Assignment 4: Decision Making under Uncertainty and Learning

Alex Smirnov, Scott Reyes

May 1, 2017

Question 1:

Note: States are referred by their index rather than their state number.
Iteration: 0

Node	Utility
0	0.0
1	0.0
2	1.0
3	0.0

Iteration: 1

Action	Node	Utility
2	0	0.0
2	1	0.7200000000000001
3	2	1.6480000000000001
1	3	1.33488

Iteration: 5

Action	Node	Utility
2	0	2.337017421689511
2	1	2.6655934354214406
3	2	3.4750243795365288
1	3	3.0622721853782413

Iteration: 15

Action	Node	Utility
2	0	3.713231422293591
2	1	4.176962967443245
3	2	4.816810731150936
1	3	4.283297765347352

Iteration: 60

Action	Node	Utility
2	0	3.929362496465146
2	1	4.414469642322995
3	2	5.027597905122635
1	3	4.475114093661598

Iteration: 150

Action	Node	Utility
2	0	3.9293895701383725
2	1	4.414499393612255
3	2	5.027624309391853
1	3	4.4751381215465855

The implementation used to code value iteration was the Bellman equation. The equation iteratively updates the utility of a node via the maximum of the utility of the best available actions. This value is then discounted and added to the reward before being stored as the next iteration's utility value. The convergence criteria was comparing the maximum change in utility of the current iteration to a maximum error multiplied by the complement of the discount factor expressed in terms of the discount factor. This is essentially to keep iterating until we reach some level of precision in terms of converging with the actual utility. When value iteration is called on the graph with the arguments discount = 0.9, and maxError = .000000000000001, the method iterates 177 times until it reaches convergence. Increasing the discount factor increases the number of iterations needed to reach convergence, and increasing the maximum error allowed deceases the number of iterations needed to reach convergence. The time it takes to call the function on a computer with a CPU: i7-4770k will be around 2 milliseconds for 177 iteration when nothing is printed to the terminal.

Question 2:

a

b

c

Question 3:

a

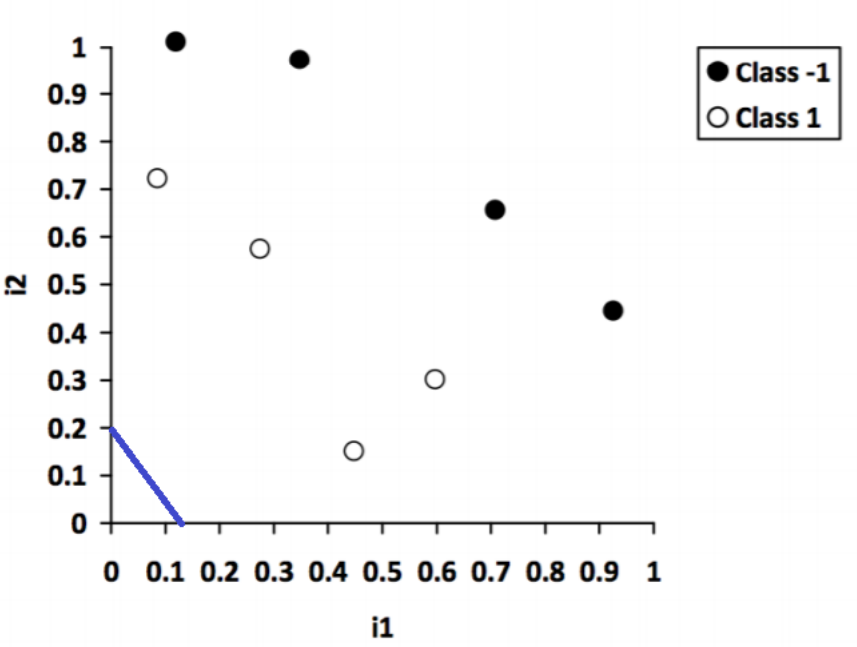
b

c

Question 4:

