HW02p

Joseph Huaynate

March 6, 2018

```r
knitr::opts_chunk$set(error = TRUE) #this allows errors to be printed into the PDF
```

Welcome to HW02p where the "p" stands for "practice" meaning you will use R to solve practical problems. This homework is due 11:59 PM Tuesday 3/6/18.

You should have RStudio installed to edit this file. You will write code in places marked "TO-DO" to complete the problems. Some of this will be a pure programming assignment. Sometimes you will have to also write English.

The tools for the solutions to these problems can be found in the class practice lectures. I want you to use the methods I taught you, not for you to google and come up with whatever works. You won't learn that way.

To "hand in" the homework, you should compile or publish this file into a PDF that includes output of your code. To do so, use the knit menu in RStudio. You will need LaTeX installed on your computer. See the email announcement I sent out about this. Once it's done, push the PDF file to your github class repository by the deadline. You can choose to make this respository private.

For this homework, you will need the `testthat` libray.

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

1. Source the simple dataset from lecture 6p:

```r
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)
```

Try your best to write a general perceptron learning algorithm to the following `Roxygen` spec. For inspiration, see the one I wrote in lecture 6.

```r
#' This function implements the "perceptron learning algorithm" of Frank Rosenblatt (1957).
#'
#' @param Xinput      The training data features as an n x (p + 1) matrix where the first column is all
#' @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 perceptron algorithm performs. Defaults to 1
#' @param w           A vector of length p + 1 specifying the parameter (weight) starting point. Defaul
#'                    \code{NULL} which means the function employs random standard uniform values.
#' @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
  if (is.null(w)){
    w = runif(ncol(Xinput)) #intialize a p+1-dim vector with random values
  }
  for (iter in 1 : MAX_ITER){
    for (i in 1 : nrow(Xinput)){
      x_i = Xinput[i, ]
      yhat_i = ifelse(x_i %*% w > 0, 1, 0)
      w = w + as.numeric(y_binary[i] - yhat_i) * x_i
    }
  }
  w
}
```

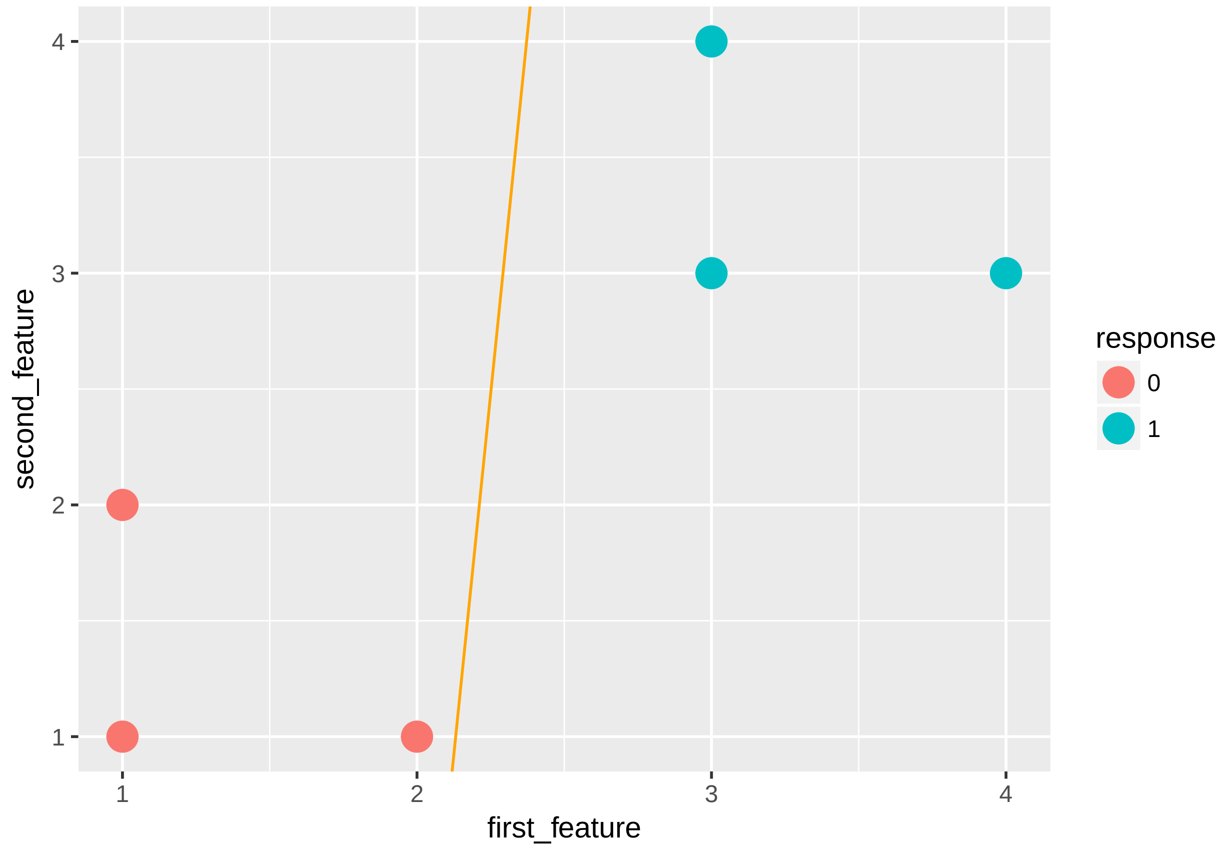
Run the code on the simple dataset above via:

```
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] -2.29317273  1.11806461 -0.08978383
```

