# Module2_Lab1

Andrew Estes

3/25/2022

```
library(tidyverse)
```

## 1

Randomly generate a data set that is linearly separable. Plot the points and color code them by group.

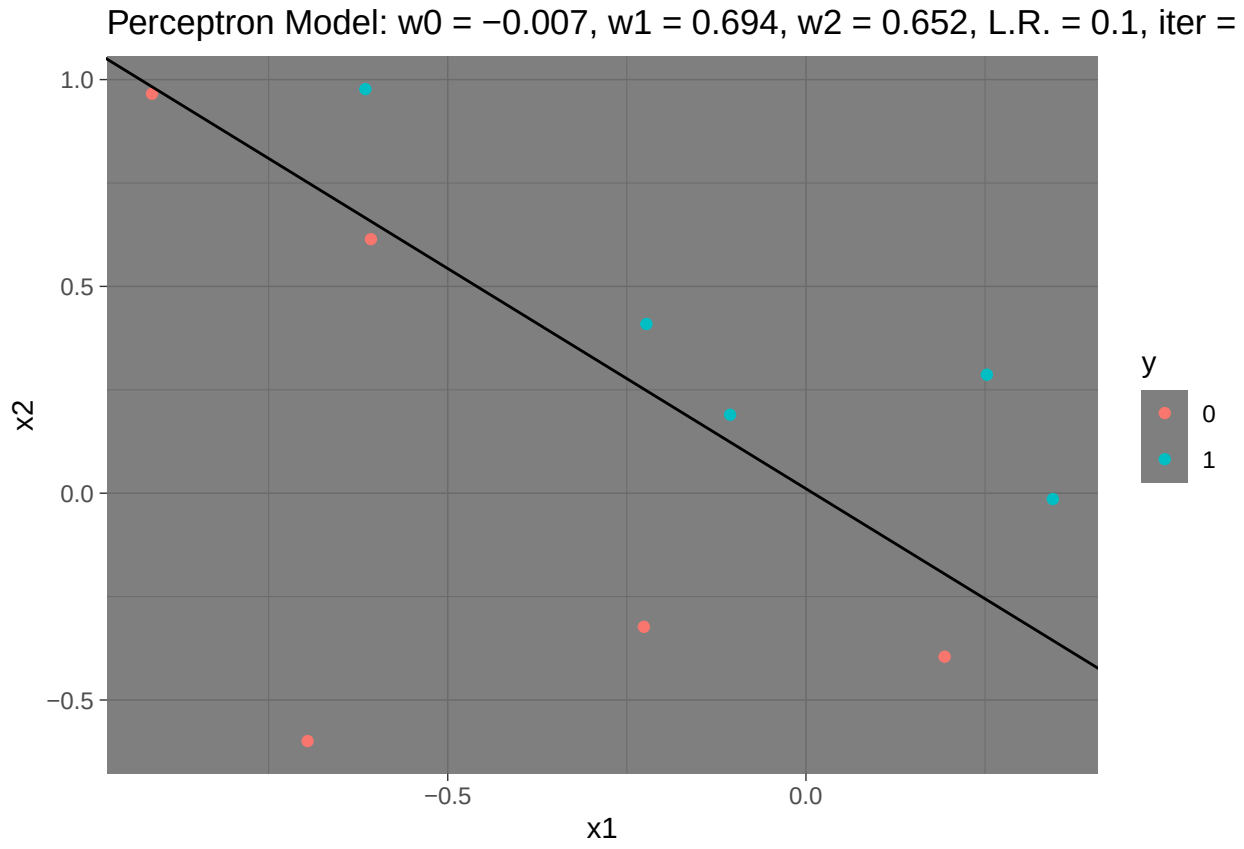
```
#first step is to create the dataframes
weight <- runif(2)

#creating the larger dataframe
large.df <- data.frame(
  x1 = runif(100, min=-1, max=1),
  x2 = runif(100, min=-1, max=1),
  y = numeric(100)
)

for(i in 1:nrow(large.df)){
  large.df$y[i] <- if_else(weight %*% as.numeric(large.df[i, 1:2]) >= 0, 1, 0)

}

#creating the smaller dataframe
small.df <- data.frame(
  x1 = runif(10, min=-1, max=1),
  x2 = runif(10, min=-1, max=1),
  y = numeric(10)
)

for(i in 1:nrow(small.df)){
  small.df$y[i] <- if_else(weight %*% as.numeric(small.df[i, 1:2]) >= 0, 1, 0)

}
```

