HW02p

Evangeline Szpylka

March 6, 2018

```
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` library.

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

1. Source the simple dataset from lecture 6p:

```
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.

```
#' 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){
  if(is.null(w)){
    w = runif(ncol(Xinput))
  }
  for(j in 1:MAX_ITER){
    for(k in 1:nrow(Xinput)){
      x_k = Xinput[k,]
      yhat_k = ifelse(x_k %*% w > 0, 1, 0)
      w = w + as.numeric(y_binary[k] - yhat_k) * x_k
```

```
    }
  }
  w
}
```

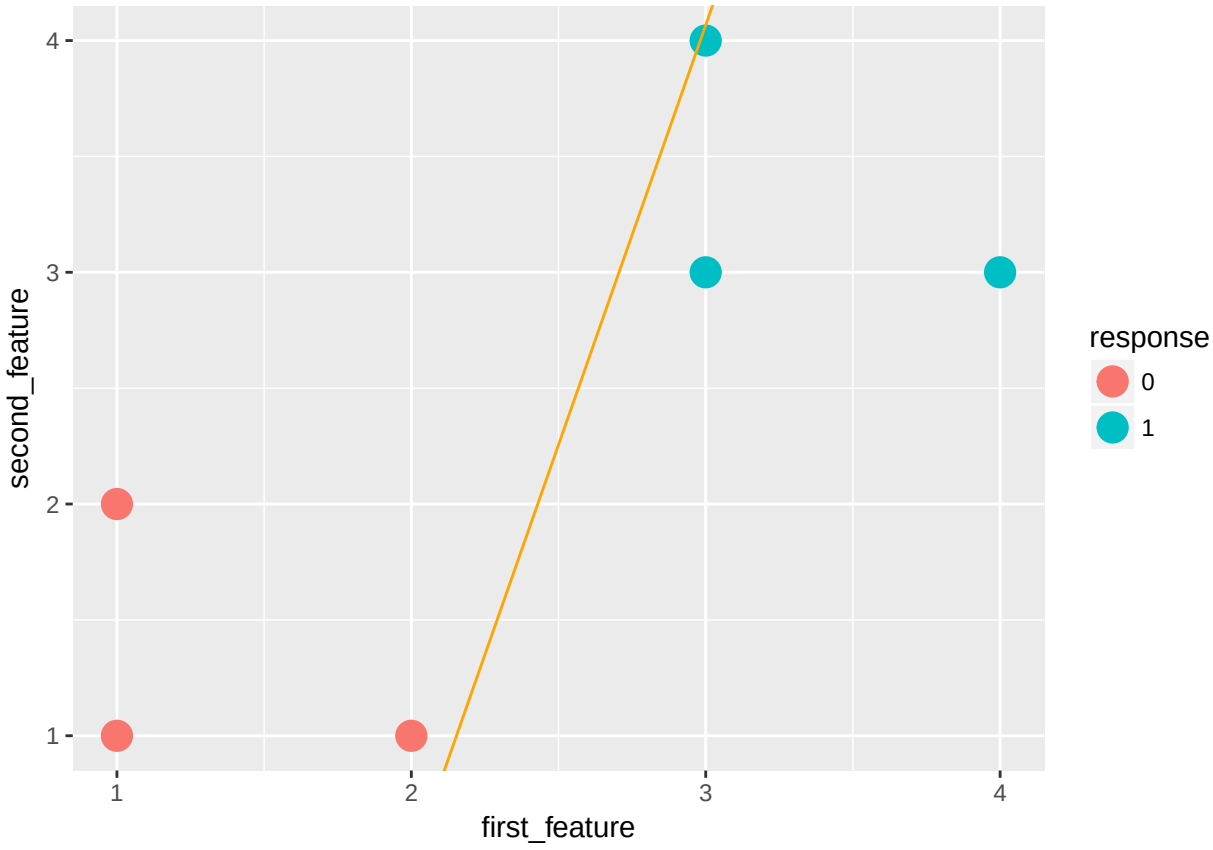
Run the code on the simple dataset above via:

```
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] -3.8643930  2.0585505 -0.5683002
```

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 it is too "close" to the "1" responses. It is not equidistant from each type of the response values. Also, the line is quite steep and if it was rotated a tad bit more to the right, the line

