HW2

Homework 2 Problem Statement In this homework you will implement several versions of the perceptron algorithm and implement the fisher linear discriminant. In addition to this markdown file there is a another file called *perceptrons_functions_hw.R*, which contains helper functions and code stubs that need to be completed.

A good reference on perceptron algorithms is The Perceptron Chapter by Hal Daume

Starting with this homework you will develop code in a .R file. Once you have debugged your code you will then use those functions to complete this markdown file.

Step 1 Load functions Create a chunk to do the following:

- Source the hw2_514.R file.

- Run ls.str to see which functions are loaded in your environment
- print.fun('name') will print the named function. Use this function to display the code for print.fun and perceptron.box.data

```
rm(list = ls())
options(scipen= 999)
setwd("G:\\math\\514")

source('./hw2_514.R')
```

You can see the effect of sourcing by checking the objects in your environment using ls.str()

```
ls.str()
```

```
## avg.perceptron.train : function (x, y, iter = 100)
## compute_vote : function (wgts, cs, bs, x)
## fisher : function (x, y)
## fisher.2d.line : function (w, m)
## freund.voted.perceptron.train : function (x, y, iter = 200)
## perceptron.box.data : function (n, gamma = 0.25, seed = NULL)
## perceptron.train : function (x, y, epoch = 100)
## plot.perceptron.box.data : function (data, title = "perceptron")
## predict.perceptron : function (w, b, x)
## predict.voted : function (wgts, cs, bs, x)
## print.fun : function (x)
## run.experiments : function (data, num.trials = 1, epochs = 100)
```

One of the helper functions provided is called *print.fun*. You use *print.fun* to display your code inside the markdown document. For example:

```
print.fun('print.fun');
```

```
## [1] "print.fun (x) "
## {
##     header = deparse(args(x))[1]
##     b = body(x)
##     print(gsub("function", x, header))
##     print(b)
## }
```

```
print.fun('perceptron.box.data');
```

```
## [1] "perceptron.box.data (n, gamma = 0.25, seed = NULL) "
## {
##     require(zeallot)
##     if (!is.null(seed))
##         set.seed(seed)
##     data = matrix(0, nrow = n, ncol = 3)
##     discriminant = function(x, y) {
##         (1 + x - y)/sqrt(2)
##     }
##     m = 0
##     while (m < n) {
##         x = runif(1, 0, 1)
##         y = runif(1, 0, 3)
##         d = discriminant(x, y)
##         d1 = d >= gamma
##         d2 = d <= -gamma
##         if (d1 & !d2) {
##             m = m + 1
##             data[m, ] %<-% c(x, y, +1)
##         }
##         else if (d2 & !d1) {
##             m = m + 1
##             data[m, ] %<-% c(x, y, -1)
##         }
##         else if (d1 & d2) {
##             m = m + 1
##             data[m, ] %<-% c(x, y, sample(c(-1, 1), 1))
##         }
##     }
##     data
## }
```

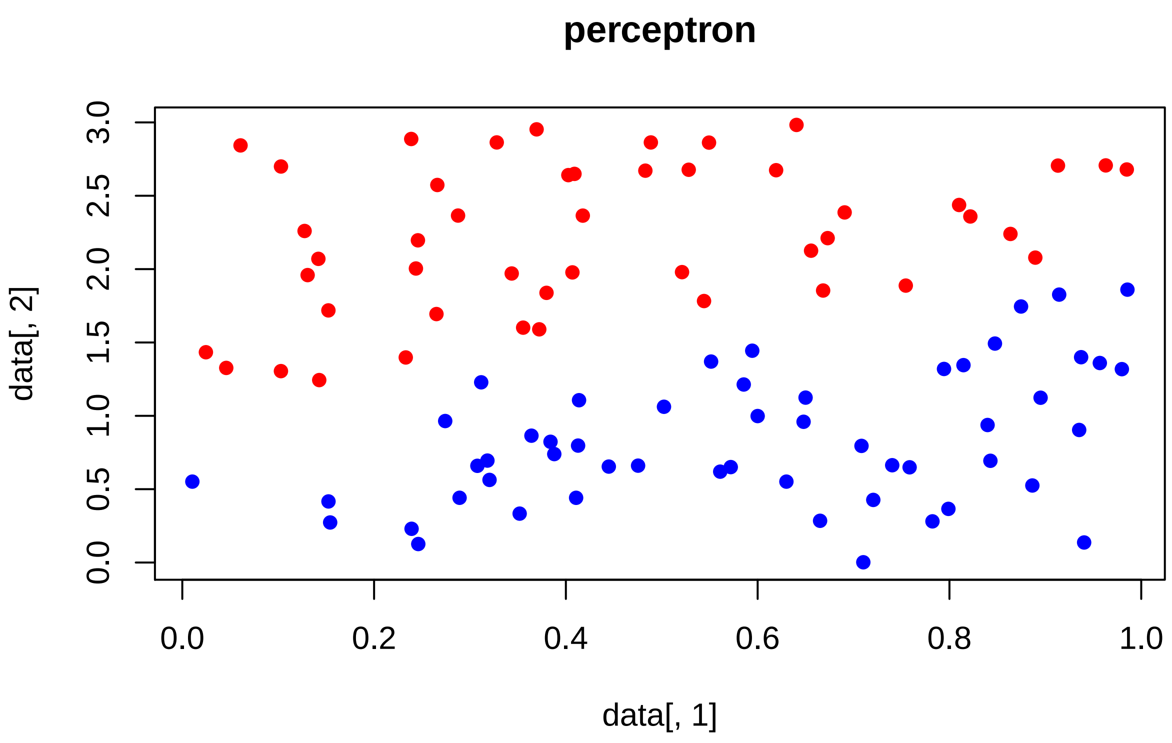
Step 2 Create data

Use the perceptron.box.data function to create a dataset of size 100 with gamma=0.05. Then plot the data. The gap between the datasets is 2*gamma.

```
set.seed(123)
gamma <- .05
dat  <- perceptron.box.data( 100, gamma, 123 )
```

```
## Loading required package: zeallot
```

```
plot.perceptron.box.data(dat );
```



