HW02p

Bryan Lliguicota

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` libray.

```
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( n in 1:MAX_ITER){
    for(i in 1:nrow(Xinput)){
        x <- Xinput[i,]
        y_i<-ifelse((w %*% x) >0 , 1 ,0)
        w<- w + as.numeric(y_binary[i]-y_i)*x
```

```
    }
  }
  w
}
```

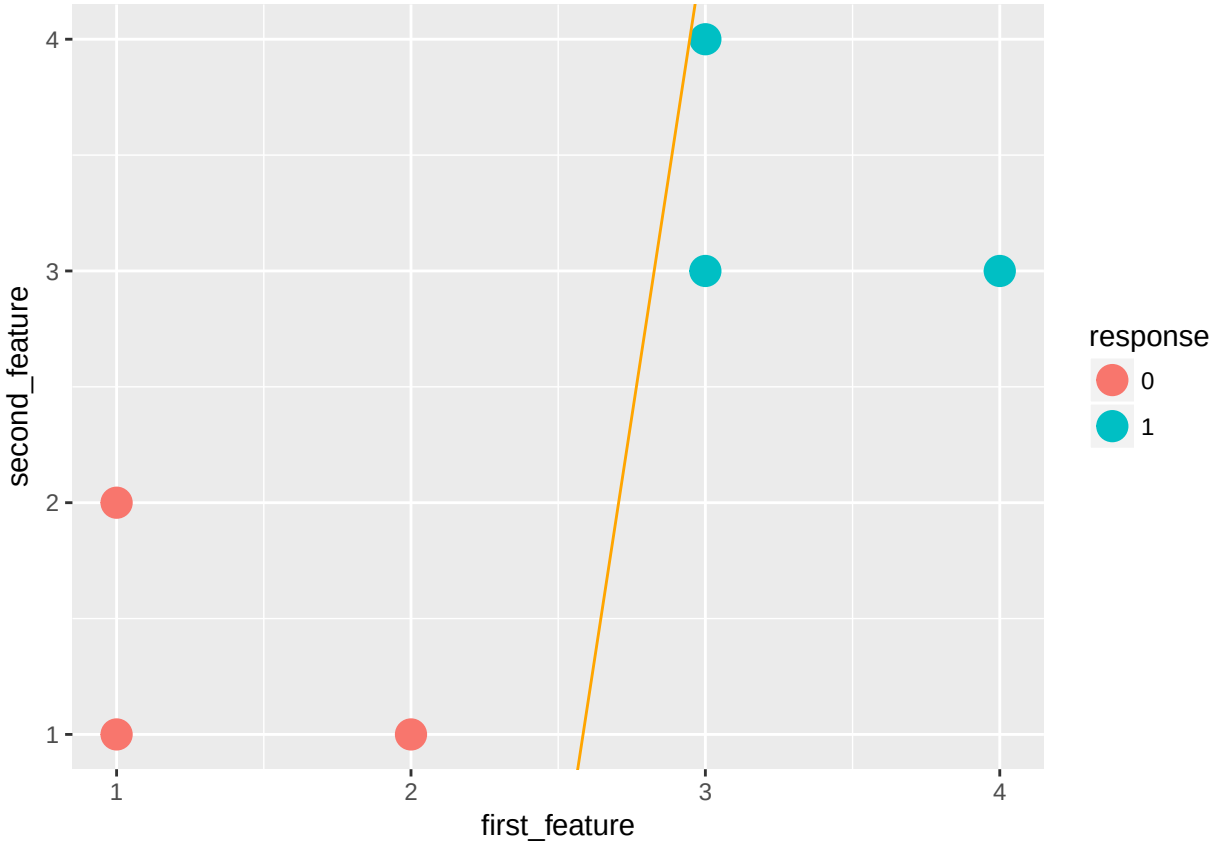
Run the code on the simple dataset above via:

```
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] -2.6604686  1.0798509 -0.1304138
```