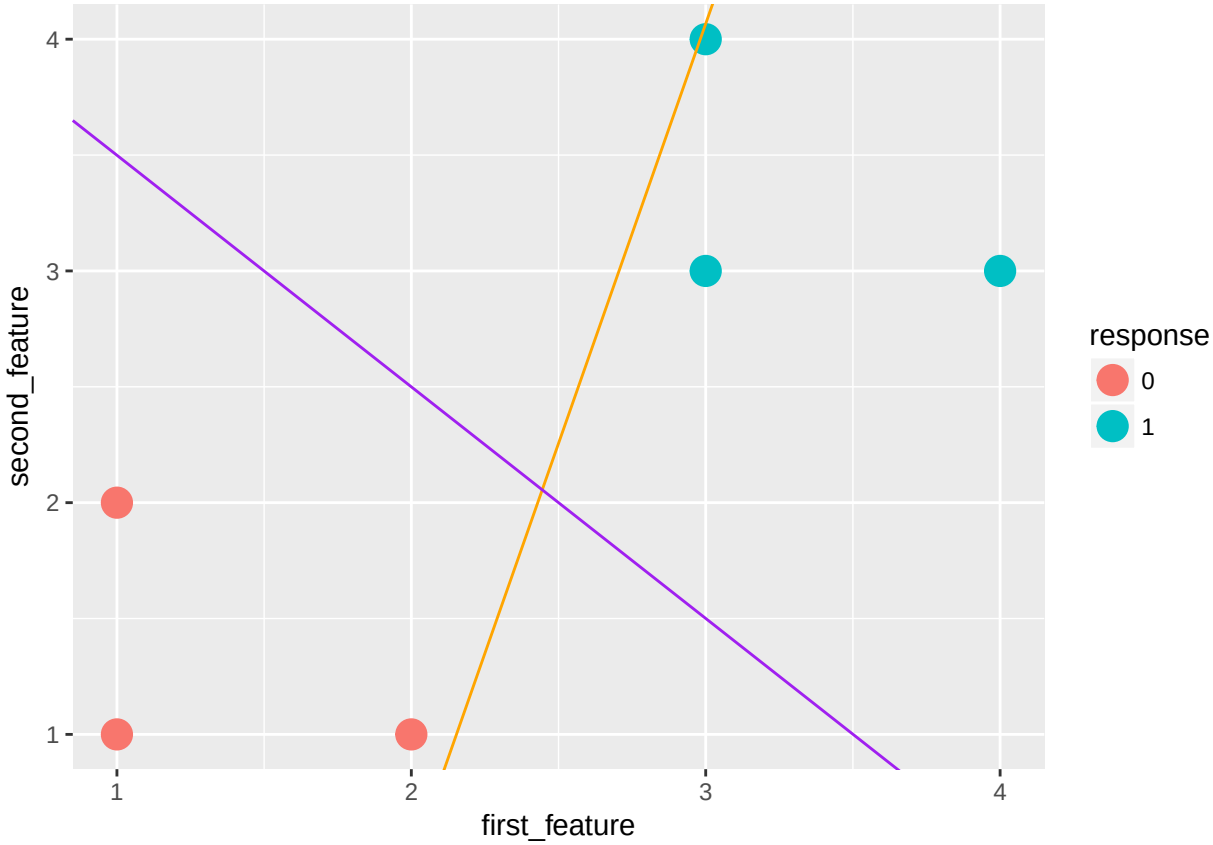
could potentially miss one of the "1" responses.

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?

Absolutely. Compared to the perceptron, the SVM line is a much better fit as it is more "equidistant" between the two types of response values.

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.

```
#' 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){
  #we need an optimal starting position that can "lead us" to the best wedge
  #at that point we want to loop over some value that checks various albeit many
  #types of lines(hyperplanes) and keeps going   until it reaches the line that contains
  #the support vectors on it
  #this should be done twice because it has to go in two directions assuming linearly
  #separable data
  #after, there needs to be a way to find the distance between these two "best" lines
  #and find its midpoint
  #by the looks of our lambda, we are more concerned with the number of errors we make
  #so our wedge may not be   of the largest distance/ not be the best hyperplane because
  #we are trying to minimize errors

}
```

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

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

```
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:

```
#' 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){
  sqd_distance = Inf
```

```
  i_star = NA
  for(i in 1 : nrow(Xinput)){
    e_distance = sqrt(sum((Xinput[i,] - Xtest[i,])^2))
    if(e_distance < sqd_distance){
      sqd_distance = e_distance
      i_star = sqd_distance
    }
  }
  y_binary[i]
}
```

Write a few tests to ensure it actually works:

```
#TO-DO
```

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.

```
#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.

```
#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:

```
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.

```
x_bar = mean(x)
y_bar = mean(y)
s_x = sqrt(sum((x - x_bar)^2) / (length(x) - 1))
s_y = sqrt(sum((y - y_bar)^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)
```

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

```
lm_mod = lm(y ~ x)   #this isnt all working
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`.

```
pacman::p_load(HistData)
```

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

```
data(Galton)
```

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`.

