Chapter 7: Supervised Hebbian Learning

Brandon Morgan

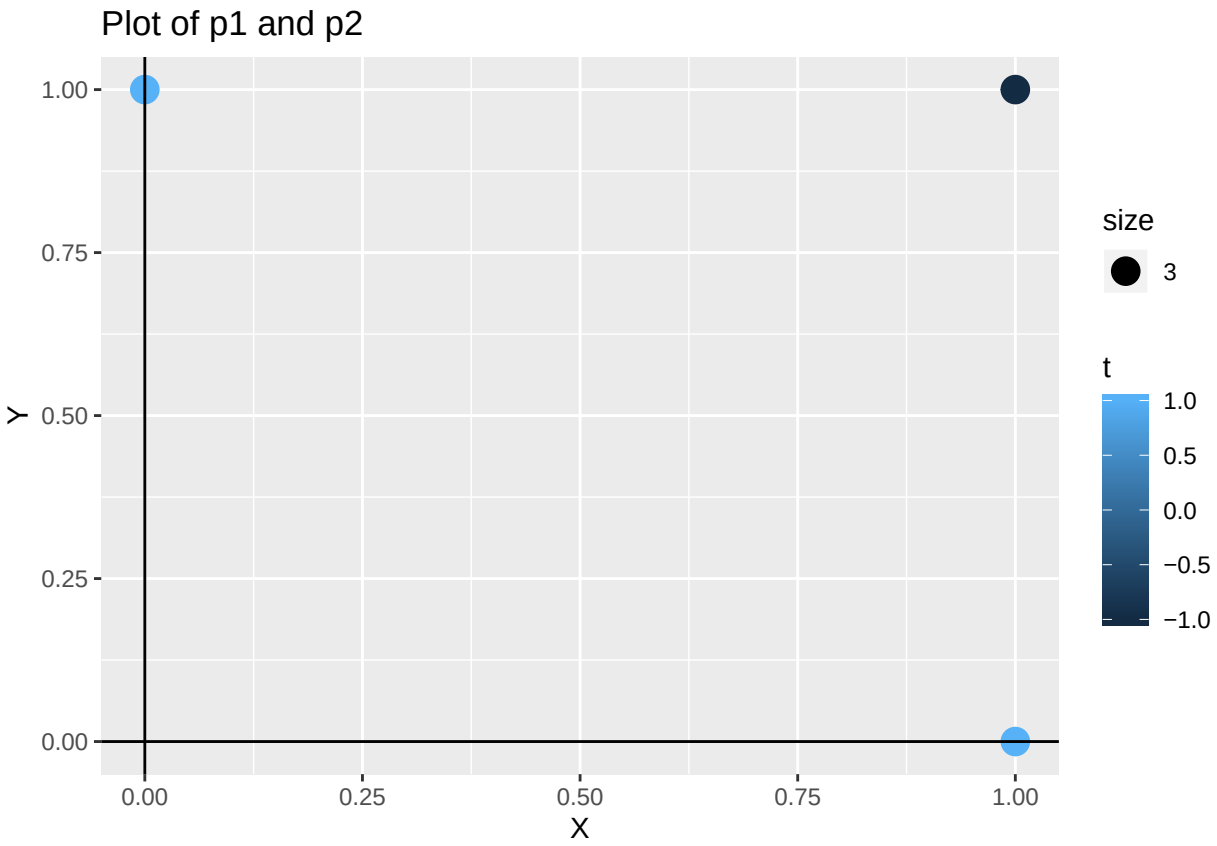
1/13/2021

E7.6

1

We can see from the plot below of the three input vectors colored by their expected t_i value, no linear discriminator centered about the origin can classify these two groups. Therefore, a y-intercept, bias, is needed to classify these two groups.

```
library(ggplot2)
data = data.frame(v1=c(1, 1, 0), v2=c(0, 1, 1), t=c(1, -1, 1))
ggplot(data=data, aes(x=v1, y=v2, color=t)) + geom_point(aes(size = 3))+xlab("X")+
  ylab("Y")+ggtitle("Plot of p1 and p2")+geom_hline(yintercept=0)+
  geom_vline(xintercept=0)
```



2

We can include bias into our calculations of the pseudoinverse rule by use of augmented vectors, i.e., including it as an extra term. Now, $p_1^t = [1, 0, 1]$, $p_2^t = [1, 1, 1]$, and $p_3^t = [0, 1, 1]$.

```
T = matrix(c(1, -1, 1), ncol=3)
p1 = matrix(c(1, 0, 1), ncol=1)
p2 = matrix(c(1, 1, 1), ncol=1)
p3 = matrix(c(0, 1, 1), ncol=1)
P = matrix(c(p1, p2, p3), ncol=3)
P
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    0
## [2,]    0    1    1
## [3,]    1    1    1
```

Now, we can use the pseudoinverse rule $W = TP^{+}$; however, because the number of rows for P is less than the number of columns, before augmenting, $P^{+} = P^{t}(PP^{t})^{-1}$