a

Constant offset = 1
Class -1 Inputs:
(0.1, 1), (0.35, 0.95), (0.7, 0.65), (0.9, 0.45)
Class 1 Inputs:
(0.1, 0.7), (0.3, 0.55), (0.45, 0.15), (0.6, 0.3)
Initial Weights:
$w_0 = 0.2$
$w_1 = 1$
$w_2 = -1$
$y = (0.2 * 1) + ((0.1 * -1) + (0.35 * -0.95) + (0.7 * -0.65) + (0.9 * -0.45) + (0.1 * -0.7) + (0.3 * -0.55) + (0.45 * -0.15) + (0.6 * -0.3))x$
$y = 0.2 + (-0.1 - 0.3325 - 0.455 - 0.405 - 0.07 - 0.165 - 0.0675 - 0.18)x$
$y = -1.775x + 0.2$



4 samples are misclassified after the initial line of separation is placed. Class 1 input (0.1, 0.7) is misclassified, so the weights will be adjusted accordingly.

$w_0 = 1$

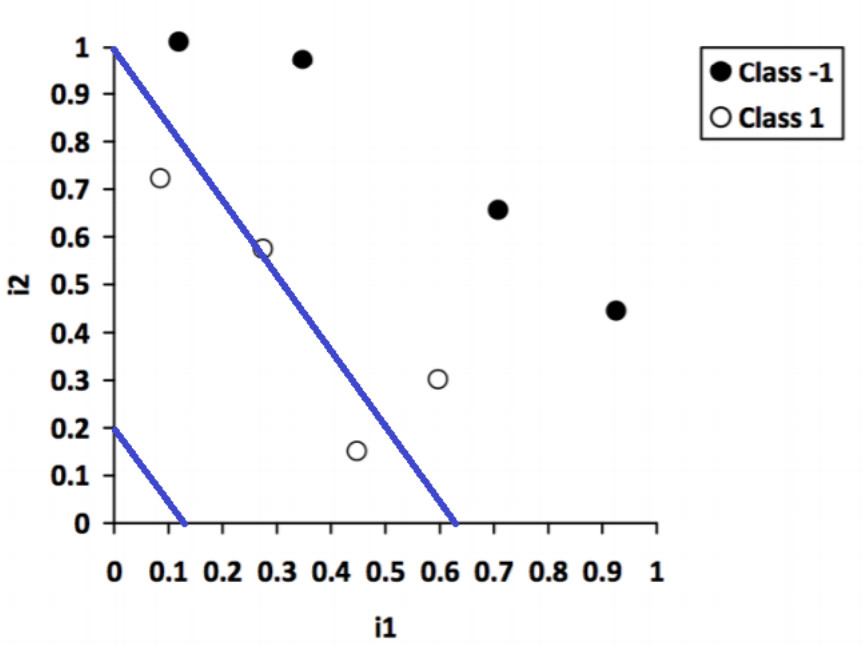
$w_1 = 1$

$w_2 = -1$

$y = (1 * 1) + ((0.1 * -1) + (0.35 * -0.95) + (0.7 * -0.65) + (0.9 * -0.45) + (0.1 * -0.7) + (0.3 * -0.55) + (0.45 * -0.15) + (0.6 * -0.3))x$

$y = 1 + (-0.1 - 0.3325 - 0.455 - 0.405 - 0.07 - 0.165 - 0.0675 - 0.18)x$

$y = -1.775x + 1$



2 samples are misclassified after the second line is placed. Class 1 input (0.3, 0.55) is misclassified, so the weights will be adjusted accordingly.