```
summary(Galton)
```

```
##      parent          child
##  Min.   :64.00   Min.   :61.70
##  1st Qu.:67.50   1st Qu.:66.20
##  Median :68.50   Median :68.20
##  Mean   :68.31   Mean   :68.09
##  3rd Qu.:69.50   3rd Qu.:70.20
##  Max.   :73.00   Max.   :73.70
```

This data frame contains an n = 928 and p = 1. The column "parent" represents the average height of the mid-parent. Each observation is the average of the mother's and father's height in the pair. The column "child" represents the height of the child. After looking at the summary, we notice that the mean height of the parent is 68.31 inches and the mean height of the child is 68.09 inches. These values are very similar. Some interesting information regarding the data set was that the data was recorded in non-integer values because there was a strong bias towards the integral units (inches). In addition, the female heights were multiplied by a factor of 1.08 to make up for the differences between the sexes.

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

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

```
## [1] 68.19833
```

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.

```
height_model = lm(child ~ parent, Galton)
betas = coef(height_model)
r_squared = summary(height_model)$r.squared
rmse = summary(height_model)$sigma

betas
```

```
## (Intercept)      parent
##  23.9415302   0.6462906
```

```
r_squared
```

```
## [1] 0.2104629
```

```
rmse
```

```
## [1] 2.238547
```

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

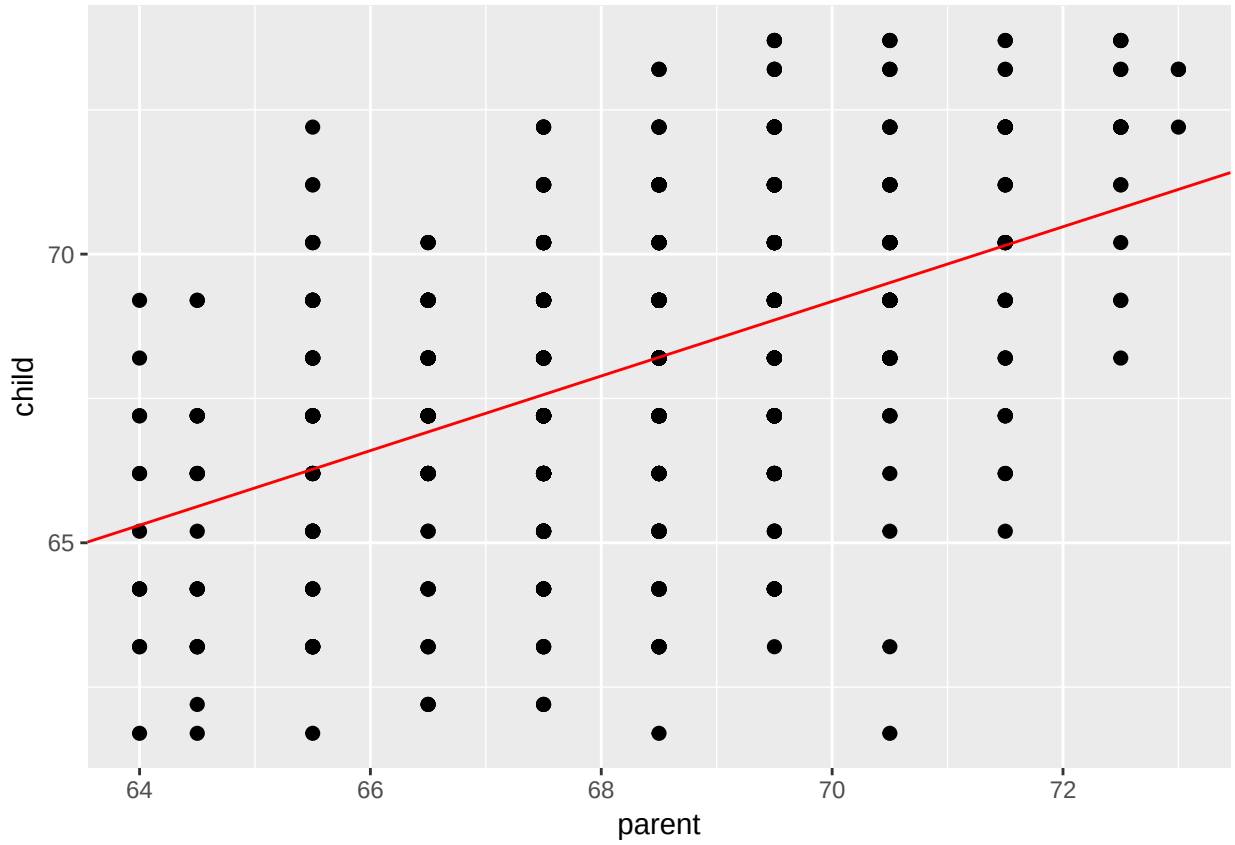
The b_0 value = 23.942 inches. This means that, if a parent had a height of 0 inches, the predicted height for the child would be 23.942 inches. The b_1 value = 0.646 inches. This means that, as the parents' height increases by 1 inch, the predicted height for the child increases by 0.646 inches. RMSE here is 2.239, which says that we can only predict within 4.48 inches 95% of the time. R^2 equals 21.046%, which suggests that 21% of the variation in a child's average height is explained by the parents' average height.

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

This model is not particularly good. An R^2 value of 21% is quite low which suggests that we would need more and accurate characteristics to capture predicting a child's height. As a result, it does not predict well. On average, it will predict 1 out of 5 potential heights accurately. In addition, our RMSE value is a bit larger than we would hope for it to be. Being able to predict within 4.5 inches 95% of the time is not accurate enough for this context. Ideally, we would like to be measuring 95% confidence within an inch or so.

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.

