Chapter 10: Widrow-Hoff Learning

Brandon Morgan

1/18/2021

E10.6

We are wanting to classify two groups of patterns:

Class 1

$$p_1^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_1 = 1$$

$$p_2^T = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 1$$

Class 2

$$p_3^T = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = -1$$

$$p_4^T = \begin{bmatrix} -4 \\ 1 \end{bmatrix}, t_4 = -1$$

1

This part was skipped as I was unable to create a diagram.

2

```
LMS = function(init, bias, learning_rate, input, expect, maxIter, tol, mask, useBias) {

  n = ncol(input)
  w = init
  b = bias

  for(i in 1:maxIter) { # loop over iterations

    if(!mask) { # if the user does want to print out statements
      print(sprintf("     Iteration %d", i))
  }

  for(j in 1:n) { # loop over input vectors

    if(useBias) { # does the user want to include a bias
      a = w%*%input[,j]+b
    }
    else {
      a = w%*%input[,j]
    }


    e = expect[j] - a

    w = w + 2*learning_rate*e%*%t(input[,j])

    if(useBias) {
      b = b + 2*learning_rate*e
    }

    if(!mask) { # if the user does want to print out statements
      print(sprintf("INPUT VECTOR: %d", j))
      print("a value: ")
      print(a)
      print("error value: ")
      print(e)
      print("new weight value: ")
      print(w)
      print("new bias value: ")
      print(b)
    }
  }


  error = 0

  for(j in 1:n) {

    if(useBias) { # does the user want to include a bias
      a = w%*%input[,j]+b
    }
    else {
      a = w%*%input[,j]
    }

    e = expect[j] - a

    error = error + e^2

  }

  # if the square root of the sum of the square
  # errors is greater than the tol, then algo
```

```
    # has not converged
    converged = 1
    if(sqrt(error)>tol) {
      converged = 0
    }

    if(converged) {
      print("    Algorithm CONVERGED:")
      print("Current weight is: ")
      print(w)
      if(useBias) { # does the user want to include a bias
        print("Current bias is: ")
        print(b)
      }

      print(sprintf("Iterations taken: %d", i))
      if(useBias) { # does the user want to include a bias
        return(list(w=w, b=b))
      }
      return(list(w=w))
    }


  }

  error = 0

  for(j in 1:n) {

    a = w%*%input[,j]+b

    e = expect[j] - a

    error = error + abs(e)

  }

  print("MAXIMUM ITERATIONS REACHED")
  print("Current weight is: ")
  print(w)
  if(useBias) { # does the user want to include a bias
        print("Current bias is: ")
        print(b)
  }
  print(sprintf("Iterations taken: %d", i))
  print("ERROR: ")
  print(error)
  if(useBias) { # does the user want to include a bias
        return(list(w=w, b=b))
  }
  return(list(w=w))

}
```

Now we will run the algorithm four steps, one iteration, with the zero vector for the initial guess, no bias, and a learning rate of $\alpha = 0.1$:

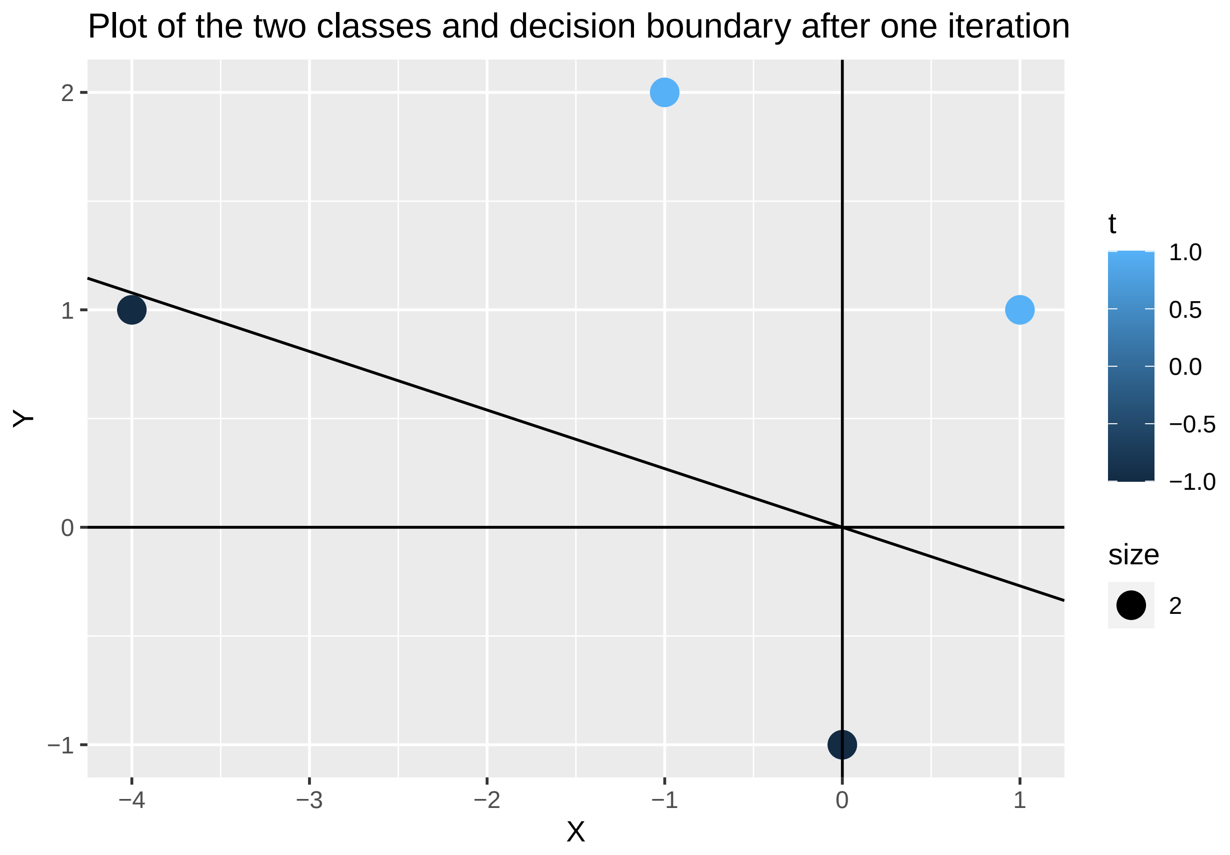
```
init = matrix(c(0, 0), ncol=2)
input = matrix(c(1, 1, -1, 2, 0, -1, -4, 1), ncol=4, byrow=FALSE)
expected=c(1, 1, -1, -1)
val = LMS(init, 0, 0.10, input, expected, 1, 1e-10, 1, 0)
```

```
## [1] "MAXIMUM ITERATIONS REACHED"
## [1] "Current weight is: "
##        [,1]   [,2]
## [1,] 1.2048 0.3248
## [1] "Iterations taken: 1"
## [1] "ERROR: "
##        [,1]
## [1,] 6.2544
```

Here we get the following weights $w = [1.2048, 0.3248]$, thus we get the following boundary decision $p_1 = \frac{-0.3248}{1.2048}p_2$:

```
library(ggplot2) # used for plots
```

```
data =data.frame(p1=c(1, -1, 0, -4), p2=c(1, 2, -1, 1), t=c(1, 1, -1, -1))
ggplot(data=data, aes(x=p1, y=p2, color=t))+geom_point(aes(size=2))+xlab("X")+
  ylab("Y")+ggtitle("Plot of the two classes and decision boundary after one iteration")+
  geom_hline(yintercept=0)+geom_vline(xintercept=0)+
  geom_abline(intercept=0, slope=-0.3248/1.2048)
```



3

The optimal weights are given by $x^* = R^{-1}h$, where $R = E[zz^T]$ and $h = E[tz]$. Our performance index can be written as $F(x) = c - 2x^Th + x^TRx$, where $c = E[t^2]$.

Assuming equal probability,

$$c = E[t^2] = (1)^2(0.5) + (1)^2(0.5) + (-1)^2(0.5) + (-1)^2(0.5) = 2$$

$$h = E[tz] = 0.5(1)\begin{bmatrix}1\\1\end{bmatrix} + 0.5(1)\begin{bmatrix}-1\\2\end{bmatrix} + 0.5(-1)\begin{bmatrix}0\\-1\end{bmatrix} + 0.5(-1)\begin{bmatrix}-4\\1\end{bmatrix} = \begin{bmatrix}1.5\\1.5\end{bmatrix}$$

$$R = E[zz^T] = 0.5(\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1 & 1\end{bmatrix} + \begin{bmatrix}-1\\2\end{bmatrix}\begin{bmatrix}-1 & 2\end{bmatrix} + \begin{bmatrix}0\\-1\end{bmatrix}\begin{bmatrix}0 & -1\end{bmatrix} + \begin{bmatrix}-4\\1\end{bmatrix}\begin{bmatrix}-4 & 1\end{bmatrix}) = \begin{bmatrix}9 & -2.5\\-2.5 & 3.5\end{bmatrix}$$

