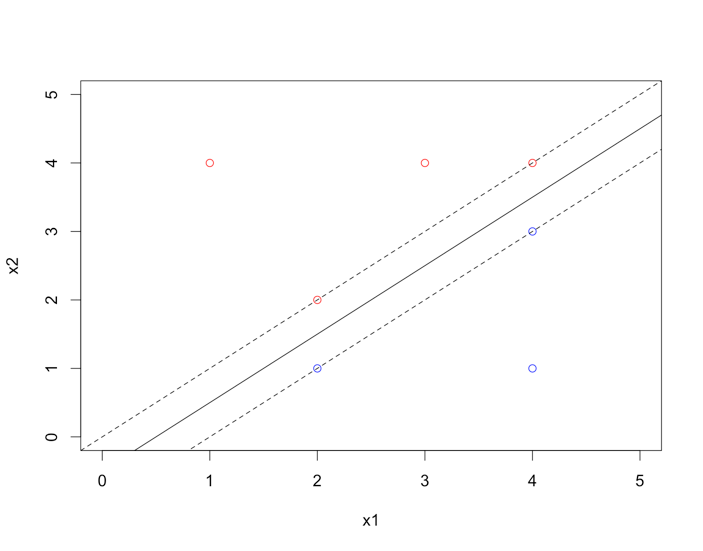
(b)

The classification rule:

|  |  |
| --- | --- |
| Red | X1 – X2 - 0.5 < 0 |
| Blue | Other wise. |
| β1 = 1 ; β2 = -1; β0 = -0.5 |  |

(c)

The margin of maximum margin plane is the zone where point fall in it won’t be able to be classified. The zone is between blue(2, 1) red(2, 2), and blue(4,3) red(4, 4).



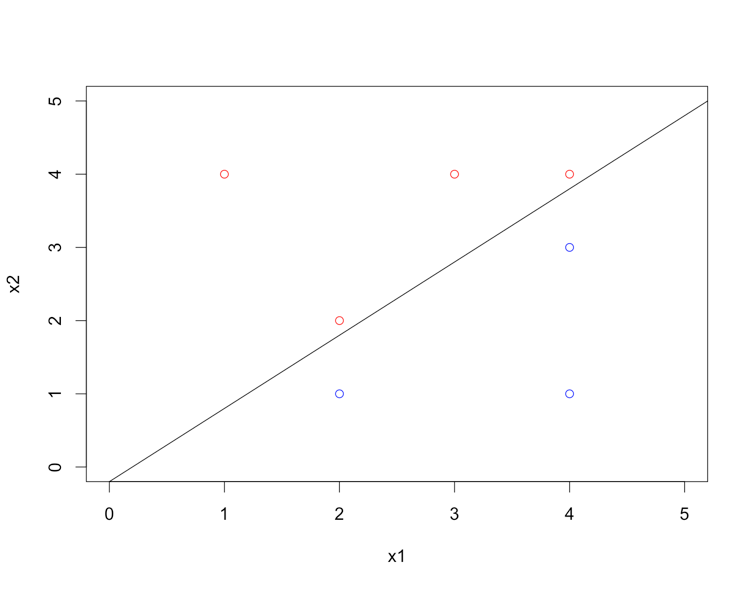
(d)

The support vectors are points blue(2, 1) red(2, 2), and blue(4,3) red(4, 4).

(e)

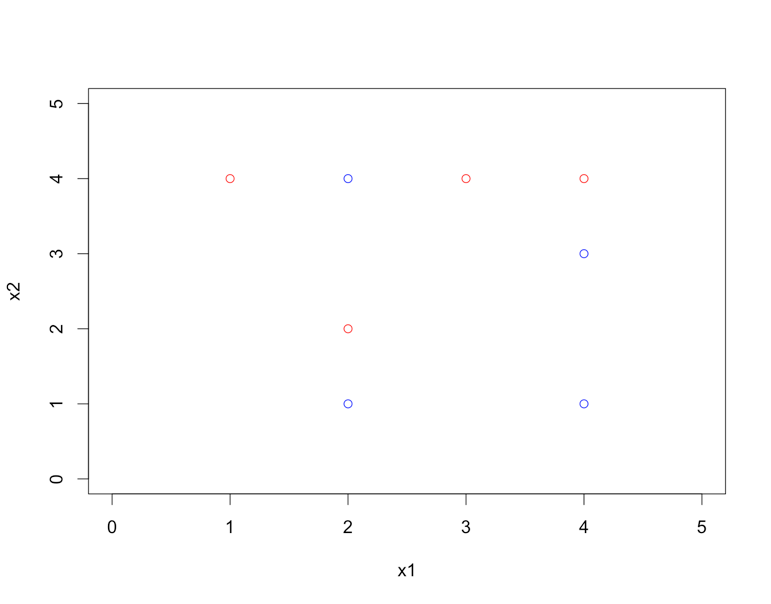
The seventh observation is blue(4, 1). Slight movements will not affect the hyperplane because it is nowhere near the margin. It is not a support vector.

(f)



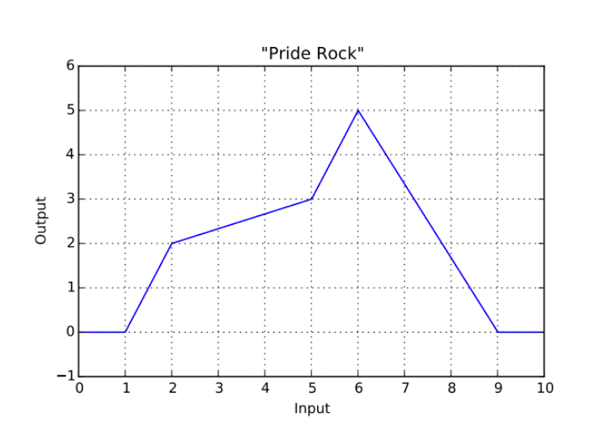
This is a hyper plane with equation of: X2 = X1 - 0.2. It separates the data correctly but it is not a maximum-margin separating hyperplane. It is close to red support vectors and father away from blue support vectors. Points have higher chance of been misclassified as blue.