```
pacman::p_load(ggplot2)
height_obj = ggplot(Galton, aes(parent, child)) + geom_point(size = 2)
height_ls_regression_line = geom_abline(intercept = betas[1], slope = betas[2], color = "red")
height_obj + height_ls_regression_line
```



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

This is reasonable because we inherit genes from both of our parents and one of the observable characteristics we inherit is height. There are both dominant and recessive alleles for this trait, which directly contribute

to how tall or short a child will grow to be. This is considered "nature's" doing. However, there are other factors like nutrition and quality of life that are "nurture's" doing which can also affect height.

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?

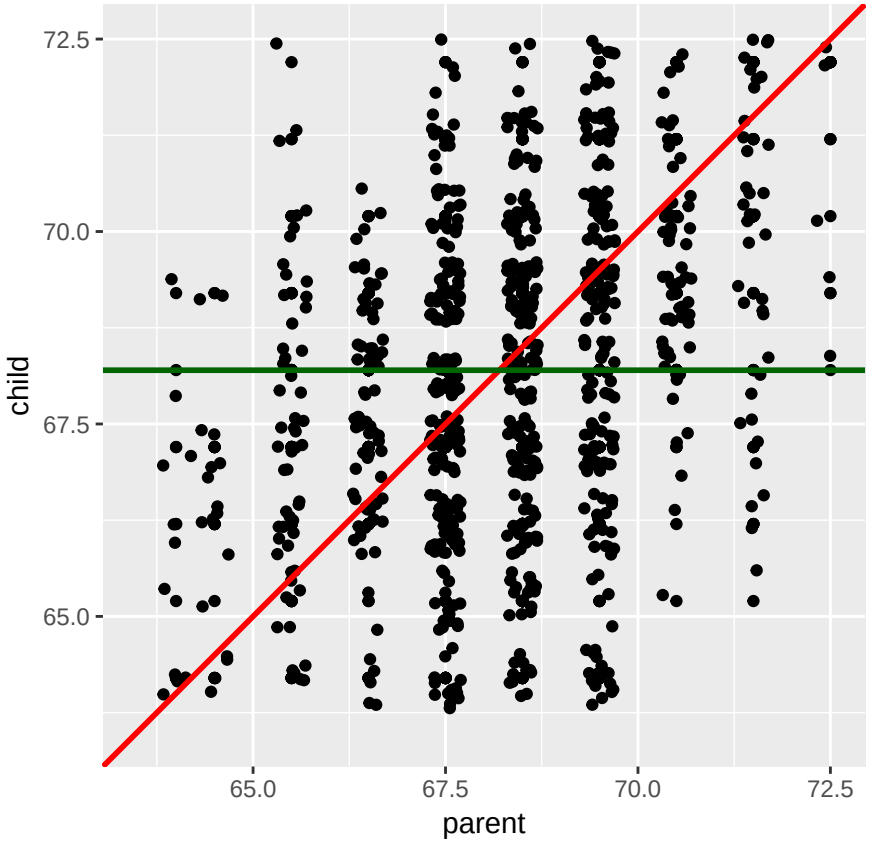
If they have the same height, that would mean the input equals the output. The value of β_0 would equal 0 and the value of β_1 would equal 1.

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.

```
ggplot(Galton, aes(x = parent, y = child)) +
  geom_point() +
  geom_jitter() +
  geom_abline(intercept = b_0, slope = b_1, 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)
```

```
## Warning: Removed 76 rows containing missing values (geom_point).
```

```
## Warning: Removed 87 rows containing missing values (geom_point).
```



Fill in the following sentence:

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"?

Galton called it this because he realized that extreme characterstics are not completely passed from parents onto their children. Instead, the characteristics in the children regress (move backwards/approach) the mediocre point(i.e. mean value). Simply put, a child who has parents that lie in the extremes of the height distribution (i.e. extremely tall or extremely short) will have heights that move closer to the center of the distribution, also known as the mean.

Why should this effect be real?

This would be real because we inherit our genetic makeup from our parents which dictates many of our characteristics.

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.

Everyone calls it a "regression" because the model allows us to predict the behavior of future events. When building predictive models, it would be more appropriate to call them approximations or estimations.