HW02p

[Joseph Peltroche]

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 (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
}
```

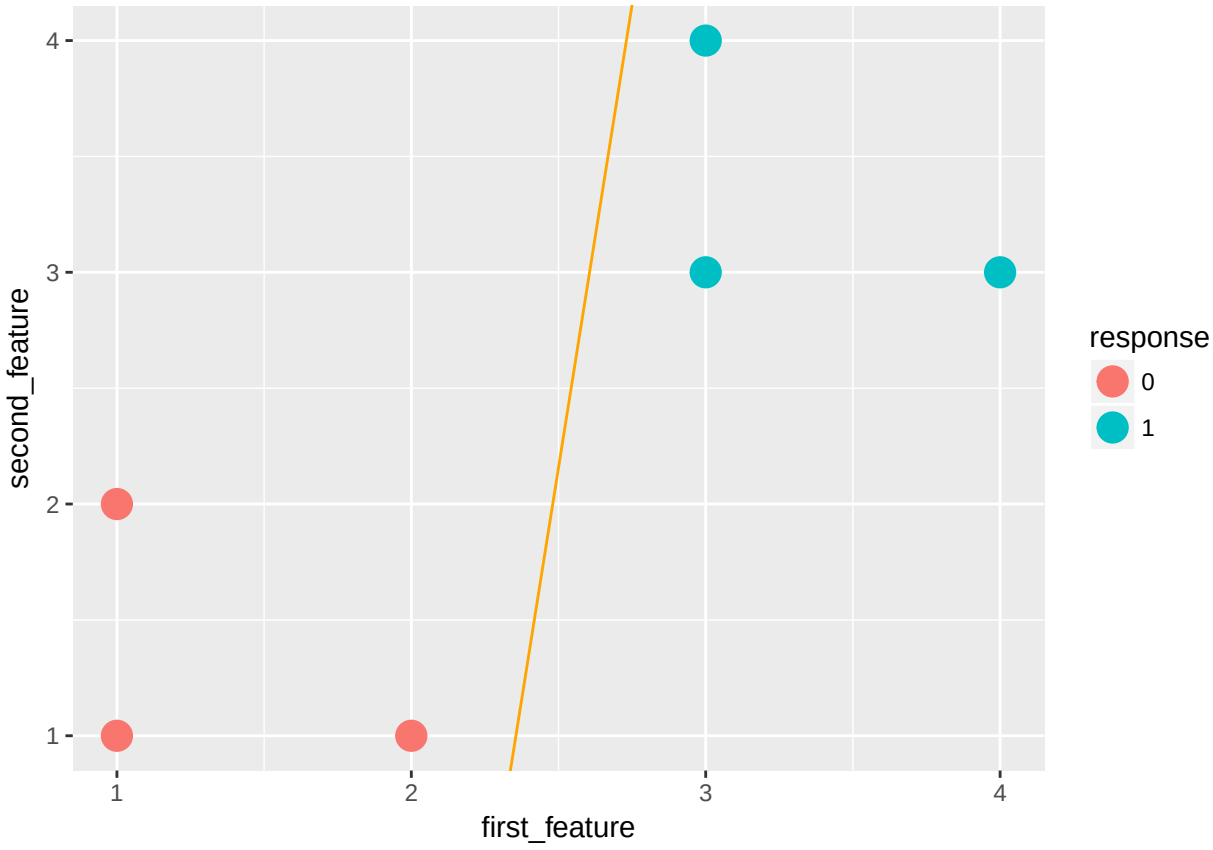
Run the code on the simple dataset above via:

```
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] -8.1501028  3.6544967 -0.4571735
```

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?

The line isn't directly in the middle of the margin; it doesn't produce the best line for this linearly separable case, but it does produce a line somewhere within the margin.

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])
lambda=1e-13
svm_model=svm(Xy_simple_feature_matrix, Xy_simple$response, kernel = "linear", cost = (2 * n * lambda)^
```

```
## Error in svm.default(Xy_simple_feature_matrix, Xy_simple$response, kernel = "linear", : object 'n' n
```

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
)
```

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

```
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")
```

```
## Error in data.frame(intercept = intercept, slope = slope): object 'w_vec_simple_svm' not found
```

```
simple_viz_obj + simple_perceptron_line + simple_svm_line
```

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

Is this SVM line a better fit than the perceptron?

The SVM is a better fit than the perceptron because it returns a line right in the middle of the margin, opposed to the perceptron that spits out any line within the margin.

3. Now write pseuocode 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.