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
```



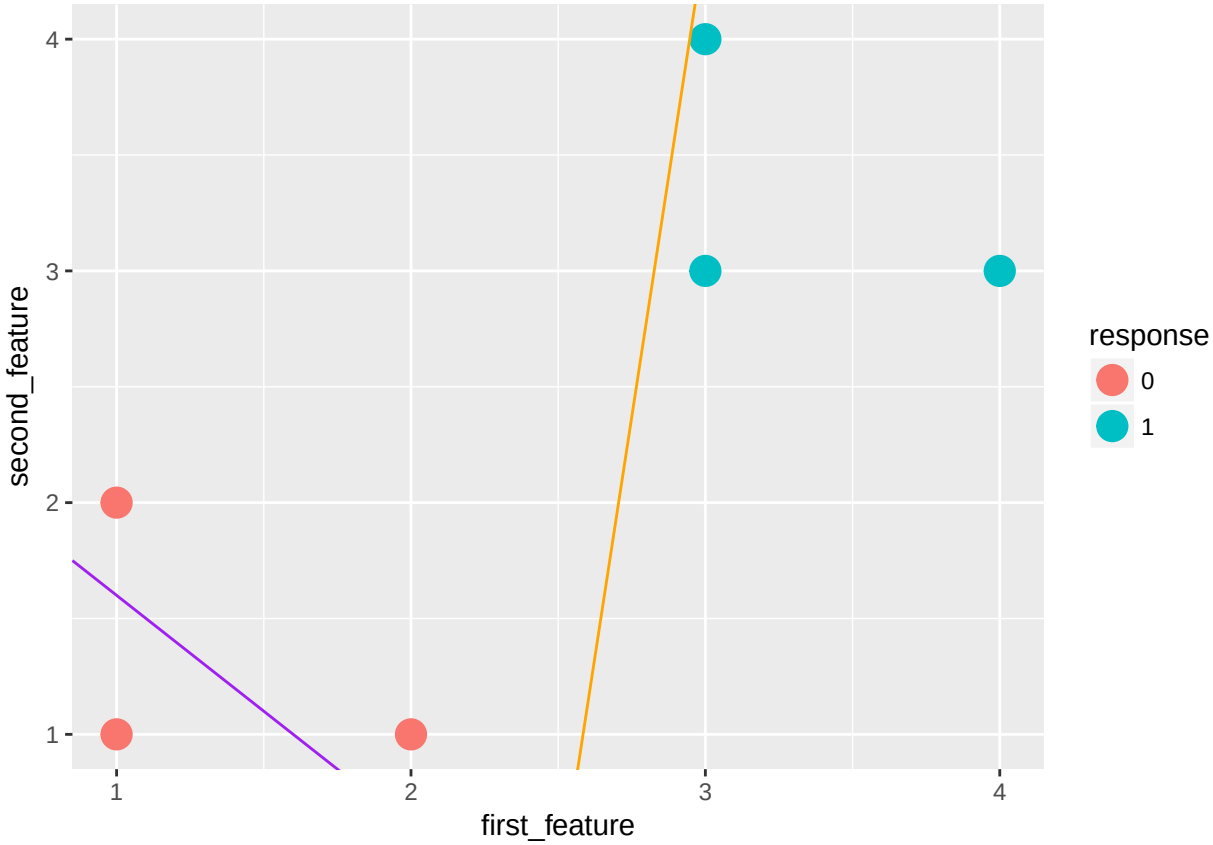
Why is this line of separation not "satisfying" to you? There are many lines that can satisfy the perceptron model, the line choosen is not necessarily the best line. I better line would be one that is equidistant from both the the 1's and 0's.

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)
?svm
svm_model = svm(X_simple_feature_matrix,y_binary,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? No

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