(g)



I added a blue point (2, 4). In this case, two classed are no longer separable by a hyperplane. There is no linear straight line that could divide blues and reds.

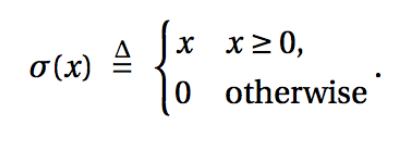
2. Neural Networks as Function Approximators.



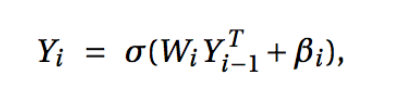
We notice this function looks like:

|  |  |
| --- | --- |
| y = |  |
| 0 | x ∈ [0, 1) |
| 2x -2 | x ∈ [1, 2) |
| () x + | x ∈ [2, 5) |
| 2x - 7 | x ∈ [5, 6) |
| -x + 15 | x ∈ [6, 9) |
| 0 | x ∈ [9, 10] |

And notice that the Relu function looks like:



Each layer has the form:



Y = σ(W1Y + β1) = 2Y – 2

W1  = 2, β1 = -2

Y = σ(W2(2Y – 2) + β2) = () Y +

2W2Y - 2W2 + β2 = () Y +

W2  = , β2 =

Y = σ(W3(Y + ) + β3) = 2Y – 7

W3Y + W3 + β3 = 2Y – 7

W3  = 12, β3 = -27

Y = σ(W4(12Y – 27) + β4) = -Y + 15

12W4Y - 27 W3 + β4 = -Y + 15

W4  = -, β4 = 11.25

The neural network has 1 hidden layer and 4 units with weights and biases:

|  |  |
| --- | --- |
| W1  = 2 | β1 = -2 |
| W2  = | β2 = |
| W3  = 12 | β3 = -27 |
| W4  = - | β4 = 11.25 |

Programming Exercises:

1.

(a) The default neural network has 7 layers with 20 neurons each. And it has an input layer and an output layer.

(b) The loss here means the weighted sum of the current lost and the loss of the past iteration. Based on the source code of ConvNet, the loss function is:

smooth\_loss = 0.99 \* smooth\_loss + 0.01 \* batch\_loss [1]

(c)

(d)

(e)

I dropped the layers one by one, and noticed that I can drop 3 layers at the most before quality drops noticeably.

(f)

I tried to add 3, 5, 10, and 20 layers in the neural network and did not notice an increase in accuracy.

2.

(a)

Image Mona Lisa



(b)

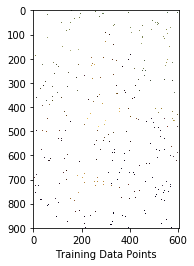
We uniformly randomly select 5000 (x, y) coordinate points.

No other preprocessing steps are needed since it has no effects on decision tree.

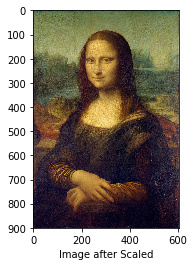
(c)

I mapped all 3 r, g, b values using f: R2 🡪 R3

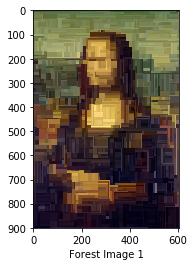
And the mapped points look like:



(d) Here is the image after scaling from 255 to 1.



And When I run the random forest with depth None and get the predicted graph



(e)

(i) Observe that when start with depth 1, the graph is divided into 2. The more the depth we set, the richer the picture gets. It is getting more and more distinct parts (like pixels).

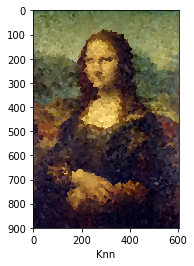
|  |  |  |
| --- | --- | --- |
| /Users/disheng/Desktop/Machine Learning/hw4/depths/download.png |  |  |
|  |  |  |

(ii) Every tree is different at making predictions. So, the more the trees, the variances can be reduced. Therefore, the more trees we get, the smoother the image.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

(iii)

Using a regressor of Knn with k = 1



Knn is to assign its nearest pixel with the same color. Therefore, unlike random forest, we end up with a blurred graph with no straight-line boundaries.

(iv)

There is no necessity of pruning trees. Our sample size is 5000, we are using 100 trees.

(f)

(i) The decision rule at the split point of each tree takes input at a position (x , y) and output a prediction (x, y) for the next tree split. In the end, when a leaf node is reached it out puts an r, g, b color. Here is the formula:

Split = {left branch of tree | x >= threshold}

{right branch of tree | x < threshold}

(ii) Random forest only has as many color as the number of the leaf nodes, so the predictions look like low resolution. The classifiers are assigned with straight lines; thus, the graphs are divided into overlapping rectangles.

Knn outputs a blurred graph, where each point color prediction is based on its nearest neighbor. It looks like a picture with low resolution.

(iii) For only one tree, each leaf node has a color. There are 2depth number of leaf nodes and colors.

(iv) If the forest contains n trees. is the number of colors possible. for 1 tree there are 2depth number of leaves. For n trees, there are n \* 2depth choose n ways to assign a color.

References

[1] <https://github.com/pranv/ARC/blob/master/main.py>