Step 3 Write functions to train and test the vanilla perceptron

Implement a function called train.perceptron that takes as input x data and an outcome y and number of epochs of training to be performed. This function will implement the perceptron training rule to adjust weights when mistakes are encountered and stop updating weights when no mistakes occur.

Test your function on data generated from the perceptron.box.data function and plot perceptron decision boundary. Note: you will need to save source your code to in the perceptron_functions_hw.R and source the code before you can call it in this notebook.

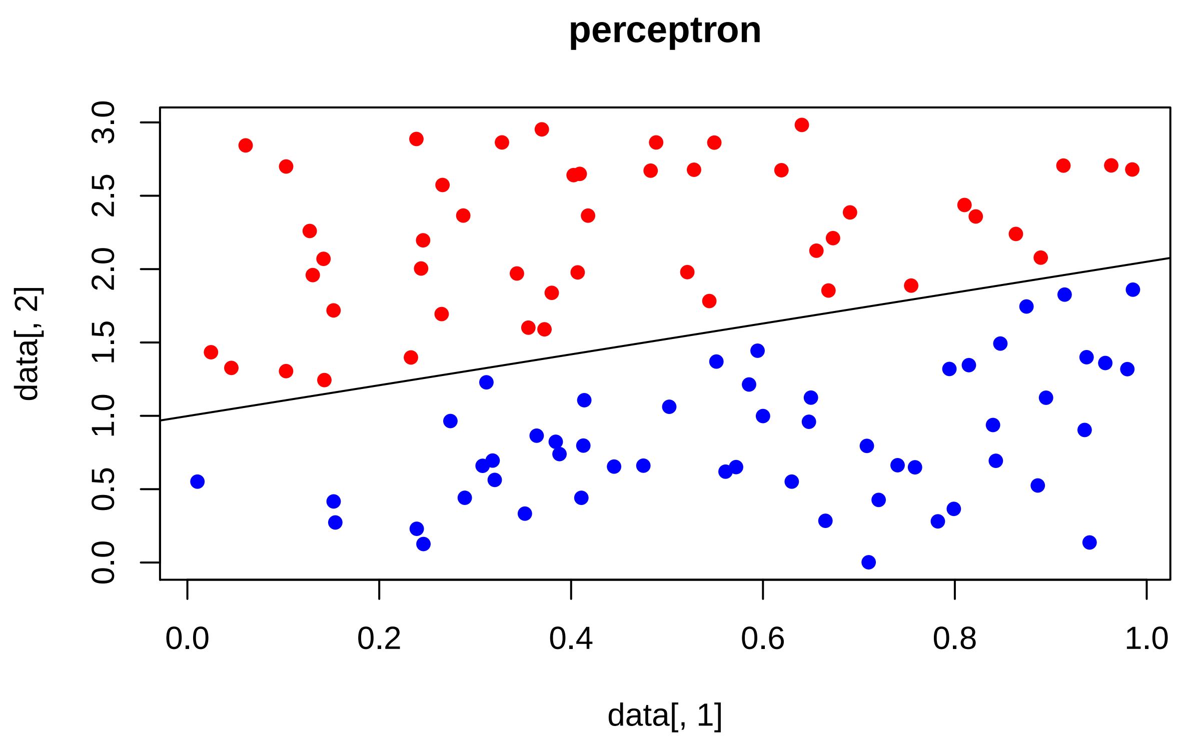
In addition implement a predict.perceptron function that takes input as w (weights returned from perceptron train), b (intercept learned from perceptron.train) and applies the learned weights to predict data x.

Use the gamma of .05 and number of observations of a 100 from step 2 to generate data for train and test.

Comments in the homework fill explain the inputs and outputs the functions should generate.

use the abline function in R to plot the decision boundary

```r
source("perceptron_functions_hw.R")
 y = dat[,ncol(dat )]
 x = dat[,-ncol(dat )]

# Wrote the plot as an argument to the function
# Did not want a global b & w running around
train.perceptron(x,y, Plot = T)
```