The second step is to create the perceptron plotting function

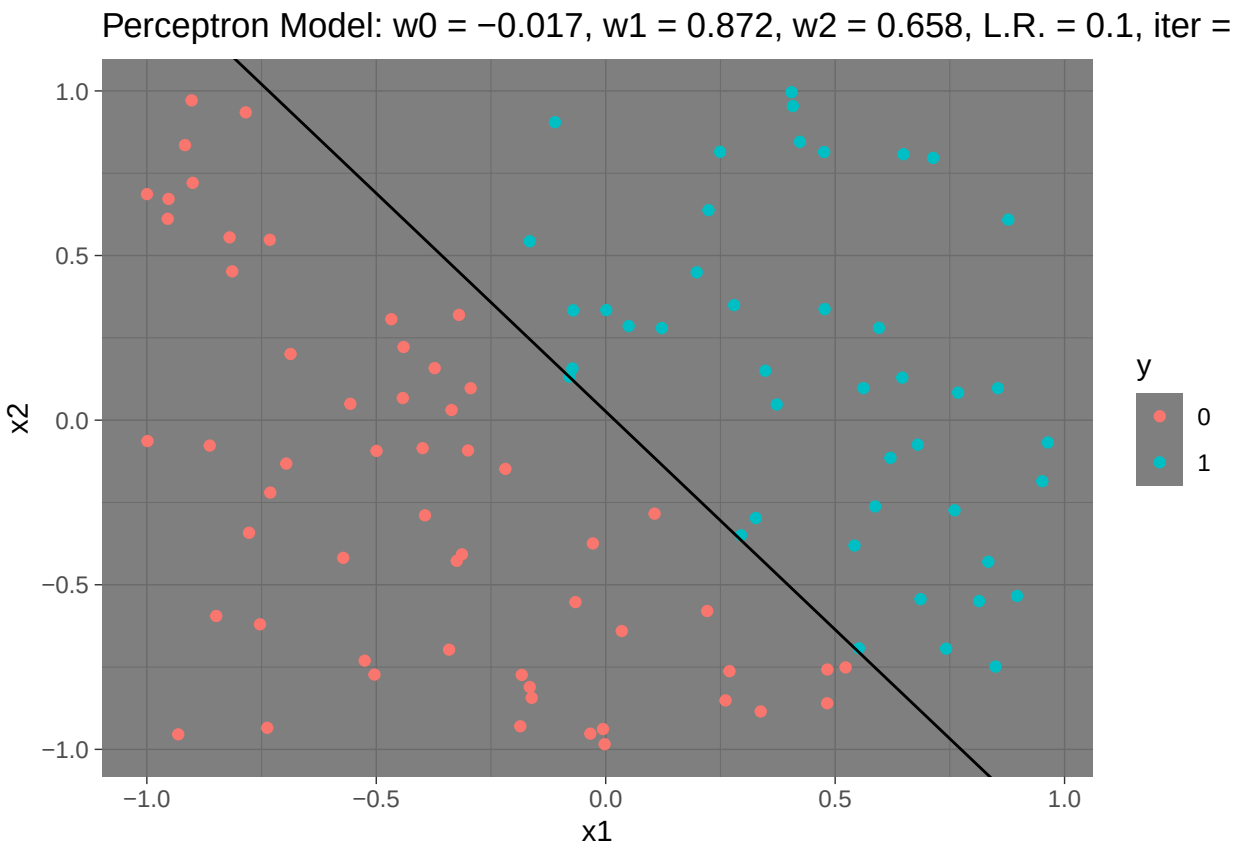
```
perceptron.plot <- function(x, y, seed=1234, rho=.1, iter=1){

  # Add the extra column of 1's to allow a linear separation not through origin.
  x <- cbind(x, rep(1, nrow(x)))

  # Random starting value for the direction vector w.
  weight <- runif(ncol(x))

  # weight.mod is the flag for convergence.
  # start as TRUE so we get at least once through the loop.
  weight.mod <- TRUE

  # iter counts the number of times through the dataset before algo converges.
  iter <- 0

  iter.vec <- as.numeric(iter)
  #initialize while loop
  while(weight.mod){
    weight.mod <- FALSE
    for(i in 1:nrow(x)){
      y.hat <- as.numeric(weight %*% x[i, ] >= 0)
      weight.mod <- weight.mod || (y[i] != y.hat)
      weight <- weight + rho * (y[i] - y.hat) * x[i, ]
    }
      iter <- iter + 1
  }

  g1 <-
  ggplot(data.frame(x1=x[, 1],
                    x2=x[, 2],
                    y=as.factor(y))) +
  geom_point(aes(x=x1,
                 y=x2,
                 color=y)) +
  geom_abline(slope=-weight[1]/weight[2],
              intercept=-weight[3]/weight[2]) +
  ggtitle(paste0("Perceptron Model: w0 = ", round(weight[3], 3),
                 ", w1 = ", round(weight[1], 3),
                 ", w2 = ", round(weight[2], 3),
                 ", L.R. = ", rho,
                 ", iter = ", iter, ".")) +
  theme_dark()

  return(g1)
}
```