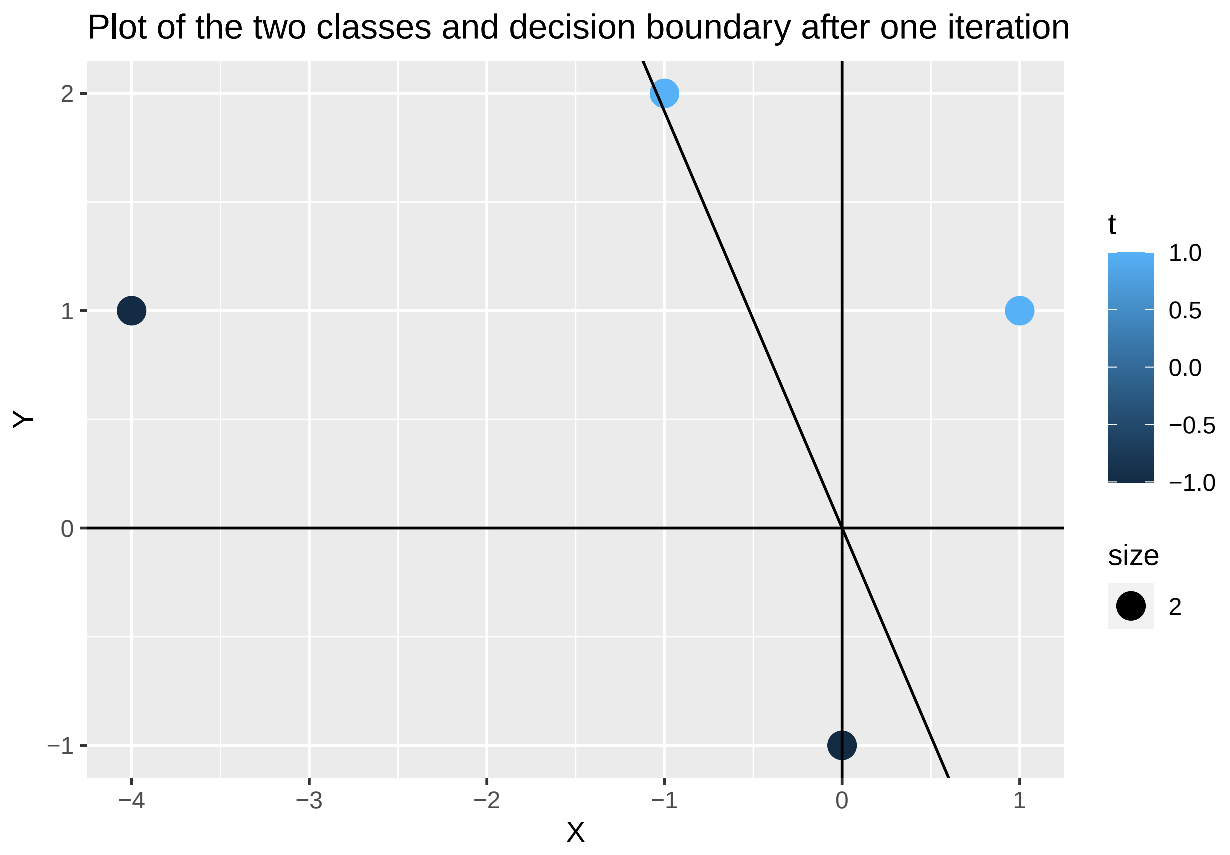
Thus, our optimal weight vector is:

$$x^* = \begin{bmatrix}9 & -2.5\\-2.5 & 3.5\end{bmatrix}^{-1}\begin{bmatrix}1.5\\1.5\end{bmatrix} = \begin{bmatrix}0.35642\\0.68315\end{bmatrix}$$

4

Here we have the following optimal weight vector,$w = [0.35642, 0.68315]$, which corresponds to the following decision boundary $p_1 = \frac{-0.68315}{0.35642} p_2$:

```
data =data.frame(p1=c(1, -1, 0, -4), p2=c(1, 2, -1, 1), t=c(1, 1, -1, -1))
ggplot(data=data, aes(x=p1, y=p2, color=t))+geom_point(aes(size=2))+xlab("X")+
  ylab("Y")+ggtitle("Plot of the two classes and decision boundary after one iteration")+
  geom_hline(yintercept=0)+geom_vline(xintercept=0)+
  geom_abline(intercept=0, slope=-0.683145/0.356415)
```



5

The boundary would change for the better if bias was included as it might minimize the least square error better than without bias. By looking at our graph, the input vectors are not symmetric, and one vetor is furthur away from the orign that the others, $p = [-4, 1]$; thus, we can hypothesis that no decision boundary through the origin can discriminate between these two clusters perfectly.