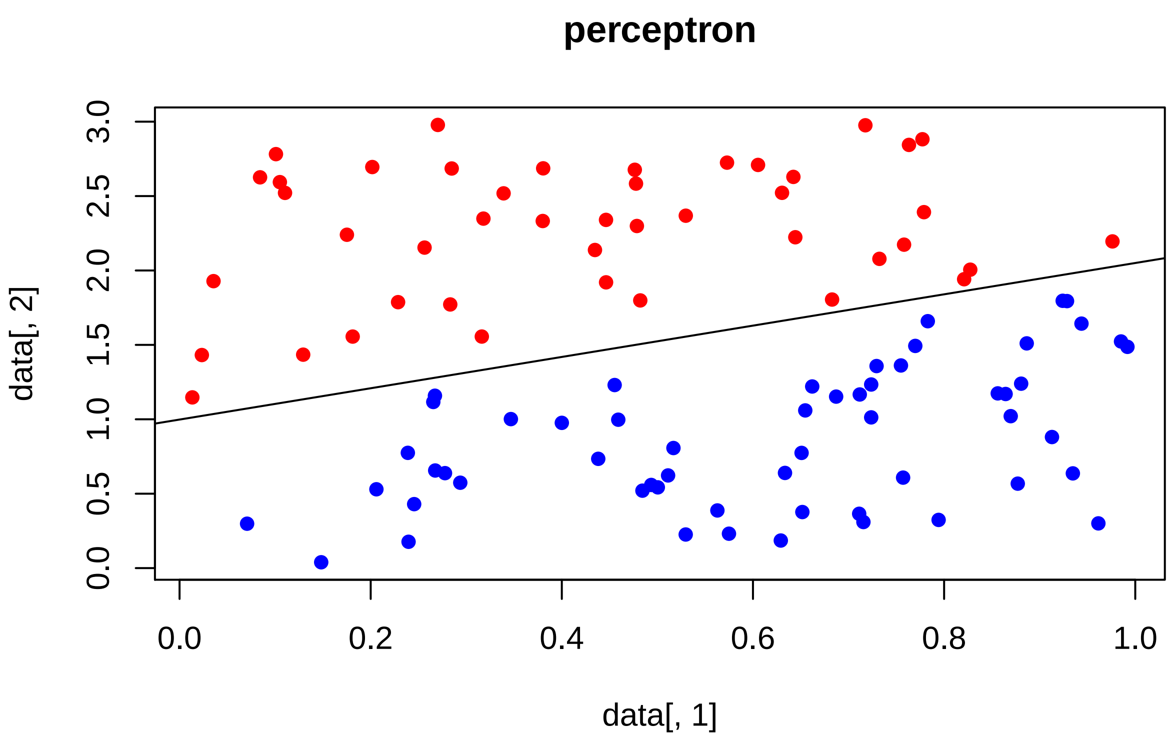


```
## $w
## [1]  3.162171 -3.005580
##
## $b
## [1] 3
```

```
vanilla <- train.perceptron(x,y)

test.dat  <- perceptron.box.data( 100, .05 , seed = 1)
predict.perceptron(vanilla$w,vanilla$b, test.dat[,-3] )
```

```
## $w
##   [1]  1 -1 -1  1  1  1  1 -1 -1  1  1  1 -1  1 -1  1 -1  1  1 -1  1 -1 -1
##  [24] -1 -1  1  1 -1  1  1  1  1 -1 -1 -1  1 -1  1 -1 -1  1  1  1  1  1 -1
##  [47] -1  1  1 -1  1 -1  1 -1  1  1 -1  1  1  1  1 -1  1 -1  1  1 -1  1  1
##  [70] -1 -1  1  1  1 -1 -1  1  1 -1  1 -1 -1 -1  1  1  1 -1  1  1 -1 -1 -1
##  [93] -1  1  1 -1  1 -1  1 -1
```

```
plot.perceptron.box.data( test.dat )
abline( -vanilla$b/  vanilla$w[2] ,  - vanilla$w[1] /  vanilla$w[2] )
```