Use the ggplot code to plot the data and the perceptron's g function.

```
pacman::p_load(ggplot2)
simple_viz_obj = ggplot(Xy_simple, aes(x = first_feature, y = second_feature, color = response)) +
  geom_point(size = 5)
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
```



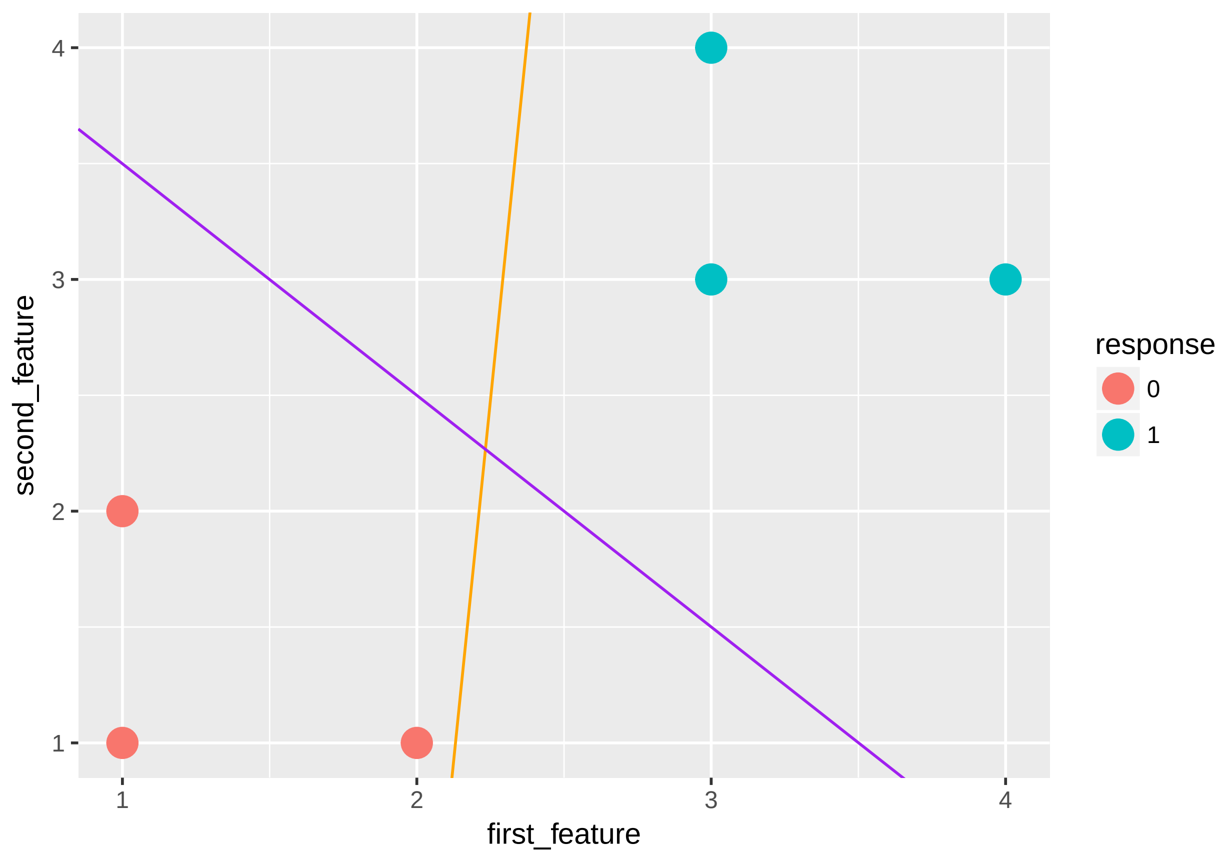
Why is this line of separation not "satisfying" to you? This line is not satisfying because probably won't work for large amounts of data points mixed together which would happen in the real world. In other words it is too clean.