$w_0 = 1$

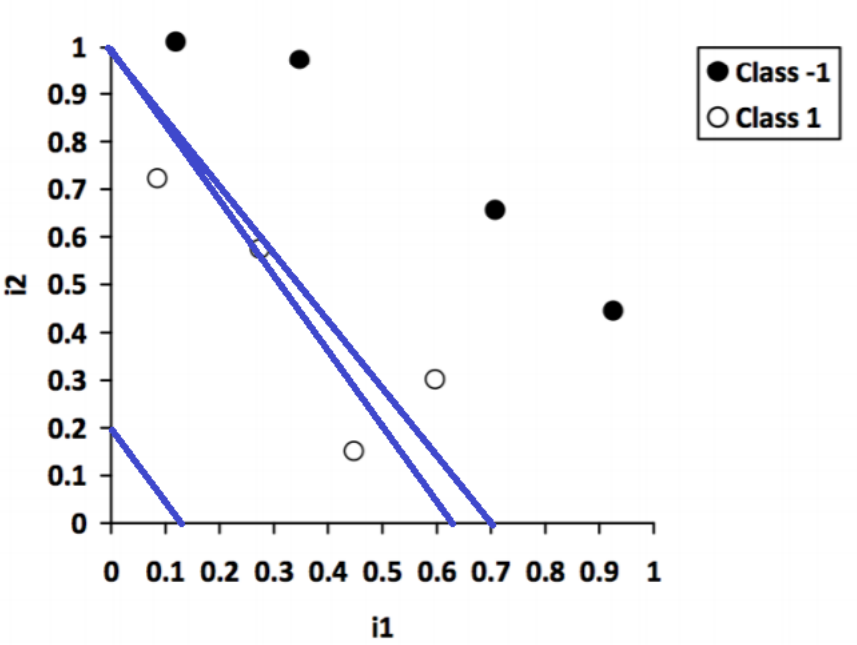
$w_1 = 0.8$

$w_2 = -1$

$y = (1 * 1) + ((0.08 * -1) + (0.28 * -0.95) + (0.56 * -0.65) + (0.72 * -0.45) + (0.08 * -0.7) + (0.24 * -0.55) + (0.36 * -0.15) + (0.48 * -0.3))x$

$y = 1 + (-0.08 - 0.266 - 0.364 - 0.324 - 0.056 - 0.132 - 0.054 - 0.144)x$

$y = -1.42x + 1$



1 sample is misclassified after the third line is placed. Class 1 input (0.6, 0.3) is misclassified, so the weights will be adjusted accordingly.

$w_0 = 1$

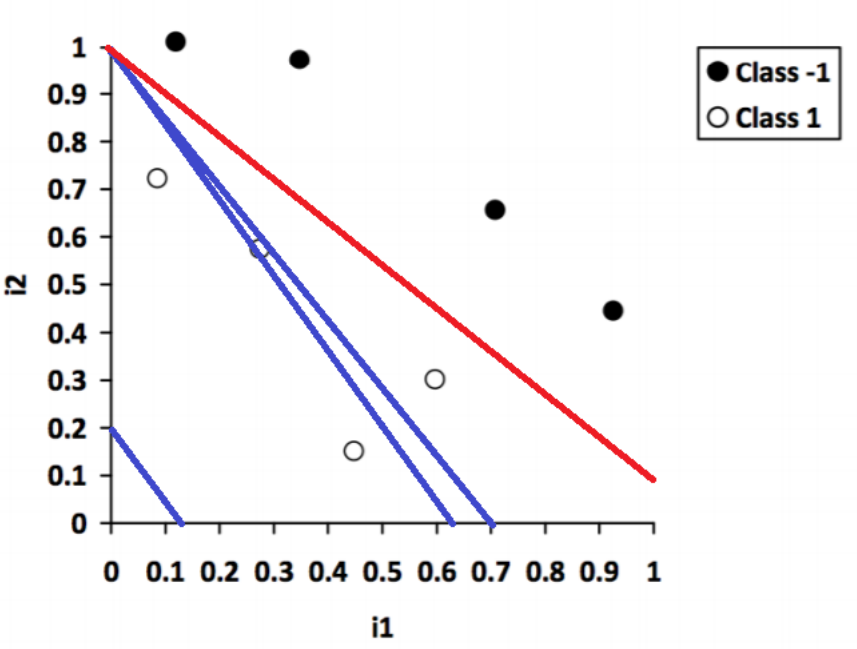
$w_1 = 0.5$

$w_2 = -1$

$y = (1 * 1) + ((0.05 * -1) + (0.175 * -0.95) + (0.35 * -0.65) + (0.45 * -0.45) + (0.05 * -0.7) + (0.15 * -0.55) + (0.225 * -0.15) + (0.3 * -0.3))x$