In fact, if we were run our algorithm for 1000 iterations, 4000 steps, it sill does not converge:

```
val = LMS(init, 0, 0.10, input, expected, 1000, 1e-10, 1, 0)
```

```
## [1] "MAXIMUM ITERATIONS REACHED"
## [1] "Current weight is: "
##            [,1]      [,2]
## [1,] 0.4756618 0.7591802
## [1] "Iterations taken: 1000"
## [1] "ERROR: "
##            [,1]
## [1,] 0.6618275
```

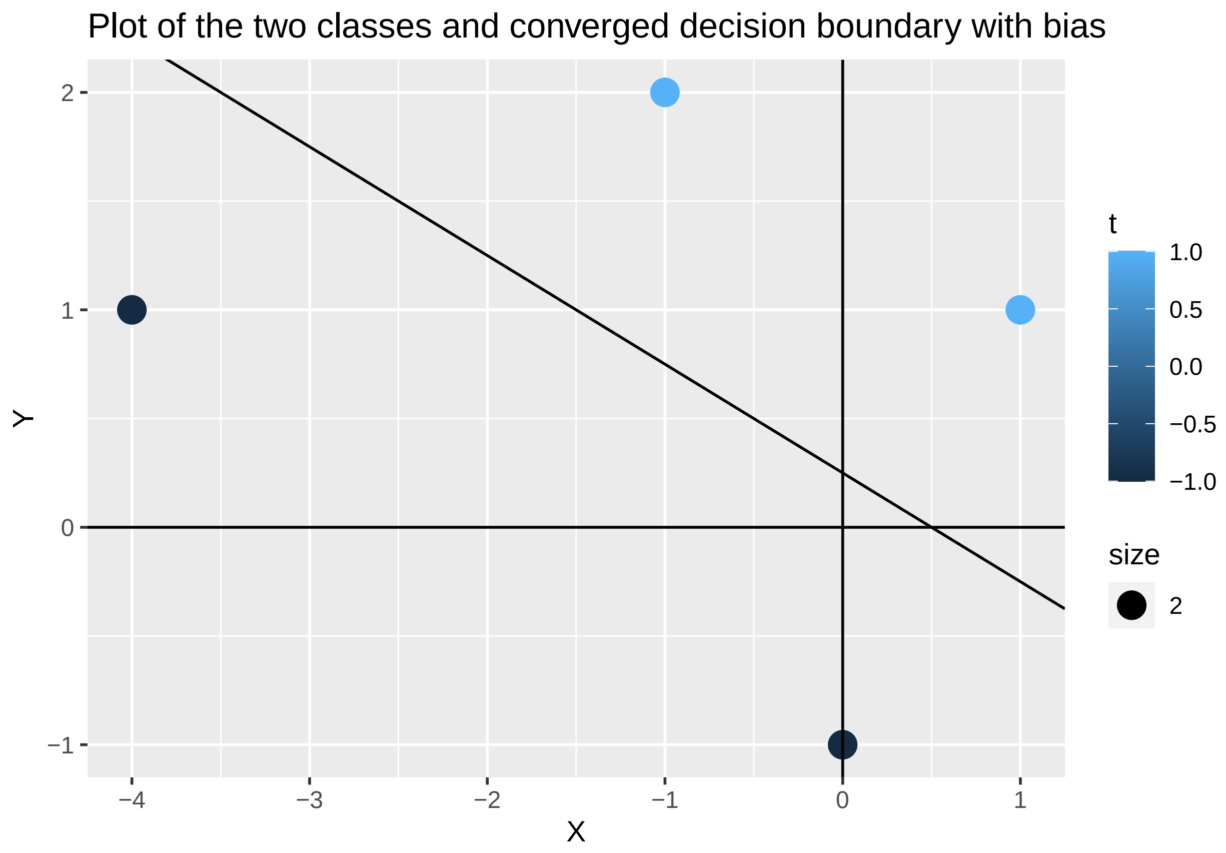
However, here we will run our algorithm again, but this time with bias and ran until convergence:

```r
val = LMS(init, 0, 0.10, input, expected, 100, 1e-10, 1, 1)
```

```
## [1] "    Algorithm CONVERGED:"
## [1] "Current weight is: "
##      [,1] [,2]
## [1,]  0.4  0.8
## [1] "Current bias is: "
##      [,1]
## [1,] -0.2
## [1] "Iterations taken: 42"
```

Our algorithm converged with weight vector $w = [0.4, 0.8]$ and bias $b = -0.2$. Therefore, our decision boundary would be $p_2 = 0.25 - 0.5p_1$

```r
data =data.frame(p1=c(1, -1, 0, -4), p2=c(1, 2, -1, 1), t=c(1, 1, -1, -1))
ggplot(data=data, aes(x=p1, y=p2, color=t))+geom_point(aes(size=2))+xlab("X")+
  ylab("Y")+ggtitle("Plot of the two classes and converged decision boundary with bias ")+
  geom_hline(yintercept=0)+geom_vline(xintercept=0)+
  geom_abline(intercept=0.25, slope=-0.5)
```


Plot of the two classes and converged decision boundary with bias
Y
X
2
1
0
−1
−4
−3
−2
−1
0
1
t
1.0
0.5
0.0
−0.5
−1.0
size
2