```
#' 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 0.1 to mimic hard margin in the linearly separable case.
#' @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){
#  cost_func = function(w_vec){
#  h_i=ifelse(0.5-(w_vec%*%x-b)*(y_binary-0.5)>0,0.5-(w_vec%*%x-b)*(y_binary-0.5),0)
#  min_a=sum(h_i)/length(h_i)+lambda*w_vec%*%w_vec
#  min_a
}

}
```

```
## Error: <text>:14:1: unexpected '}'
## 13: #  min_a
## 14: }
##     ^
```

To properly get a linear svm leanring algorithm, we must first take a look at the cost function derived in class. Within the function, we will establish another function that will be our cost function, incorporating the hinge loss error and the other term that maximizes the margin with an input of the w vector. Next, this function will be optimized using the optim function with a w vector intialized. The return of the optim will be the output of the whole function.

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)
```

```
## Error in linear_svm_learning_algorithm(X_simple_feature_matrix, y_binary): could not find function "l
```

```
my_svm_line = geom_abline(
    intercept = -svm_model_weights[1] / svm_model_weights[3],
    slope = -svm_model_weights[2] / svm_model_weights[3],
    color = "brown")
```

```
## Error in data.frame(intercept = intercept, slope = slope): object 'svm_model_weights' not found
```

```
simple_viz_obj + simple_svm_line + my_svm_line
```

```
## Error in eval(expr, envir, enclos): object 'simple_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 y_output   The predictions as a n* length vector.


nn_algorithm_predict = function(Xinput, y_binary, Xtest){
  n=ifelse(is.null(nrow(Xtest))==1,1,nrow(Xtest))
  #dummy=matrix(NA,nrow = nrow(Xinput),ncol = n )
  j_star=rep(NA, n )
  y_output=rep(NA,n)
  for(i in 1: n ){
    if (n < 2){
      x_star = Xtest
    }else{
    x_star=Xtest[i,]
    }
    d=Inf
    for(j in 1 : nrow(Xinput)){
      dummy=as.numeric((c(Xinput[j,])-x_star)%*%(c(Xinput[j,])-x_star))
      if (dummy < d){
        d=dummy
        j_star[i]=j
      }
```

```
    }
  }
  for(k in 1: n){
    y_output[k]=y_binary[j_star[k]]
  }
  y_output
}
```

Write a few tests to ensure it actually works:

```
pacman::p_load(class)

Xy = na.omit(MASS::biopsy)
#X = matrix(Xy[, 2 : 3], nrow= nrow(Xy),ncol = 2)
X=as.matrix(Xy[,2:3])
y_binary = as.numeric(Xy$class == "malignant")
Xtest=c(4,2)
expect_equal(as.factor(nn_algorithm_predict(X,y_binary,c(4,2))), factor(knn(X, c(4, 2), y_binary, k = 1
expect_equal(as.factor(nn_algorithm_predict(X,y_binary,c(2,1))), factor(knn(X, c(2, 1), y_binary, k = 1

X = as.matrix(Xy[, 2 : 6])
expect_equal(as.factor(nn_algorithm_predict(X,y_binary,c(1,2,3,2,4))), factor(knn(X, c(1,2,3,2,4), y_bi
```

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.

```
b_1=as.numeric((x%*%y-n*mean(x)*mean(y))/(x%*%x-n*mean(x)^2))
b_0=as.numeric(mean(y)-b_1*mean(x))
```

Verify your computations are correct using the `lm` function in 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`.

```
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
```

Galton is a data frame named after it's presenter that displays the heights of 928 adult children and the mid-parents heights. The two columns are expressed as the mid-parent's height and the adult child's height; the data measured is in class intervals of 1.0 in. All heights of female children were multiplied by 1.08 to compensate for sex differences. Here the sample size n=928 and p=2, because there are only two dimensions to each of our features. The Tukey 5 number summaries for each column are relatively close to each other, with the minimum of both parent and child having the greatest difference.