Now we can begin plotting

```
#plotting small dataframe
x.small <- as.matrix(small.df[, 1:2])
y.small <- small.df$y

#bring out out the weight and iterations
perceptron.plot(x.small, y.small)
```



```r
#plotting larger dataframe
x.large <- as.matrix(large.df[, 1:2])
y.large <- large.df$y
perceptron.plot(x.large, y.large)
```



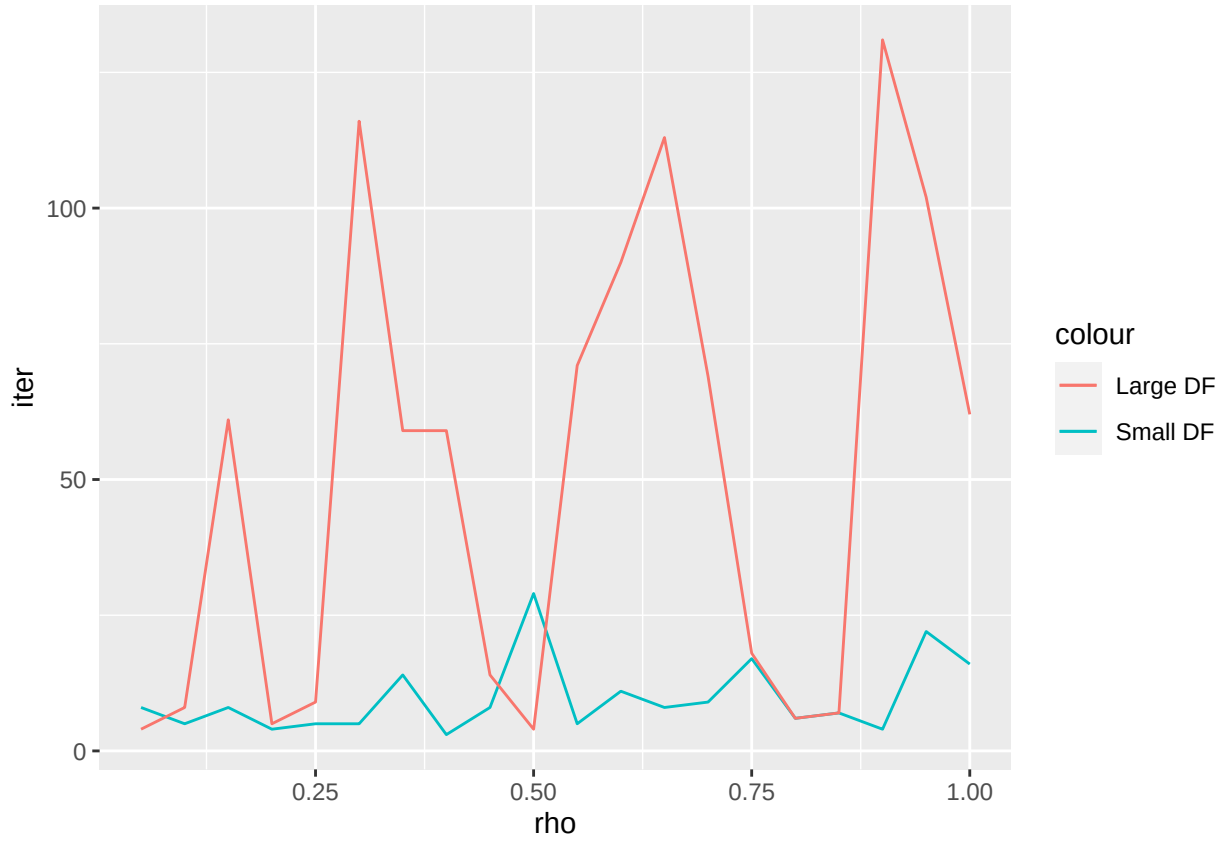
## 2

Write code that begins with a set direction vector weight. Create a loop to try several learning rates between "close to 0" and 1. Store the number of iterations needed at each learning rate. Plot a graph of the number of iterations vs. learning rate with learning rate on the "x" axis.

```
#this is the same basic function as perceptron.plot
#the only difference is instead of returning a ggplot object we return a df

perceptron <- function(x, y, seed=1234, rho=.05, ...){

  # Add the extra column of 1's to allow a linear separation not through origin.
  x <- cbind(x, rep(1, nrow(x)))

  # Random starting value for the direction vector w.
  weight <- runif(ncol(x))

  # weight.mod is the flag for convergence.
  # start as TRUE so we get at least once through the loop.
  weight.mod <- TRUE

  # iter counts the number of times through the dataset before algo converges.
  iter <- 1

