Lab 3

Tziporah Horowitz

11:59PM Saturday, February 22, 2020

Review from Lab 2

You have set of names divided by gender (M / F) and generation (Boomer / GenX / Millenial):

- M / Boomer "Theodore, Bernard, Gene, Herbert, Ray, Tom, Lee, Alfred, Leroy, Eddie"
- M / GenX "Marc, Jamie, Greg, Darryl, Tim, Dean, Jon, Chris, Troy, Jeff"
- M / Millennial "Zachary, Dylan, Christian, Wesley, Seth, Austin, Gabriel, Evan, Casey, Luis"
- F / Boomer "Gloria, Joan, Dorothy, Shirley, Betty, Dianne, Kay, Marjorie, Lorraine, Mildred"
- F / GenX "Tracy, Dawn, Tina, Tammy, Melinda, Tamara, Tracey, Colleen, Sherri, Heidi"
- F / Millennial "Samantha, Alexis, Brittany, Lauren, Taylor, Bethany, Latoya, Candice, Brittney, Cheyenne"

Create a list-within-a-list that will intelligently store this data.

```
set <- list("M" = list(), "F" = list())
set$M$Boomer = c("Theodore", "Bernard", "Gene", "Herbert", "Ray", "Tom",
                 "Lee", "Alfred", "Leroy", "Eddie")
set$M$GenX = c("Marc", "Jamie", "Greg", "Darryl", "Tim", "Dean", "Jon",
               "Chris", "Troy", "Jeff")
set$M$Millennial = strsplit("Zachary, Dylan, Christian, Wesley, Seth, Austin, Gabriel, Evan, Casey, Luis
set$F$Boomer = strsplit("Gloria, Joan, Dorothy, Shirley, Betty, Dianne, Kay, Marjorie, Lorraine, Mildred
set$F$GenX = strsplit("Tracy, Dawn, Tina, Tammy, Melinda, Tamara, Tracey, Colleen, Sherri, Heidi", split
set$F$Millennial = strsplit("Samantha, Alexis, Brittany, Lauren, Taylor, Bethany, Latoya, Candice, Britt

set
```

```
## $M
## $M$Boomer
##  [1] "Theodore" "Bernard"  "Gene"     "Herbert"  "Ray"      "Tom"
##  [7] "Lee"      "Alfred"   "Leroy"    "Eddie"
##
## $M$GenX
##  [1] "Marc"   "Jamie"  "Greg"   "Darryl" "Tim"    "Dean"   "Jon"
##  [8] "Chris"  "Troy"   "Jeff"
##
## $M$Millennial
##  [1] "Zachary"   "Dylan"     "Christian" "Wesley"    "Seth"
##  [6] "Austin"    "Gabriel"   "Evan"      "Casey"     "Luis"
##
##
## $F
## $F$Boomer
##  [1] "Gloria"   "Joan"     "Dorothy"  "Shirley"  "Betty"    "Dianne"
##  [7] "Kay"      "Marjorie" "Lorraine" "Mildred"
##
## $F$GenX
```

```
##  [1] "Tracy"   "Dawn"    "Tina"    "Tammy"   "Melinda" "Tamara"  "Tracey"
##  [8] "Colleen" "Sherri"  "Heidi"
##
## $F$Millennial
##  [1] "Samantha" "Alexis"   "Brittany" "Lauren"   "Taylor"   "Bethany"
##  [7] "Latoya"   "Candice"  "Brittney" "Cheyenne"
```

Imagine you are running an experiment with many manipulations. You have 14 levels in the variable "treatment" with levels a, b, c, etc. For each of those manipulations you have 3 submanipulations in a variable named "variation" with levels A, B, C. Then you have "gender" with levels M / F. Then you have "generation" with levels Boomer, GenX, Millenial. Then you will have 6 runs per each of these groups. In each set of 6 you will need to select a name without duplication from the appropriate set of names (from the last question). Create a data frame with columns treatment, variation, gender, generation, name and y that will store all the unique unit information in this experiment. Leave y empty because it will be measured as the experiment is executed.

```
n <- 14*3*2*3*10
X <- data.frame(treatment = rep(NA, n),
                variation = rep(NA, n),
                gender = rep(NA, n),
                generation = rep(NA, n),
                name = rep(NA, n),
                y = rep(NA, n))
X$treatment <- rep(letters[1:14], each = n/14)
X$variation <- rep(rep(LETTERS[1:3], each = n/(14*3)), 14)
X$gender <- rep(rep(c("M", "F"), each = n/(14*3*2)), 14*3)
X$generation <- rep(rep(c("Boomer", "GenX", "Millennial"), each = n/(14*3*2*3)), 14*3*2)
X$name <- rep(unlist(set), 14*3)
tail(X, 50)
```

```
##      treatment variation gender generation      name  y
## 2471         n         C      M       GenX      Marc NA
## 2472         n         C      M       GenX     Jamie NA
## 2473         n         C      M       GenX      Greg NA
## 2474         n         C      M       GenX    Darryl NA
## 2475         n         C      M       GenX       Tim NA
## 2476         n         C      M       GenX      Dean NA
## 2477         n         C      M       GenX       Jon NA
## 2478         n         C      M       GenX     Chris NA
## 2479         n         C      M       GenX      Troy NA
## 2480         n         C      M       GenX      Jeff NA
## 2481         n         C      M Millennial   Zachary NA
## 2482         n         C      M Millennial     Dylan NA
## 2483         n         C      M Millennial Christian NA
## 2484         n         C      M Millennial    Wesley NA
## 2485         n         C      M Millennial      Seth NA
## 2486         n         C      M Millennial    Austin NA
## 2487         n         C      M Millennial   Gabriel NA
## 2488         n         C      M Millennial      Evan NA
## 2489         n         C      M Millennial     Casey NA
## 2490         n         C      M Millennial      Luis NA
## 2491         n         C      F     Boomer    Gloria NA
## 2492         n         C      F     Boomer      Joan NA
```

```
## 2493         n         C      F     Boomer   Dorothy NA
## 2494         n         C      F     Boomer   Shirley NA
## 2495         n         C      F     Boomer     Betty NA
## 2496         n         C      F     Boomer    Dianne NA
## 2497         n         C      F     Boomer       Kay NA
## 2498         n         C      F     Boomer  Marjorie NA
## 2499         n         C      F     Boomer  Lorraine NA
## 2500         n         C      F     Boomer   Mildred NA
## 2501         n         C      F       GenX     Tracy NA
## 2502         n         C      F       GenX      Dawn NA
## 2503         n         C      F       GenX      Tina NA
## 2504         n         C      F       GenX     Tammy NA
## 2505         n         C      F       GenX   Melinda NA
## 2506         n         C      F       GenX    Tamara NA
## 2507         n         C      F       GenX    Tracey NA
## 2508         n         C      F       GenX   Colleen NA
## 2509         n         C      F       GenX    Sherri NA
## 2510         n         C      F       GenX     Heidi NA
## 2511         n         C      F Millennial  Samantha NA
## 2512         n         C      F Millennial    Alexis NA
## 2513         n         C      F Millennial  Brittany NA
## 2514         n         C      F Millennial    Lauren NA
## 2515         n         C      F Millennial    Taylor NA
## 2516         n         C      F Millennial   Bethany NA
## 2517         n         C      F Millennial    Latoya NA
## 2518         n         C      F Millennial   Candice NA
## 2519         n         C      F Millennial  Brittney NA
## 2520         n         C      F Millennial  Cheyenne NA
```

Packages

Install the package `pacman` using regular base R.

```
#install.packages("pacman")
```

First, install the package `testthat` (a widely accepted testing suite for R) from https://github.com/r-lib/testthat using `pacman`. If you are using Windows, this will be a long install, but you have to go through it for some of the stuff we are doing in class. LINUX (or MAC) is preferred for coding. If you can't get it to work, install this package from CRAN (still using `pacman`), but this is not recommended long term.

```
pacman::p_load(testthat)
```

- Create vector `v` consisting of all numbers from -100 to 100 and test using the second line of code:

```
v <- -100:100
expect_equal(v, -100 : 100)
```

If there are any errors, the `expect_equal` function will tell you about them. If there are no errors, then it will be silent.

Test the `my_reverse` function from lab2 using the following code:

```
my_reverse <- function(v){
  v[length(v):1]
}
expect_equal(my_reverse(v), rev(v))
expect_equal(my_reverse(c("A", "B", "C")), rev(LETTERS[1:3]))
```

Basic Binary Classification Modeling

- Load the famous `iris` data frame into the namespace. Provide a summary of the columns and write a few descriptive sentences about the distributions using the code below and in English.

```
data(iris)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
unique(iris$Species)
```

```
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
```

```
summary(subset(iris, Species == "setosa"))
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.300   Min.   :1.000   Min.   :0.100
##  1st Qu.:4.800   1st Qu.:3.200   1st Qu.:1.400   1st Qu.:0.200
##  Median :5.000   Median :3.400   Median :1.500   Median :0.200
##  Mean   :5.006   Mean   :3.428   Mean   :1.462   Mean   :0.246
##  3rd Qu.:5.200   3rd Qu.:3.675   3rd Qu.:1.575   3rd Qu.:0.300
##  Max.   :5.800   Max.   :4.400   Max.   :1.900   Max.   :0.600
##        Species
##  setosa    :50
##  versicolor: 0
##  virginica : 0
##
##
##
```

```
summary(subset(iris, Species == "versicolor"))
```

```
##   Sepal.Length    Sepal.Width     Petal.Length   Petal.Width
##  Min.   :4.900   Min.   :2.000   Min.   :3.00   Min.   :1.000
##  1st Qu.:5.600   1st Qu.:2.525   1st Qu.:4.00   1st Qu.:1.200
##  Median :5.900   Median :2.800   Median :4.35   Median :1.300
##  Mean   :5.936   Mean   :2.770   Mean   :4.26   Mean   :1.326
##  3rd Qu.:6.300   3rd Qu.:3.000   3rd Qu.:4.60   3rd Qu.:1.500
##  Max.   :7.000   Max.   :3.400   Max.   :5.10   Max.   :1.800
##        Species
```

```
##   setosa    : 0
##   versicolor:50
##   virginica : 0
##
##
##
```

```r
summary(subset(iris, Species == "virginica"))
```

```
##    Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##   Min.   :4.900   Min.   :2.200   Min.   :4.500   Min.   :1.400
##   1st Qu.:6.225   1st Qu.:2.800   1st Qu.:5.100   1st Qu.:1.800
##   Median :6.500   Median :3.000   Median :5.550   Median :2.000
##   Mean   :6.588   Mean   :2.974   Mean   :5.552   Mean   :2.026
##   3rd Qu.:6.900   3rd Qu.:3.175   3rd Qu.:5.875   3rd Qu.:2.300
##   Max.   :7.900   Max.   :3.800   Max.   :6.900   Max.   :2.500
##         Species
##   setosa    : 0
##   versicolor: 0
##   virginica :50
##
##
##
```

The `iris` dataset is used to desribe and identify three species of flowers: setosa, versicolor, and virginica. To do so, it uses measurements of sepal length, sepal width, petal length, and petal width for 50 of each species. The data suggests that out of the three species, virginicas tend to be largest in sepal width, petal length, and petal width, while setosas tend to be smallest in the three categories. However, sepal width does not seem to increase with the other measurements.

The outcome metric is `Species`. This is what we will be trying to predict. However, we only care about binary classification between "setosa" and "versicolor" for the purposes of this exercise. Thus the first order of business is to drop one class. Let's drop the data for the level "virginica" from the data frame.

```r
iris <- iris[iris$Species != "virginica", ]
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50          0
```

Now create a vector `y` that is length the number of remaining rows in the data frame whose entries are 0 if "setosa" and 1 if "versicolor".

```r
y <- ifelse(iris$Species == "setosa", 0, 1)
y
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Write a `Mode` function

```
Mode <- function(x){
  u <- unique(x)
  vec <- rep(0, length(u))
  for (i in 1:length(u)){
    for (j in 1:length(x)){
      if (x[j] == u[i]){
        vec[i] <- vec[i] + 1
      }
    }
  }
  m <- max(vec)
  tvec <- c()
  for (i in 1:length(vec)){
    if (vec[i] == m){
      tvec <- c(tvec, u[i])
    }
  }
  if (length(tvec) > 1) warning("warning: x is multimodal \n \n")
  tvec
}

k = c(1, 2, 2, 3, 4, 1, 1, 4, 4, 2)
Mode(k)
```

```
## Warning in Mode(k): warning: x is multimodal
##
```

```
## [1] 1 2 4
```

- Fit a threshold model to `y` using the feature `Sepal.Length`. Write your own code to do this. What is the estimated value of the threshold parameter? What is the total number of errors this model makes?

```
n <- nrow(iris)
numErrors <- array(NA, n)
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
##  [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5
##  [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0
##  [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8
##  [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4
##  [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7
```

```
for (i in 1:n){
  y_hat <- as.numeric(iris$Sepal.Length > iris$Sepal.Length[i])
  numErrors[i] <- sum(y_hat != y)
}
numErrors
```

```
##   [1] 18 31 39 41 25 11 41 25 46 31 11 34 34 49 24 22 11 18 22 18 11 18 41
##  [24] 18 34 25 25 16 16 39 34 11 16 14 31 25 14 31 46 18 25 45 46 25 18 34
```

```
##  [47] 18 41 15 25 50 41 49 14 42 22 39 31 44 16 25 26 30 34 19 47 19 24 36
##  [70] 19 26 34 39 34 41 44 48 47 30 22 14 14 24 30 11 30 47 39 19 14 14 34
##  [93] 24 25 19 22 22 36 18 22
```

```
threshold <- iris$Sepal.Length[which.min(numErrors)]
g <- function(x){
  as.numeric(x > threshold)
}

sum(g(iris$Sepal.Length) != y)
```

```
## [1] 11
```

Does this make sense given the following summaries:

```
summary(iris[iris$Species == "setosa", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   4.800   5.000   5.006   5.200   5.800
```

```
summary(iris[iris$Species == "versicolor", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.900   5.600   5.900   5.936   6.300   7.000
```

Write your answer here in English.

75% of versicolors have Sepal.Length greater than 5.4 while 75% of setosas have Sepal.Length less than 5.4.

Create the function `g` explicitly that can predict `y` from `x` being a new `Sepal.Length`.

```
g <- function(x){
  ifelse(x > threshold, 1, 0)
}
g(iris$Sepal.Length)
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
##  [36] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1
```

- What is the total number of errors this model makes in the dataset $\mathbb{D}$?

```
sum(g(iris$Sepal.Length) != y)
```

```
## [1] 11
```

Perceptron

You will code the "perceptron learning algorithm". Take a look at the comments above the function. This is standard "Roxygen" format for documentation. Hopefully, we will get to packages at some point and we will go over this again. It is your job also to fill in this documentation.

```
#' Perceptron Learning Algorithm
#'
#' Assuming the data is linearly seperable, the perceptron fits the best line for the data.
#'
#' @param Xinput      The training data features as a n x p matrix
#' @param y_binary    The training data responses as a n x 1 vector of 1's and 0's
#' @param MAX_ITER    The maximum number of iterations the algorithm performs, default = 1000
#' @param w           (p + 1) x 1 vector of weights, initialized as NULL
#'
#' @return            The computed final parameter (weight) as a vector of length p + 1
#' @export            [In a package, this documentation parameter signifies this function becomes a pub
#'
#' @author            [Tziporah Horowitz]
perceptron_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 1000, w = NULL){

  n <- nrow(Xinput)
  Xinput <- cbind(rep(1, n), Xinput)
  p <- ncol(Xinput)

  if (is.null(w)) {
    w <- rep(0, p)
  }

  for (i in 1:MAX_ITER) {
    for (j in 1:n) {
      X <- Xinput[j, ]
      e <- y_binary[j] - ifelse(sum(w %*% X) > 0, 1, 0)
      for (k in 1:p) {
        w[k] <- w[k] + e * X[k]
      }
    }
  }
  return(w)
}
```

To understand what the algorithm is doing - linear "discrimination" between two response categories, we can draw a picture. First let's make up some very simple training data $\mathbb{D}$.

```
Xy_simple = data.frame(
 response = factor(c(0, 0, 0, 1, 1, 1)), #nominal
 first_feature = c(1, 1, 2, 3, 3, 4),    #continuous
 second_feature = c(1, 2, 1, 3, 4, 3)    #continuous
)
```

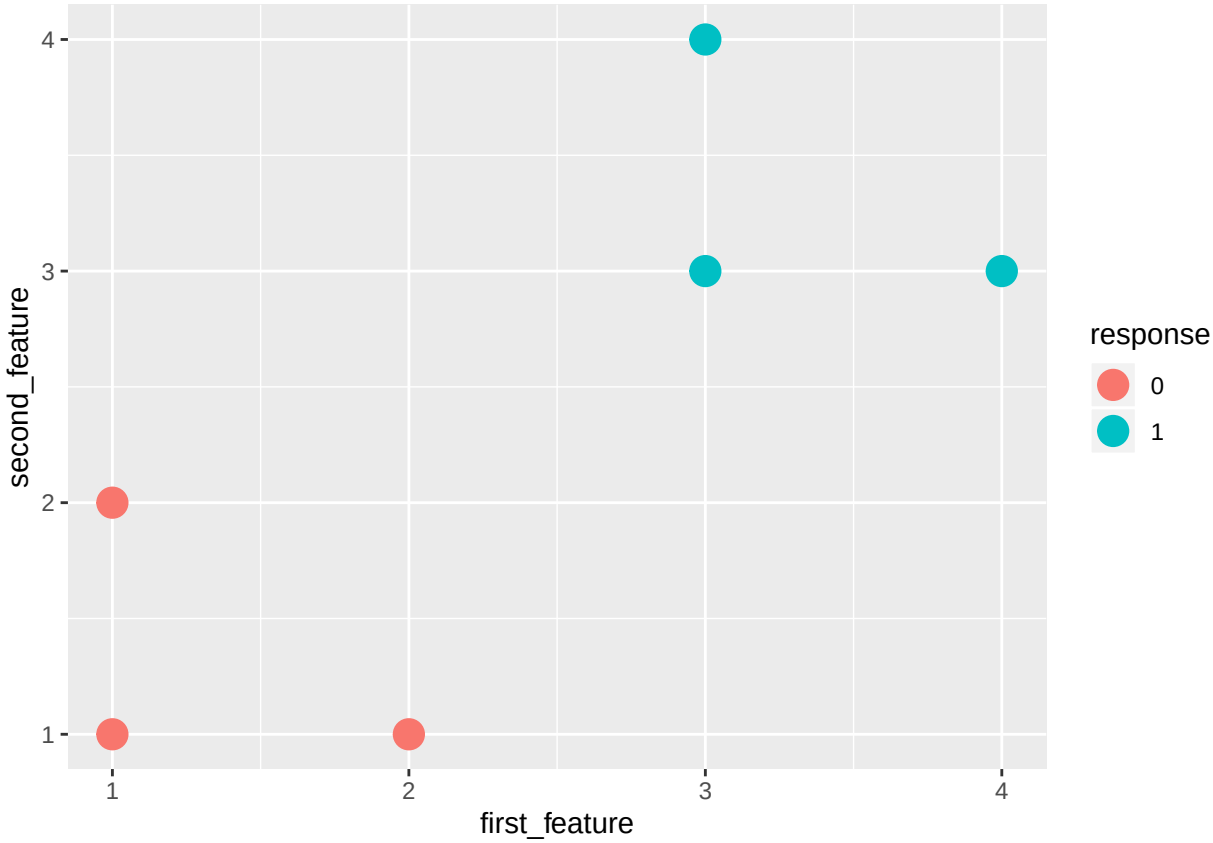
We haven't spoken about visualization yet, but it is important we do some of it now. Thus, I will write this code for you and you will just run it. First we load the visualization library we're going to use:

```
pacman::p_load(ggplot2)
```

We are going to just get some plots and not talk about the code to generate them as we will have a whole unit on visualization using `ggplot2` in the future.

Let's first plot y by the two features so the coordinate plane will be the two features and we use different colors to represent the third dimension, y.

```
simple_viz_obj = ggplot(Xy_simple, aes(x = first_feature, y = second_feature, color = response)) +
  geom_point(size = 5)
simple_viz_obj
```



The graph shows binary responses based on the first feature and the second feature.

Now, let us run the algorithm and see what happens:

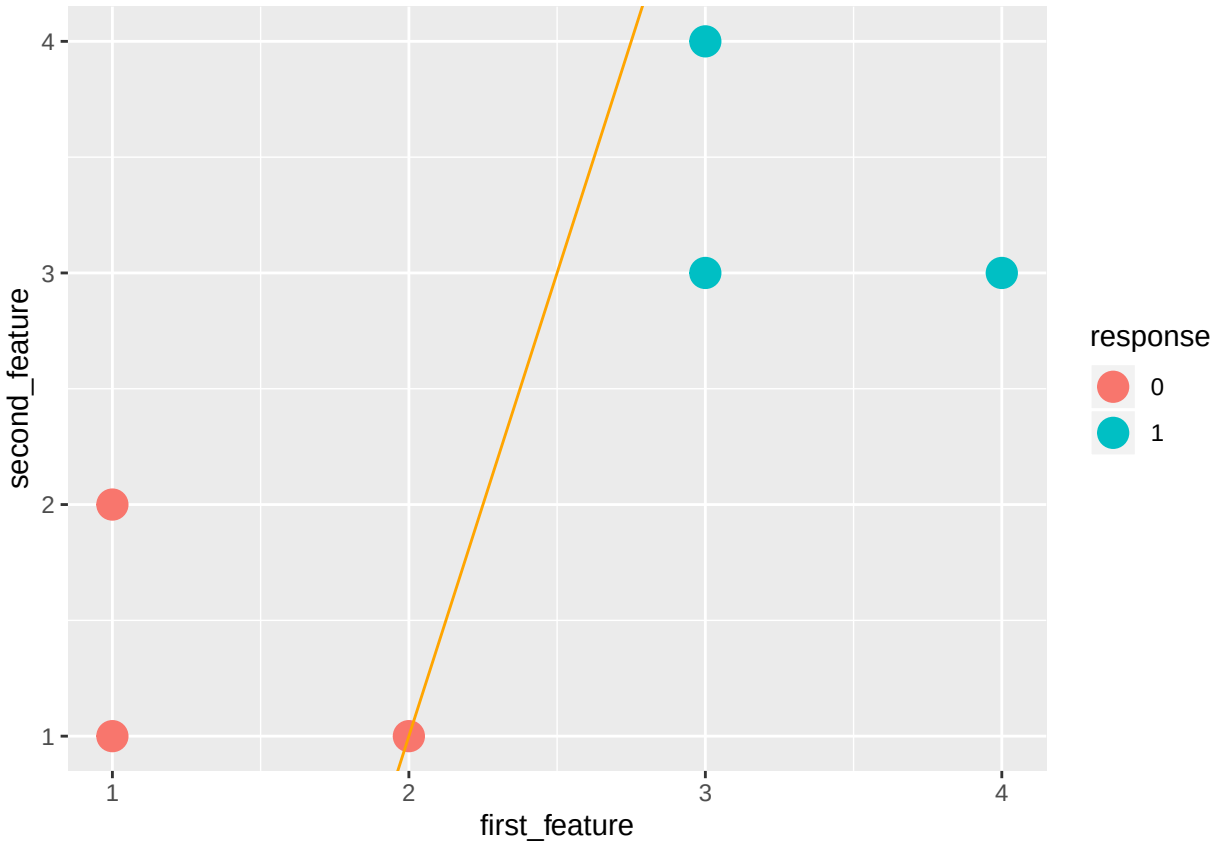
```
w_vec_simple_per = perceptron_learning_algorithm(
  cbind(Xy_simple$first_feature, Xy_simple$second_feature),
  as.numeric(Xy_simple$response == 1))
w_vec_simple_per
```

```
## [1] -7  4 -1
```

Explain this output. What do the numbers mean? What is the intercept of this line and the slope? You will have to do some algebra.

The intercept is -7 and the slope is 4. Responses above the line x2 = 4x1 + 7 will be 0 and responses below the line will be 1.

```
simple_perceptron_line = geom_abline(
    intercept = -w_vec_simple_per[1] / w_vec_simple_per[3],
    slope = -w_vec_simple_per[2] / w_vec_simple_per[3],
    color = "orange", na.rm = TRUE)
simple_viz_obj + simple_perceptron_line
```



Explain this picture. Why is this line of separation not "satisfying" to you?

The perceptron line is not the "best" way to separate the data.

For extra credit, program the maximum-margin hyperplane perceptron that provides the best linear discrimination model for linearly separable data. Make sure you provide ROxygen documentation for this function.

```
#TO-DO
```

Support Vector Machine

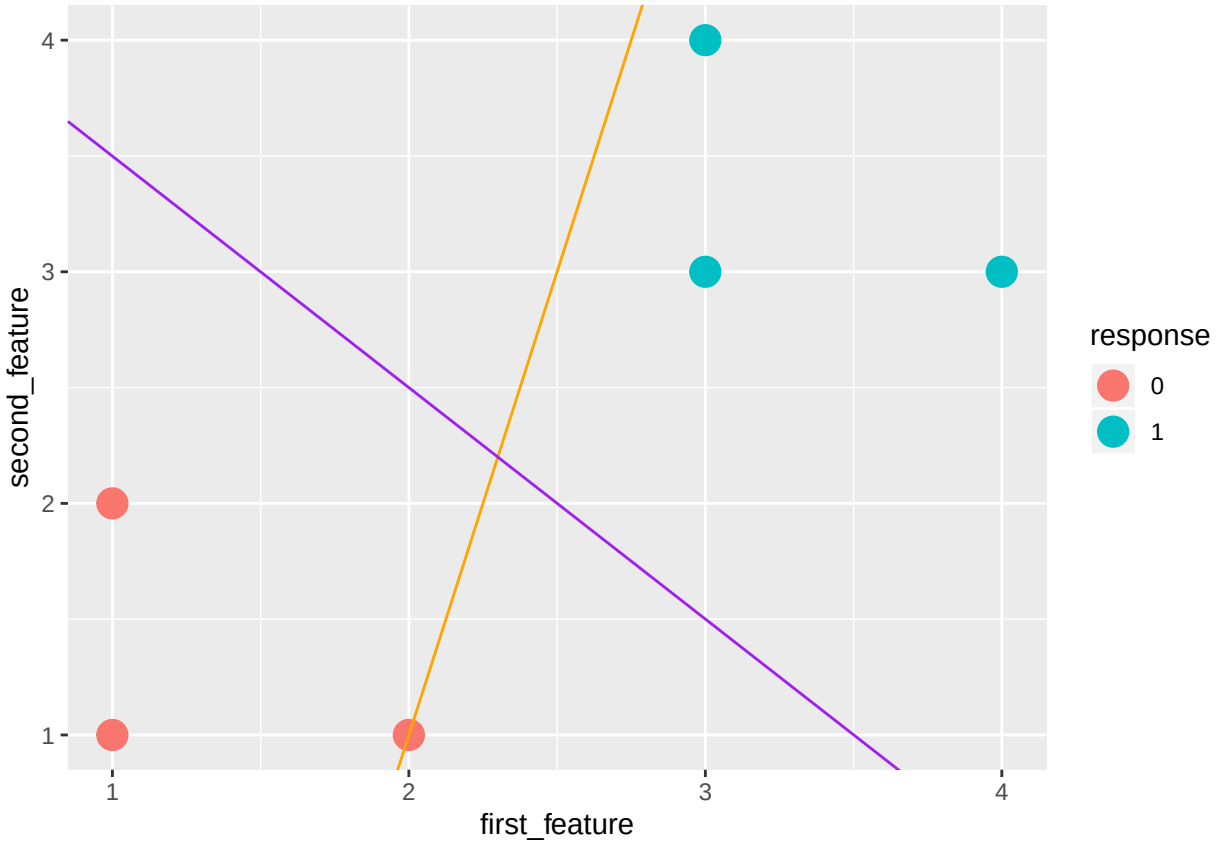
```
X_simple_feature_matrix = as.matrix(Xy_simple[, 2 : 3])
y_binary = as.numeric(Xy_simple$response == 1)
```

Use the `e1071` package to fit an SVM model to `y_binary` using the features in `X_simple_feature_matrix`. Do not specify the λ (i.e. do not specify the `cost` argument). Call the model object `svm_model`. Otherwise the remaining code won't work.

```
svm_model = e1071::svm(X_simple_feature_matrix, Xy_simple$response, kernel = "linear", scale = FALSE)
```

and then use the following code to visualize the line in purple:

```
w_vec_simple_svm = c(
  svm_model$rho, #the b term
  -t(svm_model$coefs) %*% X_simple_feature_matrix[svm_model$index, ] # the other terms
)
simple_svm_line = geom_abline(
    intercept = -w_vec_simple_svm[1] / w_vec_simple_svm[3],
    slope = -w_vec_simple_svm[2] / w_vec_simple_svm[3],
    color = "purple")
simple_viz_obj + simple_perceptron_line + simple_svm_line
```



Is this SVM line a better fit than the perceptron?

Yes.

3. Now write pseuocode for your own implementation of the linear support vector machine algorithm using the Vapnik objective function we discussed.

Note there are differences between this spec and the perceptron learning algorithm spec in question #1. You should figure out a way to respect the `MAX_ITER` argument value.

```
#' Support Vector Machine
#
#' This function implements the hinge-loss + maximum margin linear support vector machine algorithm of
#'
#' @param Xinput      The training data features as an n x p matrix.
```

```
#' @param y_binary    The training data responses as a vector of length n consisting of only 0's and 1'
#' @param MAX_ITER    The maximum number of iterations the algorithm performs. Defaults to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the hyperplane versus average hinge
#'                    The default value is 1.
#' @return            The computed final parameter (weight) as a vector of length p + 1
linear_svm_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 5000, lambda = 0.1){
  # she <- 0
  # for (i in 1:nrow(i)){
  #   she <- sum + max(0, 0.5 - (y_binary[i] - 0.5) * (w %*% Xinput[i, ] - b))
  # }
  # med <- min(she/n + lambda*(norm(w)^2))
  # med
}
```

If you are enrolled in 390 the following is extra credit but if you're enrolled in 650, the following is required. Write the actual code. You may want to take a look at the `optimx` package we discussed in class. You can feel free to define another function (a "private" function) in this chunk if you wish. R has a way to create public and private functions, but I believe you need to create a package to do that (beyond the scope of this course).

```
#' This function implements the hinge-loss + maximum margin linear support vector machine algorithm of
#'
#' @param Xinput      The training data features as an n x p matrix.
#' @param y_binary    The training data responses as a vector of length n consisting of only 0's and 1'
#' @param MAX_ITER    The maximum number of iterations the algorithm performs. Defaults to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the hyperplane versus average hinge
#'                    The default value is 1.
#' @return            The computed final parameter (weight) as a vector of length p + 1
linear_svm_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 5000, lambda = 0.1){
  #TO-DO
}
```

If you wrote code (the extra credit), run your function using the defaults and plot it in brown vis-a-vis the previous model's line:

```
# svm_model_weights = linear_svm_learning_algorithm(X_simple_feature_matrix, y_binary)
# my_svm_line = geom_abline(
#     intercept = svm_model_weights[1] / svm_model_weights[3],#NOTE: negative sign removed from interce
#     slope = -svm_model_weights[2] / svm_model_weights[3],
#     color = "brown")
# simple_viz_obj  + my_svm_line
```

Is this the same as what the `e1071` implementation returned? Why or why not?

4. Write a $k = 1$ nearest neighbor algorithm using the Euclidean distance function. Respect the spec below:

```
#' This function implements the nearest neighbor algorithm.
#'
#' @param Xinput      The training data features as an n x p matrix.
#' @param y_binary    The training data responses as a vector of length n consisting of only 0's and 1'
#' @param Xtest       The test data that the algorithm will predict on as a n* x p matrix.
```

```
#' @return            The predictions as a n* length vector.
nn_algorithm_predict = function(Xinput, y_binary, Xtest){
  n <- nrow(Xinput)
  distances <- c()
  min <- 1
  for(i in 1:n){
    distances <- c(distances, sum((Xinput[i, ] - Xtest)^2))
    if (distances[i] < distances[min]) {
      min <- i
    }
  }
  y_binary[min]
}
```

Write a few tests to ensure it actually works:

```
nn_algorithm_predict(X_simple_feature_matrix, y_binary, c(1, 2))
```

```
## [1] 0
```

```
nn_algorithm_predict(X_simple_feature_matrix, y_binary, c(1, 1))
```

```
## [1] 0
```

```
nn_algorithm_predict(X_simple_feature_matrix, y_binary, c(4, 3))
```

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

We now add an argument `d` representing any legal distance function to the `nn_algorithm_predict` function. Update the implementation so it performs NN using that distance function. Set the default function to be the Euclidean distance in the original function. Also, alter the documentation in the appropriate places.

```
#' This function implements the nearest neighbor algorithm.
#'
#' @param Xinput      The training data features as an n x p matrix.
#' @param y_binary    The training data responses as a vector of length n consisting of only 0's and 1'
#' @param Xtest       The test data that the algorithm will predict on as a n* x p matrix.
#' @param d           The distance metric with parameters Xinput[i, ] and Xtest, default = Euclidean.
#' @return            The predictions as a n* length vector.
nn_algorithm_predict_d = function(Xinput, y_binary, Xtest,
                                  d = function(Xin, Xt){
                                    sum((Xin - Xt)^2)
                                  }){
  n <- nrow(Xinput)
  distances <- c()
  min <- 1
  for(i in 1:n){
    distances <- c(distances, d(Xinput[i, ], Xtest))
    if(distances[i] < distances[min]) {
      min <- i
    }
```

```
  }
  y_binary[min]
}
```

For extra credit (unless you're a masters student), add an argument `k` to the `nn_algorithm_predict` function and update the implementation so it performs KNN. In the case of a tie, choose $\hat{y}$ randomly. Set the default `k` to be the square root of the size of $\mathcal{D}$ which is an empirical rule-of-thumb popularized by the "Pattern Classification" book by Duda, Hart and Stork (2007). Also, alter the documentation in the appropriate places.

```
#TO-DO --- extra credit for undergrads
```