```
rm(list=ls())
dist_form=function(x,y){
  s<-sum((x-y)**2)
  sqrt(s)
}
```

```
#' 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)
  output_vec<-rep(Inf,n)
  correct_y_index<-0
for(k in 1:n){
    for(i in 1:n){
    #compare the distance b/w Xtest and Xinput[1,]
    tmp<- dist_form(Xinput[i,],Xtest[k,])
    if(tmp < output_vec[k]){
      output_vec[k]=tmp
      correct_y_index<-i
    }
    }
  output_vec[k]=y_binary[correct_y_index]
}
  output_vec
}
```

Write a few tests to ensure it actually works:

```
Xy = na.omit(MASS::biopsy) #The "breast cancer" data with all observations with missing values dropped
X = Xy[, 2 : 10] #V1, V2, ..., V9
#choosing the first 60 rows, else it will take to long
y_binary = as.numeric(Xy$class == "malignant")
X<-X[c(-61:-683),]
y_binary = y_binary[1:60]
y_hat = nn_algorithm_predict(X,y_binary,X)
all.equal(y_hat, y_binary)
```

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

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:

```
rm(list=ls())
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.

```
something_bar= function(x){
  n= length(x)
  sum(x)/n
}

sd_variable=function(x){
  n= length(x)
  pt1_num<-sum(x**2)
  pt2_num<-n*(something_bar(x))**2
  num<- pt1_num - pt2_num
  den<- n-1
  sqrt(num/den)
}

my_cov_like_func=function(x,y){
  #assumption: both  x and y have the same number of elements
  n<-length(x)
  pt1_num<-sum(x*y)
  pt2_num<-n*something_bar(x)*something_bar(y)
  num<-pt1_num-pt2_num
  den<-(n-1)
  num/den
}
r<- my_cov_like_func(x,y)/(sd_variable(x)*sd_variable(y))
b_1<-r*(sd_variable(y)/sd_variable(x))
b_0<- something_bar(y)-(b_1 * something_bar(x))
```

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

```
simple_linear_model_formula = y ~ x
lm_mod = lm(simple_linear_model_formula)
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`.

```
rm(list=ls())
pacman::p_load(HistData)
```

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

```
Galton_data<-Galton
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_data)
```

```
##      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
```

Galton shows the height of the child and height of their parent, I think its comparing the childs height to the height of either the father or mother. It could have also taken the mean of both heights. Here n would represent the number of data points we have in our $\mathcal{D}$ (the number of rows). p would represent what dimension we are working in, it seems we will be creating a model with only one attribute, which is eihter the parents height/child height and one $\hat{y}_i$. Find the average height (include both parents and children in this computation).

```
p_h<- sum(Galton_data$parent)
c_h<-sum(Galton_data$child)
avg_height = ((p_h)+(c_h))/(nrow(Galton_data)*2)
```

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.

```
x<-Galton_data$parent
y<-Galton_data$child
r = cor(x,y)
s_x = sd(x)
s_y= sd(y)

b_1<- r * (s_y/s_x)
b_0<- mean(y) - (b_1*(mean(x)))
simple_linear_model_formula = y ~ x
simple_linear_model = lm(simple_linear_model_formula)

Galton_data$gs<- b_0 + b_1 * x
Galton_data$residual <- y - Galton_data$gs

n<-nrow(Galton_data)
sse<- sum(Galton_data$residual**2)
mse<- sse/(n-2)
rmse<- sqrt(mse)
sse
```

```
## [1] 4640.273
```

```
mse
```

```
## [1] 5.011094
```

```
rmse
```

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

```
s_sq_y = var(Galton_data$child)
s_sq_e = var(Galton_data$residual)
rsq = (s_sq_y - s_sq_e) / s_sq_y
rsq
```

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

```
names(summary(simple_linear_model))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
b = coef(simple_linear_model)
b
```

```
## (Intercept)           x
##  23.9415302   0.6462906
```

```
summary(simple_linear_model)$r.squared #the R^2
```

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