  #initialize while loop
  while(weight.mod){
    weight.mod <- FALSE
    for(i in 1:nrow(x)){
      y.hat <- as.numeric(weight %*% x[i, ] >= 0)
      weight.mod <- weight.mod || (y[i] != y.hat)
      weight <- weight + rho * (y[i] - y.hat) * x[i, ]
    }
      iter <- iter + 1
  }
    return(data.frame(iter=iter, rho=rho))
}
```

Analyzing the learning rate compared to iterations Similar to analyzing RMSE vs Span in a prior module's assignment

```
rho.vec <- seq(0.05, 1.0, by=0.05)
iter.vec.small <- map_dfr(rho.vec, ~perceptron(x.small, y.small, rho=.x))
iter.vec.large <- map_dfr(rho.vec, ~perceptron(x.large, y.large, rho=.x))

#plot(x=iter.vec.small$rho, y=iter.vec.small$iter)
#plot(x=iter.vec.large$rho, y=iter.vec.large$iter)


p <- ggplot() +
  geom_line(data=iter.vec.small, aes(x=rho, y=iter, colour="Small DF")) +
  geom_line(data=iter.vec.large, aes(x=rho, y=iter, colour="Large DF"))
p
```




The number of iterations for the small dataframe seems to be constant at less than 15. The larger dataframe fluctuates wildly. I'm not sure why the "Large DF" exhibits these characteristics with relatively equal minimums at rho values of .05, .15, .25, .30, .35, .40, .45, .75. Interestingly, the lowest learning rate of .05 seemed to work well for both dataframes.