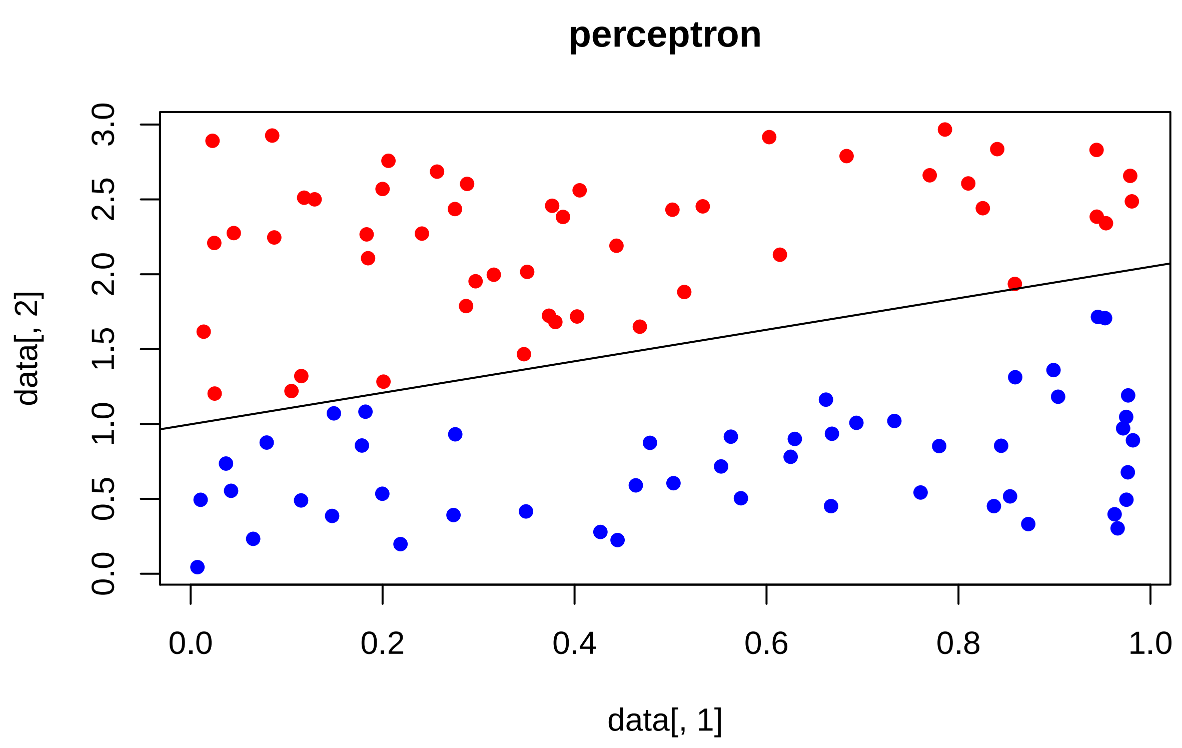


Step 4 Test vanilla perceptron on out of sample data and plot Using a gamma of .05, generate an out of sample data set using perceptron.box.data (100 observations). Use your *predict.perceptron* function to classify the out of sample data check performance. Use a plot to visually cross-check your performance.

```
# at gamma = 0.05 there are no misclassification
test.dat  <- perceptron.box.data( 100, .05  , seed = 2)
plot.test.dat <- as.data.frame( cbind(test.dat ,

predict.perceptron(vanilla$w,vanilla$b, test.dat[,-3] )$w ) )

plot.test.dat$V5 <-  (plot.test.dat$V4 == plot.test.dat$V3 )
plot.perceptron.box.data( test.dat )

predict.perceptron(vanilla$w,vanilla$b, test.dat[,-3] )$w   ==  test.dat[,3]
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE
```

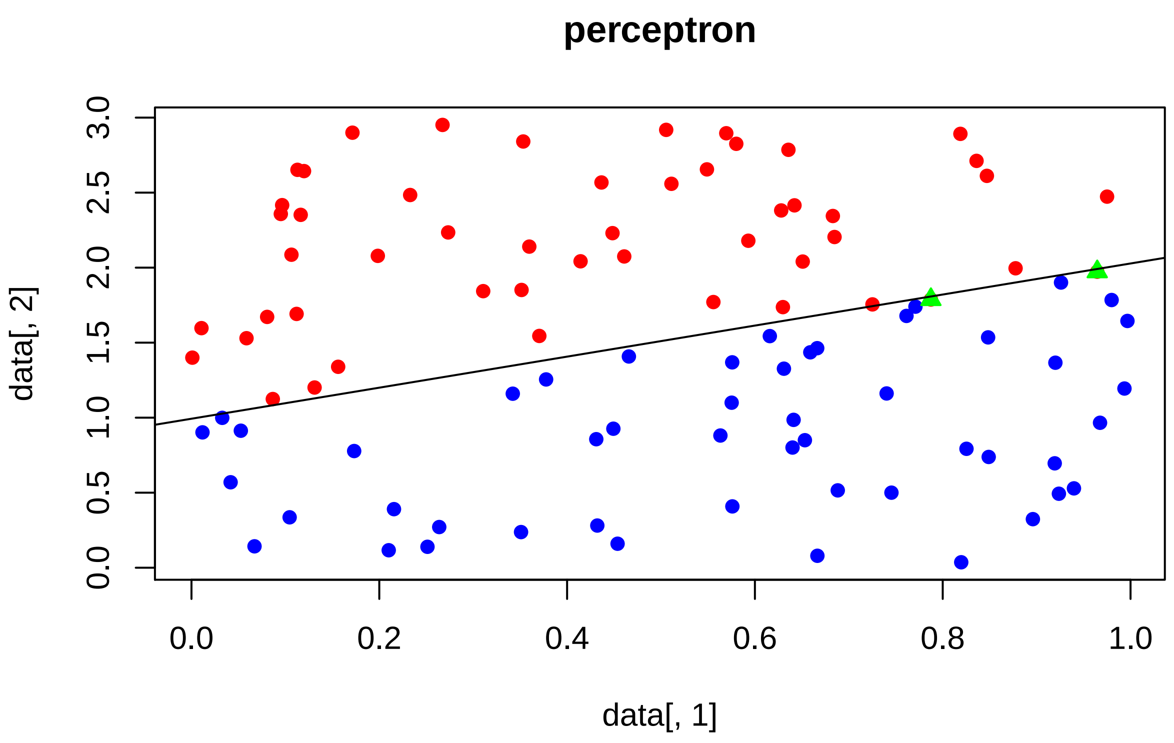
```
points( plot.test.dat[ which( plot.test.dat$V5 == F ), ]$V1,
    plot.test.dat[ which( plot.test.dat$V5 == F ), ]$V2,
    col="green", bg="green" , pch = 24)
abline( -vanilla$b/  vanilla$w[2] ,  - vanilla$w[1] /  vanilla$w[2] )
```



```r
# at gamma = .00005 there is misclassification
dat  <- perceptron.box.data( 100, .00005  , seed = 2)
vanilla <- train.perceptron(dat[,-3],dat[,3])

test.dat  <- perceptron.box.data( 100, .00005  , seed = 111)
plot.test.dat <- as.data.frame( cbind(test.dat ,

predict.perceptron(vanilla$w,vanilla$b, test.dat[,-3] )$w ) )


predict.perceptron(vanilla$w,vanilla$b, test.dat[,-3] )$w   ==  test.dat[,3]
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [12]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [56]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [67]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [78]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [89]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [100]  TRUE
```

```r
plot.test.dat$V5 <-  (plot.test.dat$V4 == plot.test.dat$V3 )
plot.perceptron.box.data( test.dat )
points( plot.test.dat[ which( plot.test.dat$V5 == F ), ]$V1,
    plot.test.dat[ which( plot.test.dat$V5 == F ), ]$V2,
    col="green", bg="green" , pch = 24)
abline( -vanilla$b/  vanilla$w[2] ,  - vanilla$w[1] /  vanilla$w[2] )
```



Step 5 Run experiments to test empirical bound on the number of mistakes perceptron makes

Verify that the number of mistakes made by your perceptron training algorithm fall within the theoretical limits.