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

```
avg_height = mean(as.vector(cbind(Galton$parent, Galton$child)))
```

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.

```
Galton_linear_model=lm(Galton$child ~ Galton$parent)
b=coef(Galton_linear_model)
b_0=b[1]
b_1=b[2]
summary(Galton_linear_model)
```

```
##
## Call:
## lm(formula = Galton$child ~ Galton$parent)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.8050 -1.3661  0.0487  1.6339  5.9264
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)   23.94153    2.81088   8.517   <2e-16 ***
## Galton$parent  0.64629    0.04114  15.711   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.239 on 926 degrees of freedom
## Multiple R-squared:  0.2105, Adjusted R-squared:  0.2096
## F-statistic: 246.8 on 1 and 926 DF,  p-value: < 2.2e-16
```

```
names(summary(Galton_linear_model))
```

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

```
R_2=summary(Galton_linear_model)$r.squared
RMSE=summary(Galton_linear_model)$sigma
b_0
```

```
## (Intercept)
##    23.94153
```

```
b_1
```

```
## Galton$parent
##     0.6462906
```

```
R_2
```

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

```
RMSE
```

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

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

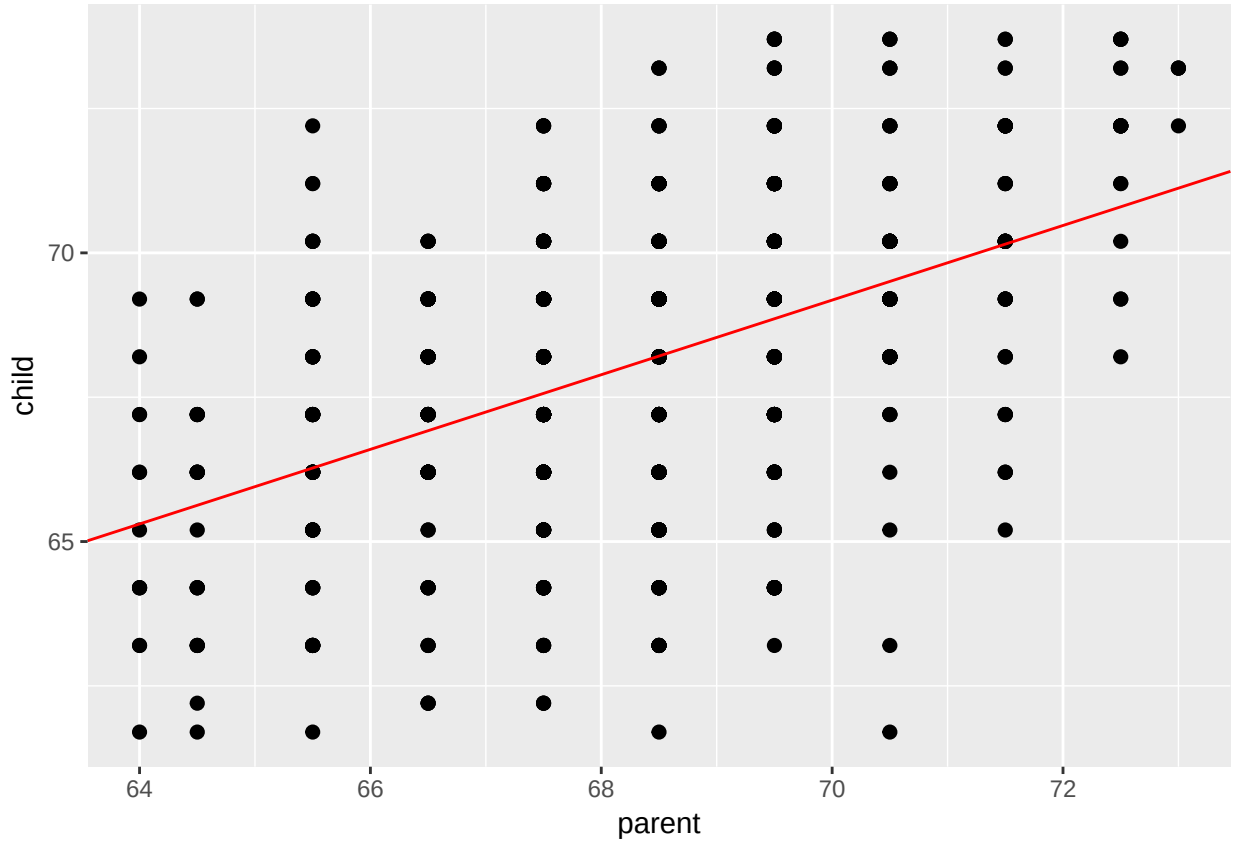
b_0 is the intercept of the linear model, b_1 is the slope of our linear model. The root mean squared error is indicates the average difference between the actual adult child's height and the predicted adult child's height, which differs by roughly 2 inches. Finally, R^2 is the difference of the sample variance to that of the null model, which in this case is 0.2, indicating the sample variance of errors is considerably big and close to that of the null model.