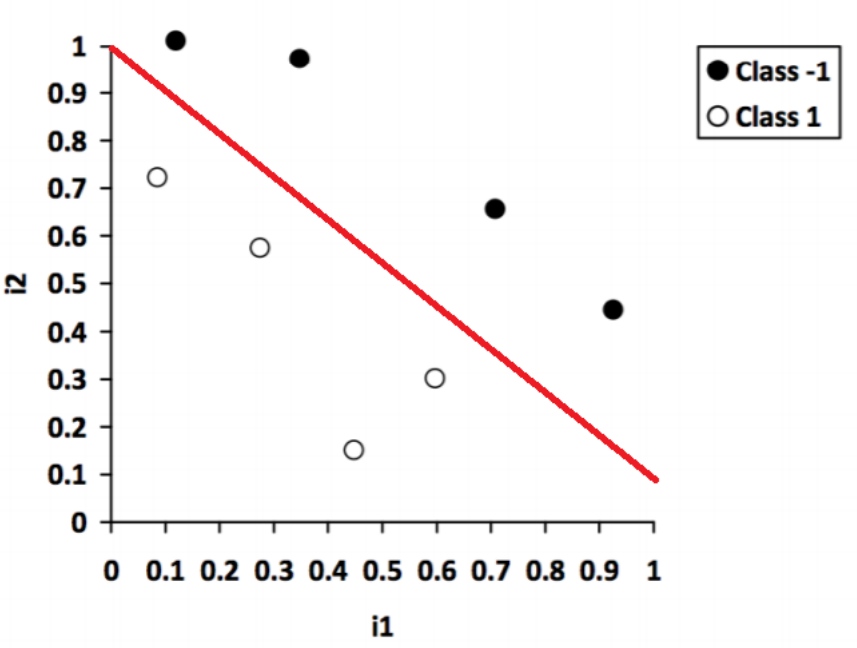
$y = 1 + (-0.05 - 0.16625 - 0.2275 - 0.2025 - 0.035 - 0.0825 - 0.03375 - 0.09)x$

$y = -0.8875x + 1$



No samples are misclassified after the fourth line is placed.