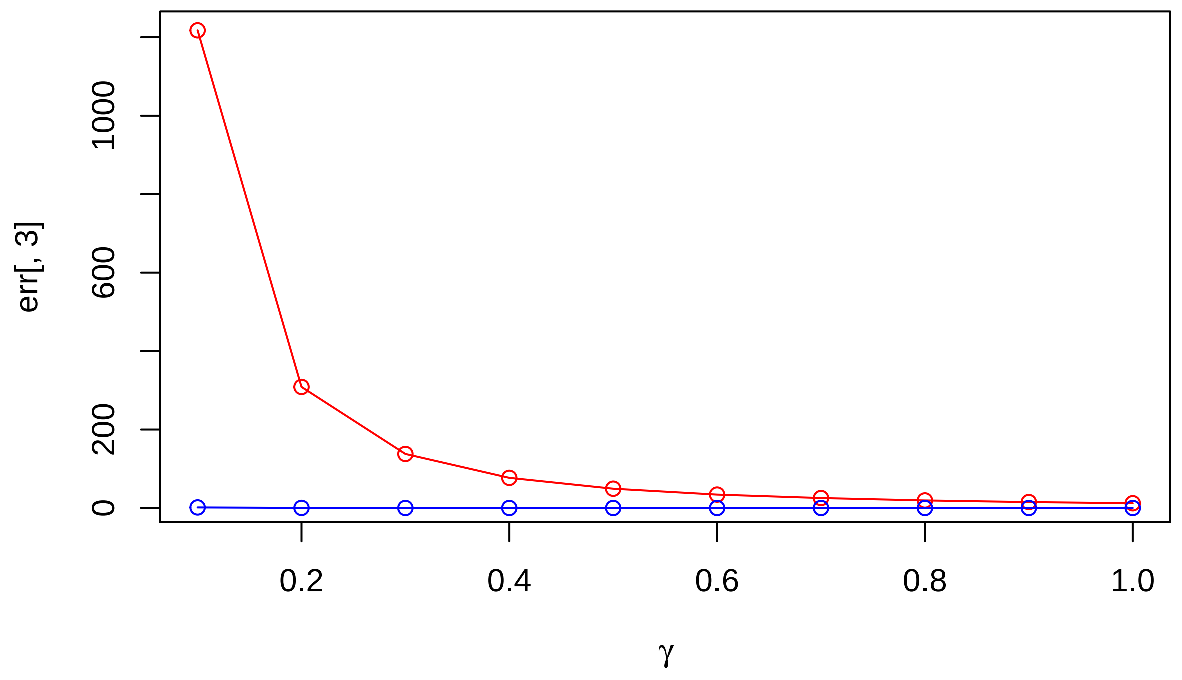
For gamma in *seq(.1,1,length.out = 10)*, train 10 models for each value of gamma and average the number of mistakes. Plot avg. number of mistakes vs theoretical limit of number of mistakes. Use 100 observations for each of the 100 datasets.

The Perceptron Convergence Proof shows that $\frac{R^2}{\gamma^2}$ where R^2 is the maximum euclidean norm of the observations in the data. The bound on mistakes is this $4\frac{(\max\|x\|)^2}{\gamma^2}$

Compute the theoretical limit for each gamma and plot this against the avg. empirical number of mistakes in your experiments.

```
gm <- gmm <- theor <-  theorm <- rep(NA,10)

for( i in 1:length( seq(.1,1,length.out = 10) ) ){

    gamma <- seq(.1,1,length.out = 10)[i]
    dat <- perceptron.box.data( 100, gamma , seed = runif(1 , 1 , 1e8)  )
    trained <- train.perceptron( dat[,-3] , dat[,3] )

    for( j in 1:10){
        PBD  <- perceptron.box.data(100, gamma, seed = runif(1, 1 , 1e8))
        PPBD <- predict.perceptron(  trained$w , trained$b , PBD[,-3] )
        gm[j] <- length( which( ( PBD[,3] == PPBD$w)  == F ) )
        theor[j] <- max( (4 / gamma^2 ) * apply(PBD[,-3] , 1,
                function(W) norm(W,"2")))

    }
    gmm[i] <- mean(gm)
```

```r
    theorm[i] <- mean(theor )
}

err <- data.frame(  seq(.1,1,length.out = 10)  , gmm , theorm)
colnames(err) <- c("gamma" , "errors", "theoretical")

plot( err[,1] , err[,3] , col = "red" ,xlab=expression(gamma) )
lines(err[,1] , err[,3], col = "red"  )
points( err[,1] , err[,2], col = "blue"  )
lines(err[,1] , err[,2], col = "blue"  )
```



Step 6 Implement voted perceptron Implement the voted perceptron algorithm described in https://cseweb.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf . Implement the function called freund.voted.perceptron.train which has x,y, and number of epochs as input and returns a list called history which will contain the weights w, b intercept and cost based on survival time for weight change and also return the number of mistakes that occurred during training.

In addition implement the function predict.voted that will use the output of the voted algorithm and use it to classify a data set x.

Then train and test the voted perceptron using these functions on sample data of 100 observations with gamma of .1 and print the number of correct predictions. For test use the predict.voted a new sample of generated data from perceptron.box.data

Voted perceptron outputs an ensemble of weights and survival times which are used to made a decision. You might find it helpful to use more functions to make your code easy to debug.

```r
source("perceptron_functions_hw.R")

print.fun('freund.voted.perceptron.train');
```

```
## [1] "freund.voted.perceptron.train (x, y, iter = 200) "
## {
##     b <- 0
##     k <- 0
##     v <- x[1, , drop = F] * 0
##     cc <- 0
##     for (J in 1:iter) {
##         for (i in 1:nrow(x)) {
##             yhat <- sign(b[k + 1] + t(v[k + 1, ]) %*% x[i, ])
##             if (yhat == y[i]) {
##                 cc[k + 1] <- cc[k + 1] + 1
##             }
##             else {
##                 v <- rbind(v, v[k + 1, ] + y[i] * x[i, ])
##                 b <- c(b, b[k + 1] + y[i])
##                 cc <- c(cc, 1)
##                 k <- k + 1
##             }
##         }
##     }
##     return(list(b = b[-1], w = v[-1, ], cc = cc[-1], mistakes = k))
## }
```

```r
train.dat = perceptron.box.data(100,.1, seed = 123)

vote.train <- freund.voted.perceptron.train( train.dat[,-3] , train.dat[,3] )

test.dat <- perceptron.box.data(100,.1, seed = 321)

predict.voted( vote.train$w ,  vote.train$cc , vote.train$b , test.dat[,-3] ) == test.dat[,3]
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE
```

Step 7 Average Perceptron Implement the avg.perceptron.train function in the hw2_514.R file and test against same data out of sample and plot results. Use the same *predict.perceptron* function from earlier to test perceptron generated by avg.perceptron.train.

using a gamma of .05 generate data to train and test the avg and vanilla perceptron on out of sample data. Plot the decision boundary of the 2 models against out of sample data using the plot.perceptron.box.data function. Plot vanilla perceptron using a black line and avg perceptron using a red line and use a legend in your plot.

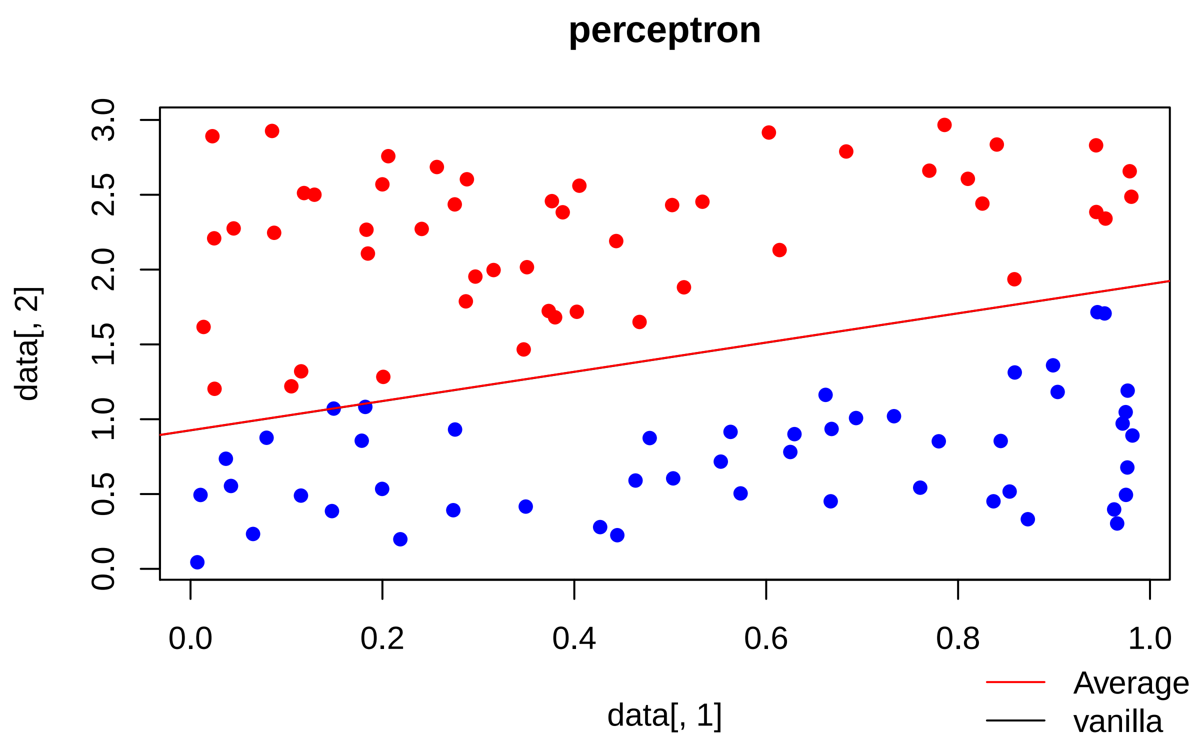
```r
print.fun('avg.perceptron.train');
```

```
## [1] "avg.perceptron.train (x, y, iter = 100) "
## {
##     w <- u <- x[1, , drop = F] * 0
##     b <- B <- 0
##     cc <- 1
##     for (K in 1:iter) {
##         for (i in 1:nrow(x)) {
##             if ((w %*% t(x[i, , drop = F]) + b) * y[i] <= 0) {
##                 w <- w + y[i] * x[i, , drop = F]
##                 b <- b + y[i]
##                 u <- u + y[i] * cc * x[i, , drop = F]
##                 B <- B + y[i] * cc
##             }
##             cc <- cc + 1
##         }
##     }
##     return(list(b = b - (1/cc) * B, w = as.vector(w - (1/cc) *
##         u)))
## }
```

```
 gamma=.05
  dat=perceptron.box.data(100,gamma, seed = 1);
  y = dat[,ncol(dat)]
  x = dat[,-ncol(dat)]

avg <- avg.perceptron.train(x , y)

vanilla <- train.perceptron(x,y)

test.dat <- perceptron.box.data(100,gamma, seed = 2)

predict.perceptron( avg$w , avg$b , test.dat[ ,-3] )$w == test.dat[,3]
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE
```

```
plot.perceptron.box.data( test.dat )
abline( -vanilla$b/  vanilla$w[2] ,  - vanilla$w[1] /  vanilla$w[2] , col = "black" )
abline( -avg$b/  avg$w[2] ,  - avg$w[1] /  avg$w[2] , col = "red" )
legend(.8,-.5,legend=c("Average", "vanilla"), bty = "n",    col=c("red", "black"), lty=1:1, xpd=T)
```



Step 8 Implement fisher discriminant function Bishop's section on Fisher discriminant might be useful to you: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf (page 186)

In order to make this document a self-contained piece of reproducible research enter the equations describing the calculations in the *fisher* function.

Enter latex here using codecogs and put it in double $ for it to be rendered in markdown.

$$\boldsymbol{m}_1 = \frac{1}{N_1}\sum_{n\in C_1}\boldsymbol{x}_n \qquad \boldsymbol{m}_2 = \frac{1}{N_2}\sum_{n\in C_2}\boldsymbol{x}_n$$

$$S_i = \sum_{n\in C_i}(\boldsymbol{x}_n - \boldsymbol{m}_i)(\boldsymbol{x}_n - \boldsymbol{m}_i)^T$$

$$\begin{aligned} S_W &= \sum_{n\in C_1}(\boldsymbol{x}_n - \boldsymbol{m}_1)(\boldsymbol{x}_n - \boldsymbol{m}_1)^T + \sum_{n\in C_2}(\boldsymbol{x}_n - \boldsymbol{m}_2)(\boldsymbol{x}_n - \boldsymbol{m}_2)^T \\ &= S_1 + S_2 \end{aligned}$$

$$\boldsymbol{w} \sim S_w^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2)$$

$$\boldsymbol{m}_j = \frac{1}{n_1+n_2}\Big(n_1\boldsymbol{m}_1 + n_2\boldsymbol{m}_2\Big)$$

Step 9 Compare fisher to vanilla and avg perceptron for non-separable data

-Implement a function called fisher to compute the fisher discriminant using the equations documented above (step 8).The fisher function returns the direction vector w and the global mean for box data x, y.

- Print the fisher function in a code chunk using print.fun.
- Generate non-separable data by calling the perceptron.box.data function with a gamma of -0.1.
- Use the provided function *fisher.2d.line* to compute the slope and intercept given the output of fisher function (w,m)
- Apply vanilla perceptron, avg perceptron and fisher discriminant to the non-separable data and plot the decision boundaries for all three algorithms on out of sample data (fisher decision boundary should be purple, avg perceptron in red and vanilla perceptron black). Add a legend to the plot.