2. Use the `e1071` package to fit an SVM model to `y_binary` using the predictors found in `X_simple_feature_matrix`. Do not specify the λ (i.e. do not specify the `cost` argument).

```
pacman::p_load(e1071)
Xy_simple_feature_matrix = as.matrix(Xy_simple[, 2:3])
n = nrow(Xy_simple_feature_matrix)
svm_model = svm(Xy_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? The SVM line is in the middle. It divides the separation of responses more evenly

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 `optim` 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.

For extra credit, write the actual code.

```r
#' 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){

}
```

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:

```r
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
    slope = -svm_model_weights[2] / svm_model_weights[3],
    color = "brown")
```

```
## Error in -svm_model_weights[2]: invalid argument to unary operator
```

```r
simple_viz_obj  + my_svm_line
```

```
## Error in eval(expr, envir, enclos): object 'my_svm_line' not found
```

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:

```r
#' 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){
  #TO-DO
  squared_distance = 1000
i_star = NA
for(i in 1:nrow(Xinput)) {
  euclid_distance = sqrt(((sum((Xinput[i, ] - Xtest[i,])^2))))
    if(euclid_distance < squared_distance) {
     squared_distance = euclid_distance
     i_star = squared_distance
    }
}
y = i_star
y
}
```

Write a few tests to ensure it actually works:

```r
#TO-DO
test = nn_algorithm_predict(X_simple_feature_matrix, y_binary, X_simple_feature_matrix)
expect_equal(as.vector(test), 0)
```

For extra credit, 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.

```r
#not required TO-DO --- only for extra credit
```

For extra credit, in addition to the argument `k`, add an argument `d` representing any legal distance function to the `nn_algorithm_predict` function. Update the implementation so it performs KNN 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.

```r
#not required TO-DO --- only for extra credit
```

5. We move on to simple linear modeling using the ordinary least squares algorithm.

Let's quickly recreate the sample data set from practice lecture 7:

```r
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
y = beta_0 + beta_1 * x + rnorm(n, mean = 0, sd = 0.33)
```

Solve for the least squares line by computing b_0 and b_1 *without* using the functions `cor`, `cov`, `var`, `sd` but instead computing it from the x and y quantities manually. See the class notes.

```r
#TO-DO
x_bar = mean(x)
y_bar = mean(y)
s_x = sqrt(sum((x - mean(x))^2/(length(x)-1)))
s_y = sqrt(sum((y - mean(y))^2/(length(y)-1)))
s_xy = (sum(x*y) - (length(x)*x_bar*y_bar))/(length(x)-1)
r = (s_xy)/(s_x*s_y)
b_1 = r*(s_y/s_x)
b_0 = y_bar - (b_1 * x_bar)
b_1
```

```
## [1] -1.693145
```

```r
b_0
```

```
## [1] 2.80748
```

Verify your computations are correct using the `lm` function in R:

```r
lm_mod = (lm(y ~ x))
b_vec = coef(lm_mod)
expect_equal(b_0, as.numeric(b_vec[1]), tol = 1e-4) #thanks to Rachel for spotting this bug - the b_vec
expect_equal(b_1, as.numeric(b_vec[2]), tol = 1e-4)
```

6. We are now going to repeat one of the first linear model building exercises in history — that of Sir Francis Galton in 1886. First load up package `HistData`.

```r
install.packages("HistData")
```

```
## Installing package into 'C:/Users/euphoria/Documents/R/win-library/3.4'
## (as 'lib' is unspecified)
```

```
## Error in contrib.url(repos, "source"): trying to use CRAN without setting a mirror
```

In it, there is a dataset called `Galton`. Load it using the `data` command:

```
#TO-DO
data(Galton)
```

```
## Warning in data(Galton): data set 'Galton' not found
```

You now should have a data frame in your workspace called `Galton`. Summarize this data frame and write a few sentences about what you see. Make sure you report n, p and a bit about what the columns represent and how the data was measured. See the help file `?Galton`.

