

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 = function(Xinput , y_binary , MAX_ITER = 1000 , w = NULL){
  p= ncol(Xinput)
  n= nrow(Xinput)
  if(is.null(w)){
    w = runif(ncol(Xinput))
  }

  for(iter in 1 : MAX_ITER){
    for(i in 1 : nrow(Xinput)){
      x_i = Xinput[i , ]
      yhat_i = ifelse(sum(x_i * w)>0 , 1, 0)
      w = w + as.numeric(y_binary[i] - yhat_i) * x_i
    }
  }
  w
}

#' TO-DO: Provide a name for this function

#'
#' TO-DO: Explain what this function does in a few sentences
#'
#' @param Xinput      TO-DO: Explain this; matrix size N x P
#' @param y_binary     TO-DO: Explain this: its a binary vector of size n
#' @param MAX_ITER     TO-DO: Explain this: numeric times to excute the algo
#' @param w            TO-DO: Explain this; intial weight vector of length p+1 if unspecfied it will be
#'
#' @return            The computed final parameter (weight) as a vector of length p + 1
#' @export
#' @author            [Burhan Hanif]
```

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

We haven’t spoken about visualization yet, but it is important we do some of it now. First we load the visualization library we’re going to use:

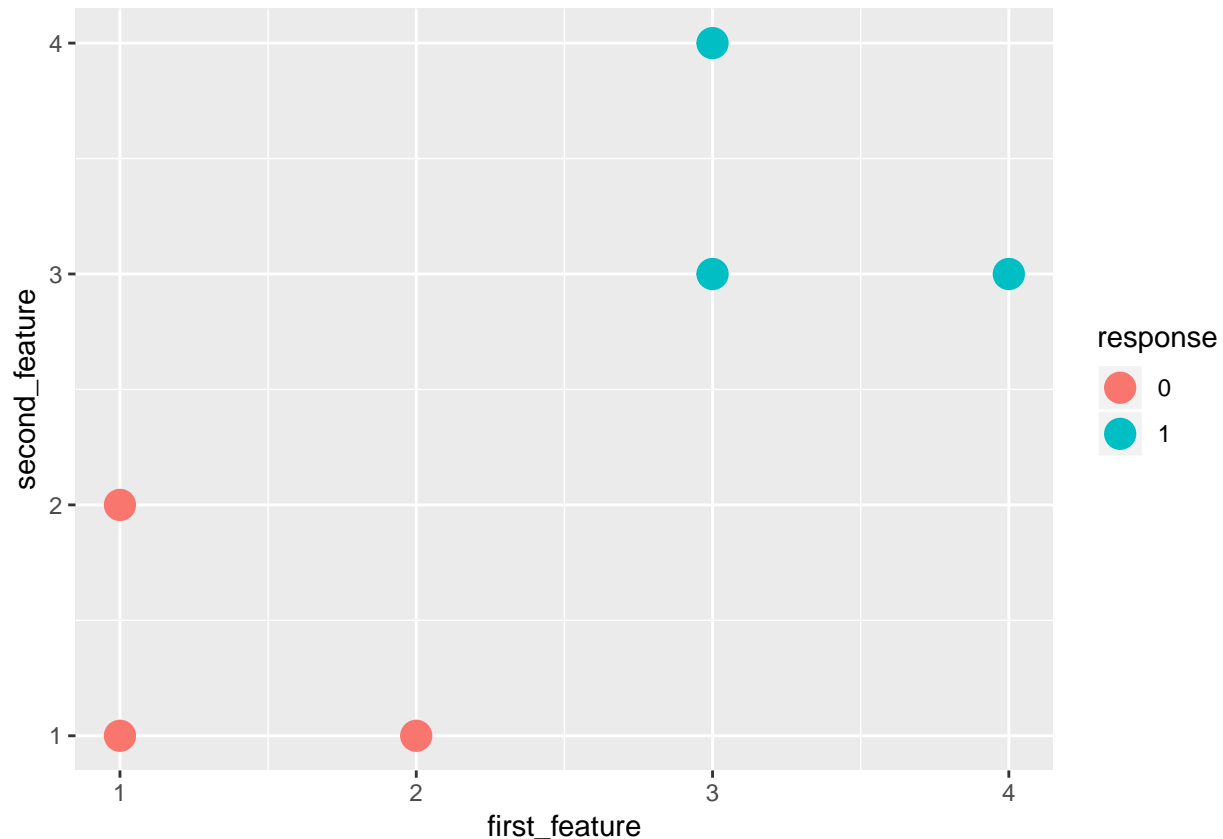
```
pacman::p_load(ggplot2)
```

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. # this is the graph of our training data, its binary and its linearly separable.

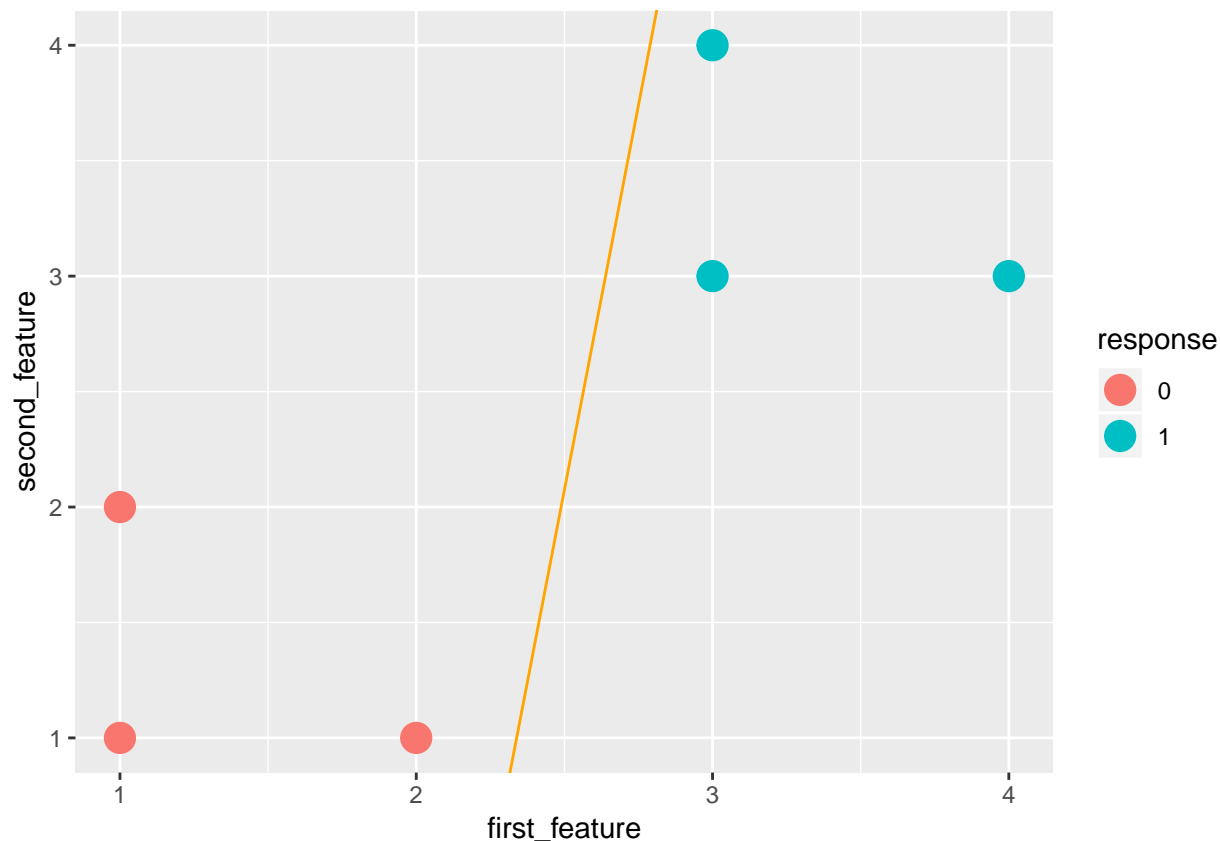
Now, let us run the algorithm and see what happens:

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

```
## [1] -7.7546067  3.5430203 -0.5317233
```

TO-DO: 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. # these are the weights in the w vector,

```
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
```



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

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 = #TO-DO
```

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

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

Is this SVM line a better fit than the perceptron?

TO-DO

3. Now write pseudocode for your own implementation of the linear support vector machine algorithm

respecting the following spec making use of the nelder mead `optimx` function from lecture 5p. It turns out you do not need to load the package `neldermead` to use this function. You can feel free to define a function within this function if you wish.

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's
#' @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: write pseudo code in comments
  #defining the p of n columns of Xinputs
  # the Xinput with the vectors of p + 1
  # w_0 gives the 0s vector as p+1 iterates
  #find the average hinge loss with maxima func and then sum the zeros , subtract the .5with the y_i's
  # find the maximum margin by multiplying lambda with w squared
  # apply minima with the sum of the average hinge error and the maximum margin
}
```

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.

```
#' 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's
#' @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 intercept argument here 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's.
#' @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){
  classify = c(rep(NA , nrow(Xtest)))
  i_star = c(rep(nrow(Xtest)))
  for(k in 1 : nrow(Xinput)){
    best_sqd_distance = Inf
    for(i in 1 : ncol(Xinput)){
      totsqd = 0
      for (j in 1 : ncol(Xinput)) {
        dsqd = (Xinput[i , j] - Xtest [k,j])^2
        totdsqd = totsqd + dsqd
      }
      if(dsqd< best_sqd_distance){
        best_sqd_distance = dsqd
        istar[k]= i
      }
    }
    classify[k]= y_binary[i_star[k]]
  }
  classify
}

```

Write a few tests to ensure it actually works:

```

Xy = na.omit(MASS :: biopsy)
X = Xy[, 2:10]
y_binary = as.numeric(Xy$class == "malignant")
for (i in 1:9){
  z = factor(X[ , i])
}

```

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.

```

Euclid_Dist= function(X_1,X_2){
  ((X_1 - X_2)^2)
}
nn_algorithm_predict_d= function(Xinput , y_binary , Xtest , d = Euclid_Dist){
  prediction = c(rep(NA, nrow(Xtest)))
  i_star = c(rep(NA , nrow(Xtest)))
  for (k in 1 : nrow(Xtest)) {
    best_sqd_distance = Inf
    for (i in 1: nrow(Xinput)) {
      total_dsqd = 0
      for (j in 1 : ncol(Xinput)) {
        dsqd = Euclid_Dist(Xinput [i,j], Xtest[k,j])
        totald_sqd= totald_sqd + dsqd
      }
      if(dsqd < best_sqd_distance){
        best_sqd_distance = dsqd
      }
    }
    i_star[k]= i
    prediction[k]= y_binary[i_star[k]]
  }
  prediction
}

```

```

        i_star[k]= i
    }

    }
    prediction[k]= y_binary[i_star[k]]

}
prediction
}

```

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

© 2019 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Help](#) [Contact](#) [GitHub](#) [Pricing](#) [API](#) [Training](#) [Blog](#) [About](#)