Then, P^{+} becomes:

```
pseudo = t(P)%*%solve(P%*%t(P))
pseudo
```

```
##      [,1]          [,2] [,3]
## [1,]    0 -1.000000e+00    1
## [2,]    1  1.000000e+00   -1
## [3,]   -1  2.220446e-16    1
```

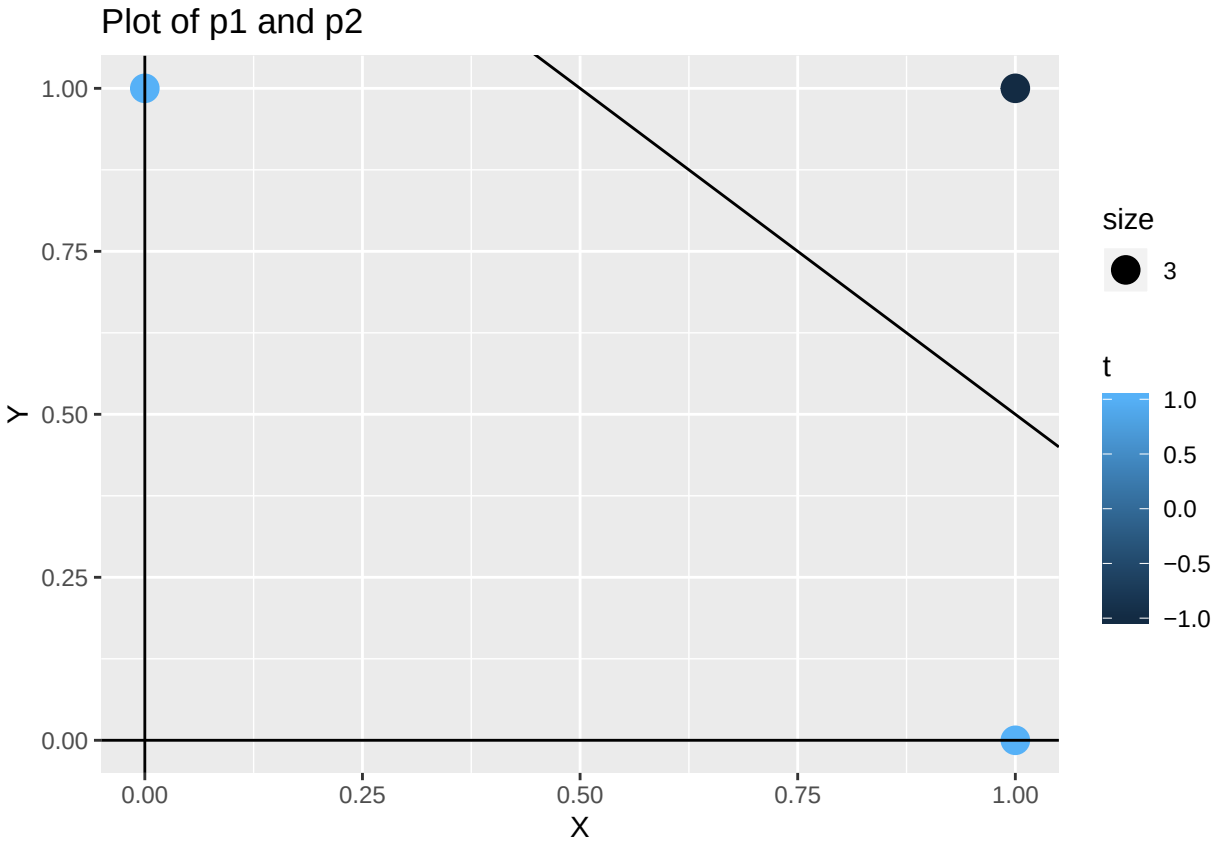
Now our weight matrix can be calculated:

```
W=T%*%(pseudo)
W
```

```
##      [,1] [,2] [,3]
## [1,]   -2   -2    3
```

From these we get the following decision boundary: $-2p_1 - 2p_2 + 3 = 0$, from $Wp + b = 0$. Therefore, $p_2 = \frac{3}{2} - p_1$.

```
ggplot(data=data, aes(x=v1, y=v2, color=t)) + geom_point(aes(size = 3))+xlab("X")+
  ylab("Y")+ggtitle("Plot of p1 and p2")+geom_hline(yintercept=0)+
  geom_vline(xintercept=0)+geom_abline(intercept=1.5, slope=-1)
```



As one can see, our decision boundary correctly classifies the two groups.

Side Note

I find it interesting how this question makes us use the pseudoinverse rule to design the network when the augmented P matrix is square and independent, therefore P^{-1} exists. If one were to use $W = TP^{-1}$ instead to train the weights:

```
W=T%*%solve(P)
W
```

```
##      [,1] [,2] [,3]
## [1,]   -2   -2    3
```

Which is the same as created from the pseudoinverse rule, therefore the same boundary will be created.