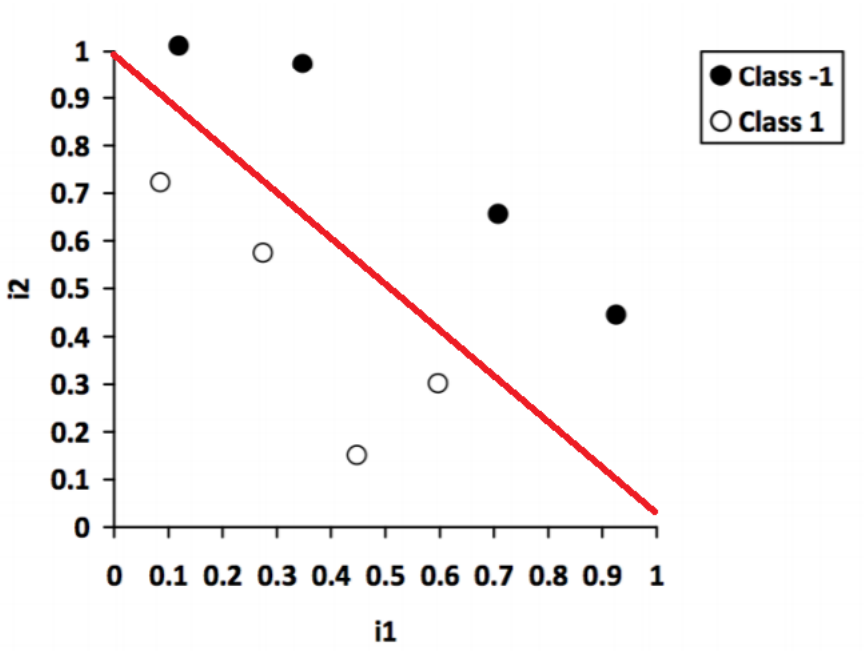
b



This is the final line that achieved perfect classification.
$w_0 = 1$
$w_1 = 0.5$
$w_2 = -1$
$y = -0.8875x + 1$

c

Constant offset = 1
Class -1 Inputs:
(0.1), (0.35), (0.7), (0.9)
Class 1 Inputs:
(0.1), (0.3), (0.45), (0.6)
Initial Weights:
$w_0 = 1$
$w_1 = -0.25$
$y = (1 * 1) + (-0.025 - 0.0875 - 0.175 - 0.225 - 0.025 - 0.075 - 0.1125 - 0.15)x$
$y = -0.875 + 1$

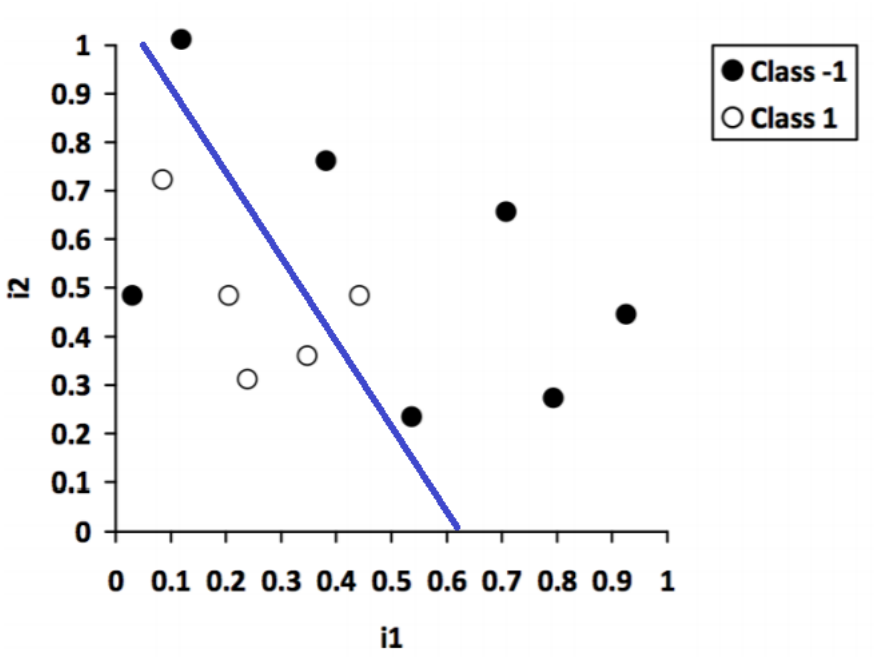


The error for this input space separation is 0 because all of the inputs are correctly classified.

Question 5:

a

The minimum error that can be reached with a single perceptron for this classification task is 2, which means 2 points will be classified incorrectly no matter how much learning the perceptron does.



b

To correctly classify all of these point, we need at least 3 linear separators and one node in the next layer to combine the output of those 3 linear separators. This final node accepts the last two perceptrons as given before inverting the third one' class to be 1. Thus a positive signal will be output only when the first perception returns a class -1 and the last two return a class 1. the linear separator is given in the form of
$w_1 * x_1 + w_2 * x_2 + w_3 * x_3 = 0$

Layer	Node	w1	w2	w3
1	0	-.8	1	.5
1	1	0	-1	.9
1	2	-.8	1	3
2	3	-1	1	1

Final result of the multilayer perceptron line drawing: