Lab 2

Jaime Lin

11:59PM February 25, 2021

More Basic R Skills

- Create a function `my_reverse` which takes as required input a vector and returns the vector in reverse where the first entry is the last entry, etc. No function calls are allowed inside your function otherwise that would defeat the purpose of the exercise! (Yes, there is a base R function that does this called `rev`). Use `head` on `v` and `tail` on `my_reverse(v)` to verify it works.

```
#TO-DO
my_reverse = function(v){
  v_rev = rep(NA, times = length(v))
  for(i in length(v):1){
    v_rev[length(v)-i + 1]=v[i]
  }
  v_rev
}
v = 1:10
my_reverse(v)

##  [1] 10  9  8  7  6  5  4  3  2  1
```

- Create a function `flip_matrix` which takes as required input a matrix, an argument `dim_to_rev` that returns the matrix with the rows in reverse order or the columns in reverse order depending on the `dim_to_rev` argument. Let the default be the dimension of the matrix that is greater.

```
#TO-DO
flip_matrix = function(x, dim_to_rev = NULL){
  if(is.null(dim_to_rev)){
    dim_to_rev = ifelse(nrow(x)>=ncol(x),"rows", "cols")
  }
  if(dim_to_rev == "rows"){
    x[my_reverse(1:nrow(x)), ]
  }else if(dim_to_rev == "cols"){
    x[,my_reverse(1:ncol(x))]
  }else stop("Illegal arg")
}
x = matrix(rnorm(100), nrow = 25)
x

##              [,1]       [,2]       [,3]        [,4]
##  [1,]  1.42413801  0.4547490 -0.5274168  0.45419771
##  [2,]  0.03186961 -0.4346660 -1.4415994 -0.88625564
##  [3,] -0.45010328  1.7209707  0.3533761 -0.09381942
##  [4,]  0.26284162 -0.7446406  0.7842840  1.78959429
##  [5,]  0.41746395  0.5302932 -0.5325928 -0.29359391
##  [6,]  0.20285436  0.8070412 -0.4474275  0.42547621
##  [7,] -0.48438803  1.0942533  1.4983695 -1.76335473
##  [8,]  0.37318988  1.2382612 -0.9712718 -0.17319147
##  [9,] -1.51810034 -0.1995811 -0.7780174 -0.69164487
## [10,]  1.20800818  1.3905840  1.1287350 -0.71963728
## [11,]  2.13422373 -0.5092465 -1.2686542  0.45978991
## [12,]  1.43536300  0.3056549 -1.0966651 -1.12242443
## [13,]  0.92941646 -0.4922364 -0.1293171  1.88756789
## [14,]  1.07677484  1.1030796  0.4036668 -1.69606801
## [15,]  0.98862340 -0.2891479 -0.6885606 -2.10780629
## [16,]  0.86627176  0.3543572 -1.5982313 -1.25007953
## [17,]  0.10840311 -1.7088621 -0.4243927  0.80871031
## [18,] -1.27449185 -0.4379589  1.3682085  0.27379761
## [19,] -0.09846199  0.4482672  1.3167703  0.40362517
## [20,] -1.26207768 -1.4176671 -0.6725240 -0.25681171
## [21,] -0.29135145  0.1994906  0.6679605 -0.31784812
## [22,]  0.69215471  1.0391853  1.2979483 -0.11788909
## [23,] -0.70426627 -0.2778769 -0.8000485  3.16450819
## [24,] -0.94264752  0.6654488  0.6350874 -0.66506028
## [25,]  0.16238735 -0.9531210 -1.2690835 -0.69956619
```

```
flip_matrix(x, dim_to_rev = "cols")
```

```
##              [,1]       [,2]       [,3]        [,4]
##  [1,]  0.45419771 -0.5274168  0.4547490  1.42413801
##  [2,] -0.88625564 -1.4415994 -0.4346660  0.03186961
##  [3,] -0.09381942  0.3533761  1.7209707 -0.45010328
##  [4,]  1.78959429  0.7842840 -0.7446406  0.26284162
##  [5,] -0.29359391 -0.5325928  0.5302932  0.41746395
##  [6,]  0.42547621 -0.4474275  0.8070412  0.20285436
##  [7,] -1.76335473  1.4983695  1.0942533 -0.48438803
##  [8,] -0.17319147 -0.9712718  1.2382612  0.37318988
##  [9,] -0.69164487 -0.7780174 -0.1995811 -1.51810034
## [10,] -0.71963728  1.1287350  1.3905840  1.20800818
## [11,]  0.45978991 -1.2686542 -0.5092465  2.13422373
## [12,] -1.12242443 -1.0966651  0.3056549  1.43536300
## [13,]  1.88756789 -0.1293171 -0.4922364  0.92941646
## [14,] -1.69606801  0.4036668  1.1030796  1.07677484
## [15,] -2.10780629 -0.6885606 -0.2891479  0.98862340
## [16,] -1.25007953 -1.5982313  0.3543572  0.86627176
## [17,]  0.80871031 -0.4243927 -1.7088621  0.10840311
## [18,]  0.27379761  1.3682085 -0.4379589 -1.27449185
## [19,]  0.40362517  1.3167703  0.4482672 -0.09846199
## [20,] -0.25681171 -0.6725240 -1.4176671 -1.26207768
## [21,] -0.31784812  0.6679605  0.1994906 -0.29135145
## [22,] -0.11788909  1.2979483  1.0391853  0.69215471
## [23,]  3.16450819 -0.8000485 -0.2778769 -0.70426627
## [24,] -0.66506028  0.6350874  0.6654488 -0.94264752
## [25,] -0.69956619 -1.2690835 -0.9531210  0.16238735
```

- Create a list named `my_list` with keys “A”, “B”, … where the entries are arrays of size 1, 2 x 2, 3 x 3 x 3, etc. Fill the array with the numbers 1, 2, 3, etc. Make 8 entries according to this sequence.

```
#TO-DO
my_list = LETTERS
arrays = list()
arrays[["A"]]=array(data = 1:4, dim = c(2,2))
arrays[["B"]]=array(data = 1:27, dim = c(3,3,3))
arrays[["C"]]=array(data = 1:(4^4), dim = c(4,4,4,4))
arrays[["D"]]=array(data = 1:(5^5), dim = c(5,5,5,5,5))
arrays[["E"]]=array(data = 1:(6^6), dim = c(6,6,6,6,6,6))
arrays[["F"]]=array(data = 1:(7^7), dim = c(7,7,7,7,7,7,7))
arrays[["G"]]=array(data = 1:(8^8), dim = c(8,8,8,8,8,8,8,8))
arrays[["H"]]=array(data = 1:(9^9), dim = c(9,9,9,9,9,9,9,9,9))
```

Run the following code:

```
lapply(my_list, object.size)

## [[1]]
## 112 bytes
##
## [[2]]
## 112 bytes
##
## [[3]]
## 112 bytes
##
## [[4]]
## 112 bytes
##
## [[5]]
## 112 bytes
##
## [[6]]
## 112 bytes
##
## [[7]]
## 112 bytes
##
## [[8]]
## 112 bytes
##
## [[9]]
## 112 bytes
##
```

```
## [[10]]
## 112 bytes
##
## [[11]]
## 112 bytes
##
## [[12]]
## 112 bytes
##
## [[13]]
## 112 bytes
##
## [[14]]
## 112 bytes
##
## [[15]]
## 112 bytes
##
## [[16]]
## 112 bytes
##
## [[17]]
## 112 bytes
##
## [[18]]
## 112 bytes
##
## [[19]]
## 112 bytes
##
## [[20]]
## 112 bytes
##
## [[21]]
## 112 bytes
##
## [[22]]
## 112 bytes
##
## [[23]]
## 112 bytes
##
## [[24]]
## 112 bytes
##
## [[25]]
## 112 bytes
##
## [[26]]
## 112 bytes
```

Use `?object.size` to read about what these functions do. Then explain the output you see above. For the later arrays, does it make sense given the dimensions of the arrays?

```
?object.size

## starting httpd help server ... done
```

#TO-DO `The object.size` shows the space that is ocupy in the memory. The array shows that each letter has the same size. Now cleanup the namespace by deleting all stored objects and functions:

```
?rm
rm(my_list)
```

A little about strings

- Use the `strsplit` function and `sample` to put the sentences in the string `lorem` below in random order. You will also need to manipulate the output of `strsplit` which is a list. You may need to learn basic concepts of regular expressions.

```
lorem = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi
posuere varius volutpat. Morbi faucibus ligula id massa ultricies viverra.
Donec vehicula sagittis nisi non semper. Donec at tempor erat. Integer
dapibus mi lectus, eu posuere arcu ultricies in. Cras suscipit id nibh
lacinia elementum. Curabitur est augue, congue eget quam in, scelerisque
semper magna. Aenean nulla ante, iaculis sed vehicula ac, finibus vel arcu.
Mauris at sodales augue. "

?strsplit
strsplit(lorem, split = ".", fixed = TRUE)

## [[1]]
##  [1] "Lorem ipsum dolor sit amet, consectetur adipiscing elit"
##  [2] " Morbi posuere varius volutpat"
##  [3] " Morbi faucibus ligula id massa ultricies viverra"
##  [4] " Donec vehicula sagittis nisi non semper"
##  [5] " Donec at tempor erat"
##  [6] " Integer dapibus mi lectus, eu posuere arcu ultricies in"
##  [7] " Cras suscipit id nibh lacinia elementum"
##  [8] " Curabitur est augue, congue eget quam in, scelerisque semper magna"
##  [9] " Aenean nulla ante, iaculis sed vehicula ac, finibus vel arcu"
## [10] " Mauris at sodales augue"
## [11] " "
```

You have a 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.

```
#HINT:
#strsplit("Theodore, Bernard, Gene, Herbert, Ray, Tom, Lee, Alfred, Leroy,
Eddie", split = ", ")[[1]]
#TO-DO
# M is masculine and F is female
# B is Boomer, G is Genx and Mi is Millenial
M_B = strsplit("Theodore, Bernard, Gene, Herbert, Ray, Tom, Lee, Alfred,
Leroy, Eddie", split = ", ")[[1]]
M_G = strsplit("Marc, Jamie, Greg, Darryl, Tim, Dean, Jon, Chris, Troy,
Jeff", split = ", ")[[1]]
M_Mi = strsplit("Zachary, Dylan, Christian, Wesley, Seth, Austin, Gabriel,
Evan, Casey, Luis", split = ", ")[[1]]
F_B = strsplit("Gloria, Joan, Dorothy, Shirley, Betty, Dianne, Kay, Marjorie,
Lorraine, Mildred", split = ", ")[[1]]
F_G = strsplit("Tracy, Dawn, Tina, Tammy, Melinda, Tamara, Tracey, Colleen,
Sherri, Heidi", split = ", ")[[1]]
F_Mi = strsplit("Samantha, Alexis, Brittany, Lauren, Taylor, Bethany, Latoya,
Candice, Brittney, Cheyenne", split = ", ")[[1]]
Classification= list(M_B, M_G, M_Mi, F_B, F_G, F_Mi)
```

Dataframe creation

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),
#                 ...
#TO-DO
```

Packages

Install the package `pacman` using regular base R.

```
install.packages("pacman", repos = "http://cran.us.r-project.org")
```

```
## package 'pacman' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\jaime\AppData\Local\Temp\RtmpiQ3BYj\downloaded_packages
```

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 su

```
v= seq(-100, 100)
#expect_equal(v, -100 : 101)
```

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:

```
v = 1:100
expect_equal(my_reverse(v), rev(v))
expect_equal(my_reverse(c("A", "B", "C")), c("C", "B", "A"))
```

Multinomial Classification using KNN

Write a $k = 1$ nearest neighbor algorithm using the Euclidean distance 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.

```
#' One Nearest Neighbors Classifier
#'
#' Classify and observation based on the label of the closest observation in
the set of training observations.
#'
#' @param Xinput      A matrix of features of training data observations.
#' @param y_binary    The vector of training data labels.
#' @param xtest       A test observation as a row vector.
#' @return            The predictive label for the test observation.
nn_algorithm_predict = function(Xinput, y_binary, Xtest){
  n=nrow(Xinput)
  distances=array(NA,n)
  for (i in 1:n) {
    distances[i]=sum((Xinput[i, ]-Xtest)^2)
  }
  y_binary[which.min(distances)]
}
```

Write a few tests to ensure it actually works:

```
#TO-DO
```

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.

```
#' One Nearest Neighbors Classifier
#'
#' Classify and observation based on the label of the closest observation in
the set of training observations.
#'
#' @param Xinput      A matrix of features of training data observations.
#' @param y_binary    The vector of training data labels.
#' @param xtest       A test observation as a row vector.
#' @param d           A distance function which take inputs to different row
vectors
#' @return            The predictive label for the test observation.
nn_algorithm_predict = function(Xinput, y_binary, Xtest, d =
function(v1,v2){sum((v1 - v2)^2)}){
  n=nrow(Xinput)
  distances=array(NA,n)
  for (i in 1:n) {
    distances[i]= d(Xinput[i,],xtest)
  }
  y_binary[which.min(distances)]
}
```

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 for the 650 students but extra credit for undergrads
```

Basic Binary Classification Modeling

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

```
#TO-DO
data(iris)
pacman::p_load(skimr)
skim(iris)
```

Data summary

Name	iris

Number of rows	150
Number of columns	5
_______________________	
Column type frequency:	
factor	1
numeric	4
________________________	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Species	0	1	FALSE	3	set: 50, ver: 50, vir: 50

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Sepal.Length	0	1	5.84	0.83	4.3	5.1	5.80	6.4	7.9	▆▇▇▅▂
Sepal.Width	0	1	3.06	0.44	2.0	2.8	3.00	3.3	4.4	▁▆▇▂▁
Petal.Length	0	1	3.76	1.77	1.0	1.6	4.35	5.1	6.9	▇▁▆▇▂
Petal.Width	0	1	1.20	0.76	0.1	0.3	1.30	1.8	2.5	▇▁▇▅▃

TO-DO: describe this data

The outcome / label / response 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
#TO-DO
iris = iris[iris$Species != "virginica", ]
```

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
#TO-DO
y = as.integer(iris$Species == "setosa")
```

- Write a function `mode` returning the sample mode.

```r
#TO-DO
mode = function(v){
  names(sorted_count=sort(table(v), decreasing=TRUE)[1])
}
```

- 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? Save the threshold value as `threshold`.

```r
#TO-DO

threshold = mean(iris$Sepal.Length)
threshold

## [1] 5.471
```

What is the total number of errors this model makes?

```r
#TO-DO
sd(iris$Sepal.Length) / sqrt(length(iris))

## [1] 0.2869762
```

Does the threshold model's performance make sense given the following summaries:

```r
threshold

## [1] 5.471

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
```

TO-DO: Write your answer here in English.

Yes. the threshold is the sum of the mean of Sepal.Length between "setosa" and "versicolor" then divide 2.

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

```r
g = function(x){
 as.integer(x <= iris$Sepal.Length)
}
```

Perceptron

You will code the "perceptron learning algorithm" for arbitrary number of features p. Take a look at the comments above the function. Respect the spec below:

```
#' Learning about perceptron
#'
#' This function is going to elaborate a table
#'
#' @param Xinput      A matrix of features of training data observations.
#' @param y_binary    The vector of training data labels.
#' @param MAX_ITER    This going to represent the space
#' @param w           dimension
#'
#' @return            The computed final parameter (weight) as a vector of
length p + 1
perceptron_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 1000, w
= NULL){
  #TO-DO
  p=runif(Xinput)
  n=runif(y_binary)

}
```

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
)
```

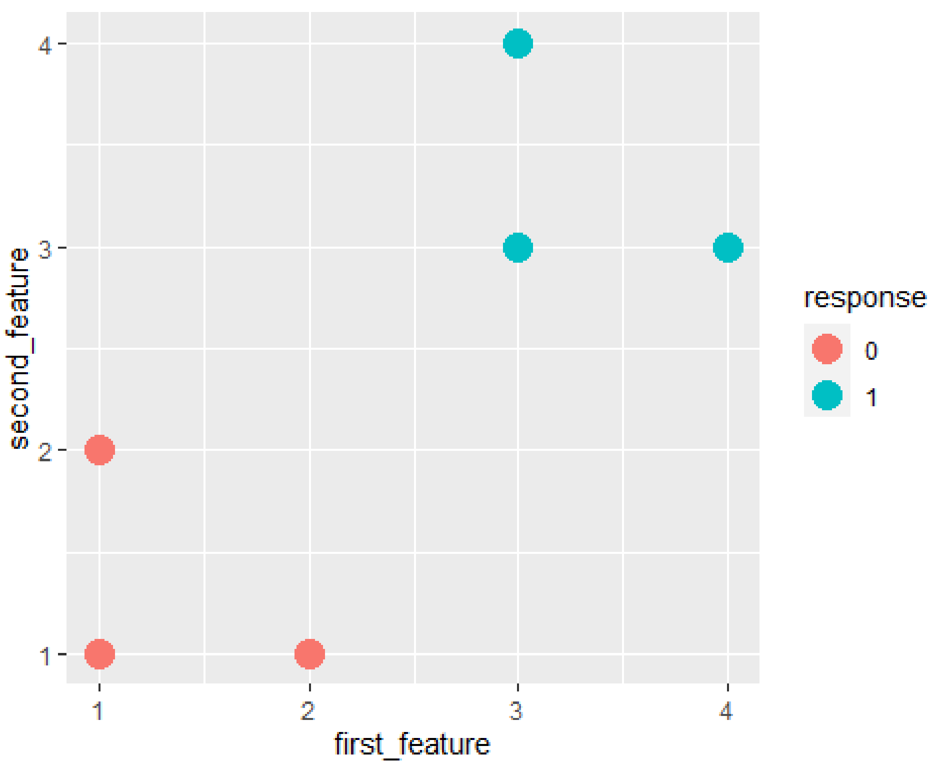
We haven’t spoken about visualization yet, but it is important we do some of it now. Thus, I will write this code for you and you will just run it. First we load the visualization library we’re going to use:

```
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
```



TO-DO: Explain this picture. In the plane we can see different points that represent the response. Also, we can see how the red dot move to another coordinates while maintaining the same imaginary triangle.

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

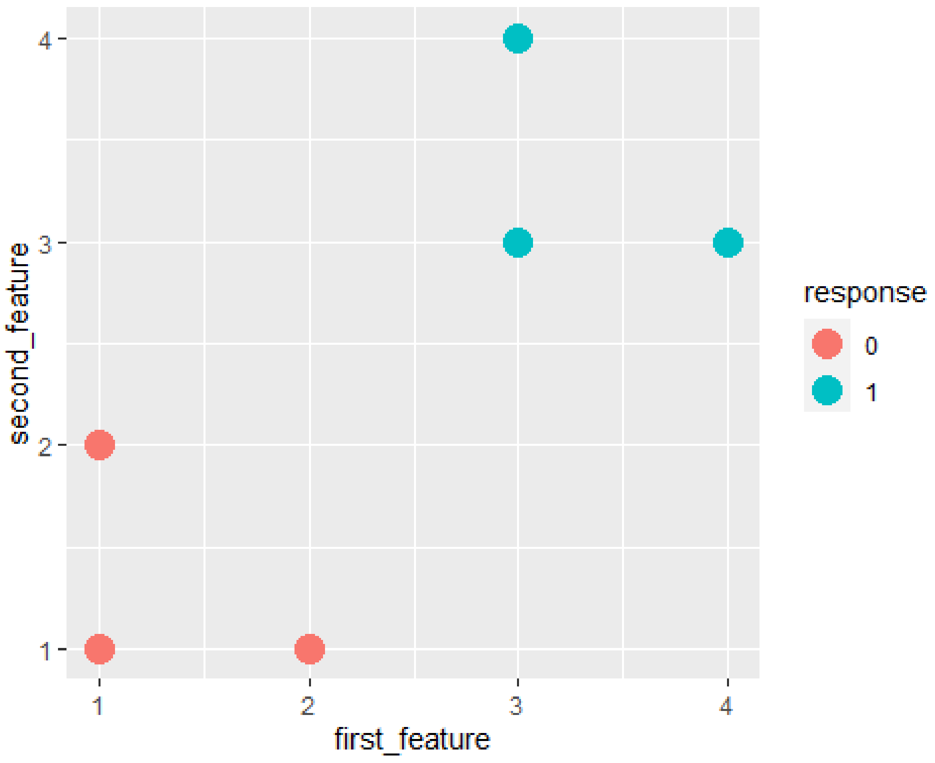
```
w_vec_simple_per = perceptron_learning_algorithm(
  cbind(Xy_simple$first_feature, Xy_simple$second_feature),
  as.numeric(Xy_simple$response == 1))
w_vec_simple_per

## [1] 0.20737746 0.09738597 0.89901068 0.96673848 0.97330340 0.40173435
```

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.

TO-DO This represent the distance of to the line. Slope = m, (1,1) and (4,3) m = (y_2-y_1) / (x_2-x_1) m = (3-1) / (4-1) m = 2/3 = 0.67

Intercept line 3(y-3) = 2(x-4) 3y-9 = 2x -8 -2x+3y-1=0

```
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")
simple_viz_obj + simple_perceptron_line
```



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

TO-DO Because the dots are far of the line making to have errors.

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
```