```
summary(simple_linear_model)$sigma #the RMSE
```

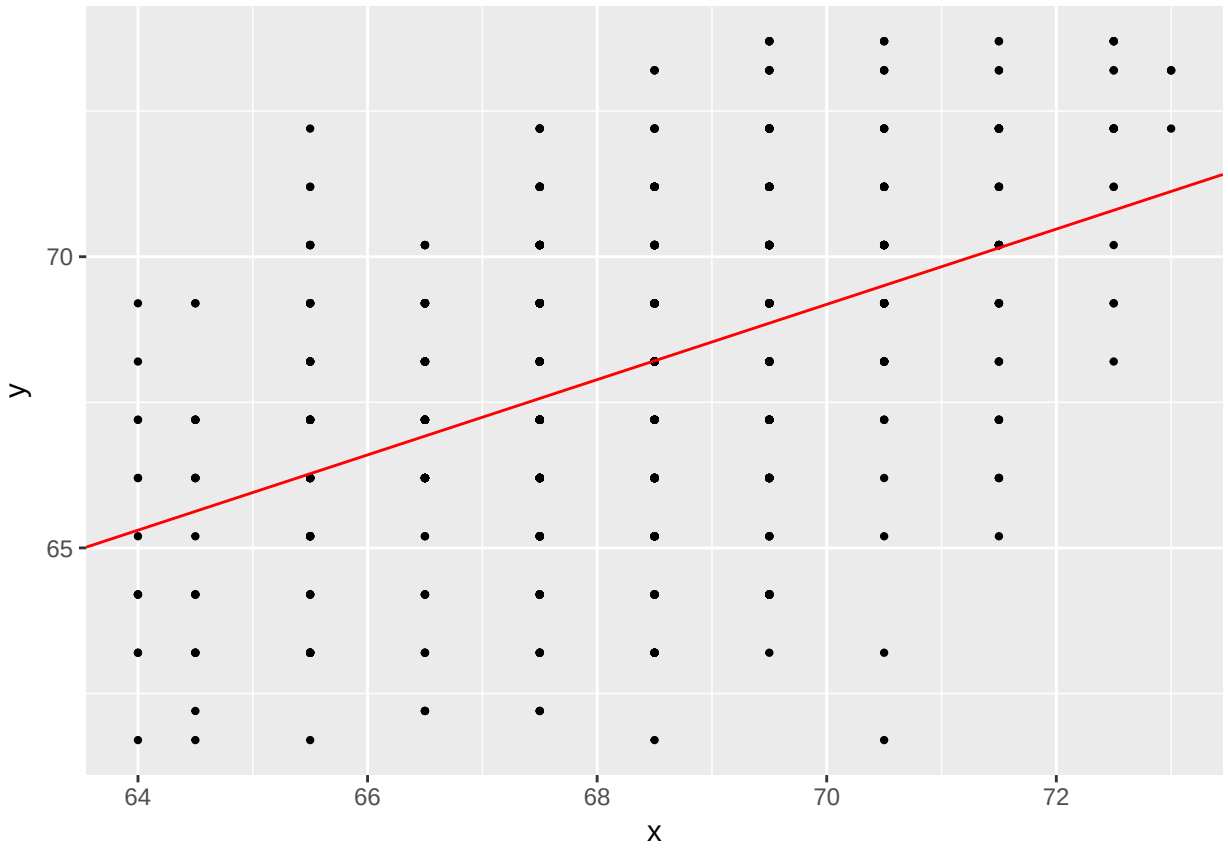
```
## [1] 2.238547
```

Interpret all four quantities: b_0, b_1, RMSE and R^2. b_0 is the x intercept and b_1 is would be the slope. R^2 is the propability of variance explained. The R^2 is the sse of the null model minus the sse of yor linear model over the sse of the of the null model. It ranges from 0 to 1 and tells you how well your model fits the data with relation to the null model. RMSE is the sqrt of the mean square error, this measurment is not unitless and gives you a measurment of your error, it does not compare the error to the null models error.

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)

simple_viz_obj = ggplot(Galton_data, aes(x, y)) +
  geom_point(size = .8)
simple_ls_regression_line = geom_abline(intercept = b_0, slope = b_1, color = "red")
simple_viz_obj + simple_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.

-Its linked to genetics, you will inherit traits from both parent, just like parents inherits therer physical traits from their parent. Their is a good chance that the childs height would be similar to their parents height.

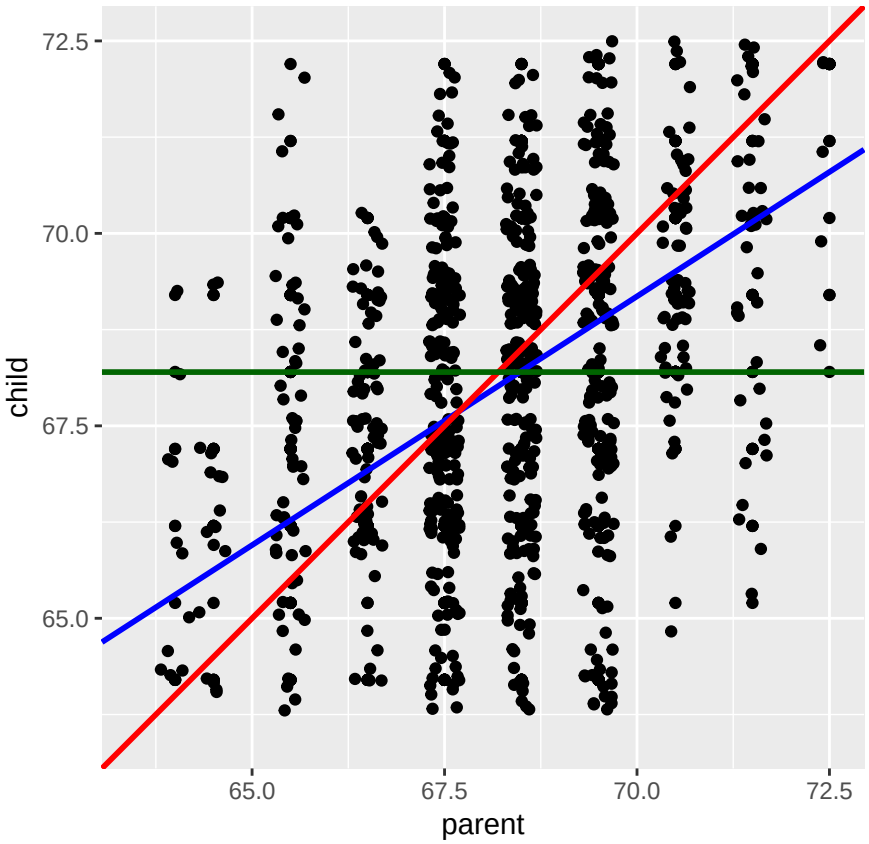
If they were to have the same height and any differences were just random noise with expectation 0, what would the values of β_0 and β_1 be? We would have a β_1 of 0 (a slope of 0) and β_0 would be the $\bar{y}$

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 89 rows containing missing values (geom_point).
```



Fill in the following sentence:

TO-DO: Children of short parents grow to be taller than their parents on average and children of tall parents became shorter than their parents on average.

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

For a given data set $\mathcal{D}$, if the attributes are iid you will notice that the majority of realizations occuring around the mean. If you subset the dataset to only the ones who where realized to be above the 75 percentile and perform a realization, you wil notice once again the bulk of your realizations being around the mean.

Why should this effect be real? -Relating this to the normal curve, with $\bar{y}$ on the x-axis being 'mu', the majority of area under the curve is between ± 2 standard divations from 'mu'. Everyone tends to be average, few are above and below 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.

Predicating with the mean.