```
#TO-DO
summary(Galton)
```

```
## Error in summary(Galton): object 'Galton' not found
```

```
?Galton
```

```
## No documentation for 'Galton' in specified packages and libraries:
## you could try '??Galton'
```

```
#Galton
```

Galton is a cross-tabulation of 928 adult children born to 205 fathers and mothers, by their height and their mid-parent's height. The data frame has two variables: parent, which is a numeric vector: height of the mid_parent, and child, a numeric vector:height of the child. The data was recorder in class intervals of width 1.0 in. Galton than used non-integer values for the center of each class interval because of the bias toward inches.

Find the average height (include both parents and children in this computation).

```
avg_height = (sum(Galton$parent) + sum(Galton$child))/(length(Galton$parent) + (length(Galton$child)))
```

```
## Error in eval(expr, envir, enclos): object 'Galton' not found
```

```
avg_height
```

```
## Error in eval(expr, envir, enclos): object 'avg_height' not found
```

Note that in Math 241 you learned that the sample average is an estimate of the "mean", the population expected value of height. We will call the average the "mean" going forward since it is probably correct to the nearest tenth of an inch with this amount of data.

Run a linear model attempting to explain the childrens' height using the parents' height. Use `lm` and use the R formula notation. Compute and report b_0, b_1, RMSE and R^2. Use the correct units to report these quantities.

```
#TO-DO
linear_model = lm(parent ~ child, data = Galton)
```

```
## Error in is.data.frame(data): object 'Galton' not found
```

```
#linear_model
coef(linear_model)
```

```
## Error in coef(linear_model): object 'linear_model' not found
```

```r
#TO-DO

summary(linear_model)$r.squared
```

```
## Error in summary(linear_model): object 'linear_model' not found
```

```r
#TO-DO
summary(linear_model)$sigma
```

```
## Error in summary(linear_model): object 'linear_model' not found
```

```r
b = coef(linear_model)
```

```
## Error in coef(linear_model): object 'linear_model' not found
```

Interpret all four quantities: b_0, b_1, RMSE and R^2.

b_0 : This tells us that if the child's height is 46 inches than the parent's height is 0.

b_1 : This is the slope of the model which is 0.3256475

R^2 : This says that the height of the parent accounts for 21% of the child's height.

$RMSE$: it the most common metric that regression models report.

How good is this model? How well does it predict? Discuss.

The model is not accurate according to the value of R^2. Since it is very low, it tells us that a child's height is absolutely determined by the height of their parent.

Now use the code from practice lecture 8 to plot the data and a best fit line using package `ggplot2`. Don't forget to load the library.

```r
#TO-DO
ggplot(Galton, aes(parent, child)) + geom_point() + geom_smooth(method = 'lm')
```

```
## Error in ggplot(Galton, aes(parent, child)): object 'Galton' not found
```

It is reasonable to assume that parents and their children have the same height. Explain why this is reasonable using basic biology.

If we solely look at genetics and heights of the parents than it is reasonable to consider that the height of the parents will determine the height of the child. But there are other factors before and after the child is born that can paint a different picture.

If they were to have the same height and any differences were just random noise with expectation 0, what would the values of β_0 and β_1 be?

b_0 would be 0 and b_1 is 1.

Let's plot (a) the data in $\mathbb{D}$ as black dots, (b) your least squares line defined by b_0 and b_1 in blue, (c) the theoretical line β_0 and β_1 if the parent-child height equality held in red and (d) the mean height in green.

```r
ggplot(Galton, aes(x = parent, y = child)) +
  geom_point() +
  geom_jitter() +
  geom_abline(intercept = b[1], slope = b[2], color = "blue", size = 1) +
  geom_abline(intercept = 0, slope = 1, color = "red", size = 1) +
  geom_abline(intercept = avg_height, slope = 0, color = "darkgreen", size = 1) +
  xlim(63.5, 72.5) +
  ylim(63.5, 72.5) +
  coord_equal(ratio = 1)
```

```
## Error in ggplot(Galton, aes(x = parent, y = child)): object 'Galton' not found
```

```
# b_0 = b[1] and b_1 = b[2]
```

Fill in the following sentence:

TO-DO: Children of short parents became short on average and children of tall parents became tall on average.

Why did Galton call it "Regression towards mediocrity in hereditary stature" which was later shortened to "regression to the mean"?

Perhaps because as we can see the trend gets closer to the predicated average over time.

Why should this effect be real?

If measurements are done multiple times then there will be an average to which the measurements will lead too. This is probably why the word "mediocre" is a synonym for average.

You now have unlocked the mystery. Why is it that when modeling with y continuous, everyone calls it "regression"? Write a better, more descriptive and appropriate name for building predictive models with y continuous.

An more appropriate name for y would probably be 'prediction'. I'd say regression refers to the explaination of the variables being cross-examined.