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

Based off this model, the RMSE is considerably big and R^2 is considerably low. Both indicate that its predictions have huge error and the errors have a huge sample variance, implying it is not a good model.

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)
Galton_obj= ggplot(Galton, aes(x=parent,y=child)) + geom_point(size=2)
best_fit_line=geom_abline(intercept=b_0,slope = b_1,color="red")
Galton_obj+best_fit_line
```



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

It is reasonable to assume the parents and their children have the same height since the child has received roughly even mixtures of DNA from both of his/her parents.

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?

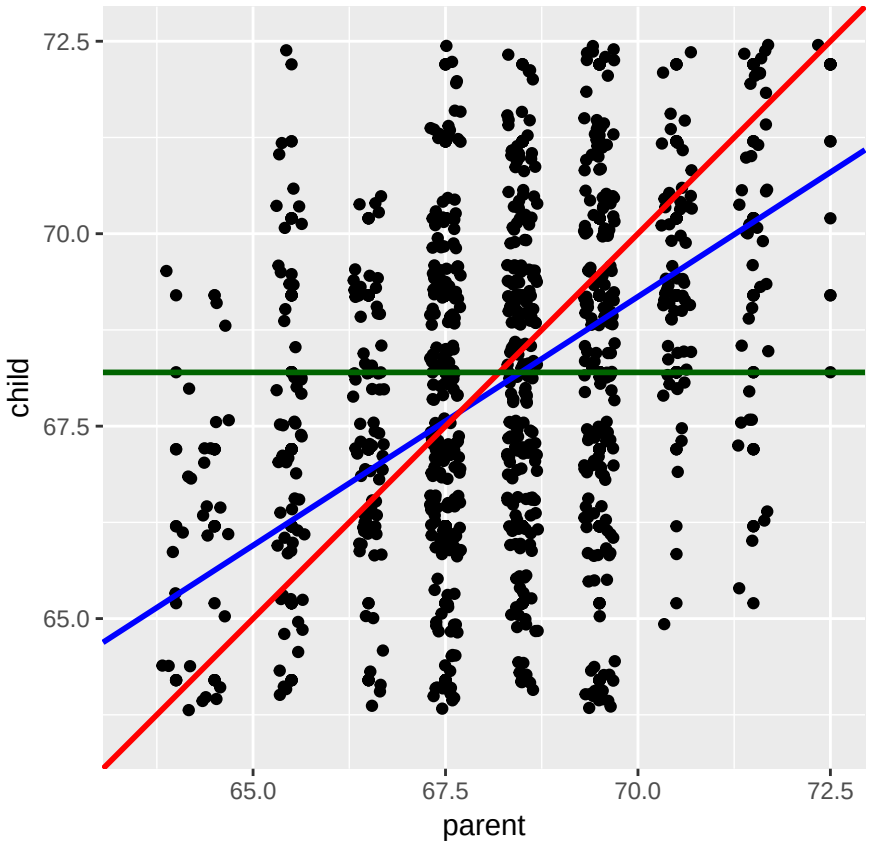
If they were to have the same height and the differences were really just random noise, then there would be a y=x realtionship, hence the intercept, b_0=0, and the slope b_1 would be equal to 1.

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



Fill in the following sentence:

TO-DO: Children of short parents became 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"?

Galton called it "Regression towards mediocrity in hereditary stature" because the data indicates the theoretical linear relationship is approaching an average height. This idea aligns with the use of the word "mediocrity", since the line is regressing to ordinary height.

Why should this effect be real?

This effect, in regards to adult/children height, should be real since genetic makeup is passed down from parents to children but the dominant traits are likely to prevail over recesssive traits, thereby filtering out extreme characteristics with every passing generation.

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.

Galton coined the term regression when he observed the trend of data points in his data frame cluttering towards the average. In quantifying the observed trend, Galton invented linear regression analysis which has grown in popularity and is commonly used toward by modern statisticans. However, for a more descriptive and appropriate name for building predictve models with y continuous, I would name the model Optimal Linear Comformity, since we are using the information from our training data to create a linear model that conforms to the data in the most favorable way.