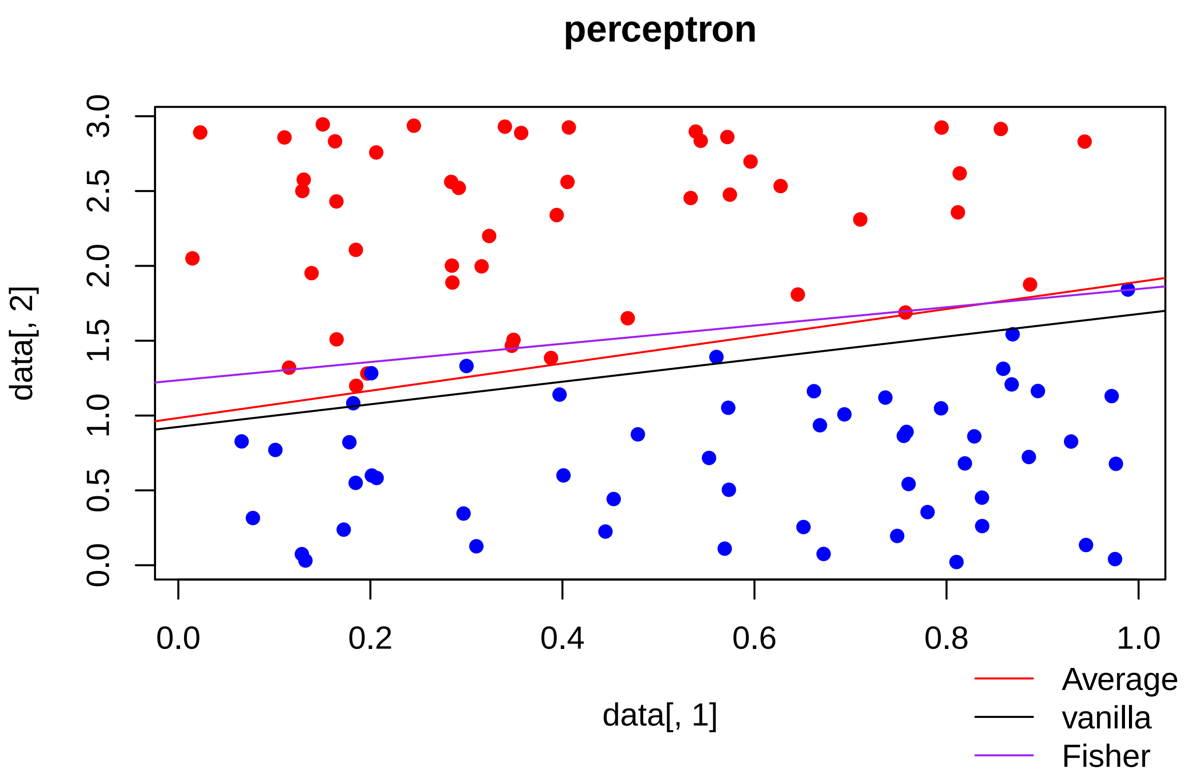
```
  print.fun('fisher')
```

```
## [1] "fisher (x, y) "
## {
##     Dat <- cbind(x, y)
##     Dat1 <- Dat[which(Dat[, 3] == 1), ]
##     X1 <- cbind(Dat1[, -3])
##     n1 <- nrow(Dat1)
##     m1 <- (1/n1) * apply(X1, 2, sum)
##     s1 <- apply(X1, 1, function(W) W - m1)
##     s1 <- matrix(rowSums(as.matrix(apply(s1, 2, function(W) W %*%
##         t(W)))), 2, 2)
##     Dat2 <- Dat[which(Dat[, 3] == -1), ]
##     X2 <- cbind(Dat2[, -3])
##     n2 <- nrow(Dat2)
##     m2 <- (1/n2) * apply(X2, 2, sum)
##     s2 <- apply(X2, 1, function(W) W - m2)
##     s2 <- matrix(rowSums(as.matrix(apply(s2, 2, function(W) W %*%
##         t(W)))), 2, 2)
##     Sw <- s1 + s2
##     w <- solve(Sw) %*% (m2 - m1)
##     m <- 1/(n1 + n2) * (n1 * m1 + n2 * m2)
##     return(list(w = as.vector(w), m = m))
## }
```

```
  dat = perceptron.box.data(100,-.1);
  y = dat[,ncol(dat)]
  x = dat[,-ncol(dat)]

fd <- fisher( x , y )
fisher.2d.line( fd$w,fd$m )
```

```
## $a
## [1] 1.235555
##
## $b
## [1] 0.6100525
```

```
avg <- avg.perceptron.train(x , y)
vanilla <- train.perceptron(x,y)

test.dat <- perceptron.box.data(100, -.1 , seed = 2)
plot.perceptron.box.data( test.dat );
abline( -vanilla$b/  vanilla$w[2] ,  - vanilla$w[1] /  vanilla$w[2] , col = "black" )
abline( -avg$b/  avg$w[2] ,  - avg$w[1] /  avg$w[2] , col = "red" )
abline( fisher.2d.line( fd$w,fd$m )$a , fisher.2d.line( fd$w,fd$m )$b , col = "purple" )
legend(.8,-.5,legend=c("Average", "vanilla", "Fisher"), bty = "n",  col=c("red", "black", "purple"), lty=1:
```



Step 10 Write a margins function to compare the models

Implement a function called margin that will compute the margin for the model on given data x using w,b. Use this function to compare the 3 models on out of sample data.

Similar to experiments in other homeworks generate 10 trials of different data samples with gamma of -.05 and compare the mean margins of the points for the 3 decision boundaries learned earlier.

Which model seems to perform best (largest positive margin)?

```
dat=perceptron.box.data(100,.05 , 1);
y = dat[,ncol(dat)]
x = dat[,-ncol(dat)]

fd <- fisher( x , y )
avg <- avg.perceptron.train(x , y)
vanilla <- train.perceptron(x,y)

test.dat <- perceptron.box.data(100, .05 , seed = 2)
y = test.dat[,3]
x = test.dat[,-3]


margin( x , y , vanilla$w , vanilla$b )
```

```
## [1] 0.0002037238
```

```
margin( x , y , avg$w , avg$b )
```

```
## [1] 0.0007736073
```

```
margin( x , y ,  -fd$w ,   sum(fd$w *fd$m)  )
```

```
## [1] 0.01212444
```

```
Vm <- Am <- Fm <- rep(NA, 10)

for( i in 1:10){
    test.dat <- perceptron.box.data(100, -.05 , seed = i)
    y = test.dat[,3]; x = test.dat[,-3]
    Vm[i] <- margin( x , y , vanilla$w , vanilla$b )
    Am[i] <- margin( x , y , avg$w , avg$b )
    Fm[i] <- margin( x , y , -fd$w ,  sum(fd$w *fd$m)    )
}

mean(Vm); mean(Am); mean(Fm)
```

```
## [1] -0.08473821
```

```
## [1] -0.08427764
```

```
## [1] -0.0594419
```

It looks as if the Fisher one is doing the best.