HW02p

Adina Bechhofer

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){
  if (is.null(w)){
    w = runif(ncol(Xinput))
  }
  for (iter in 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
}
```

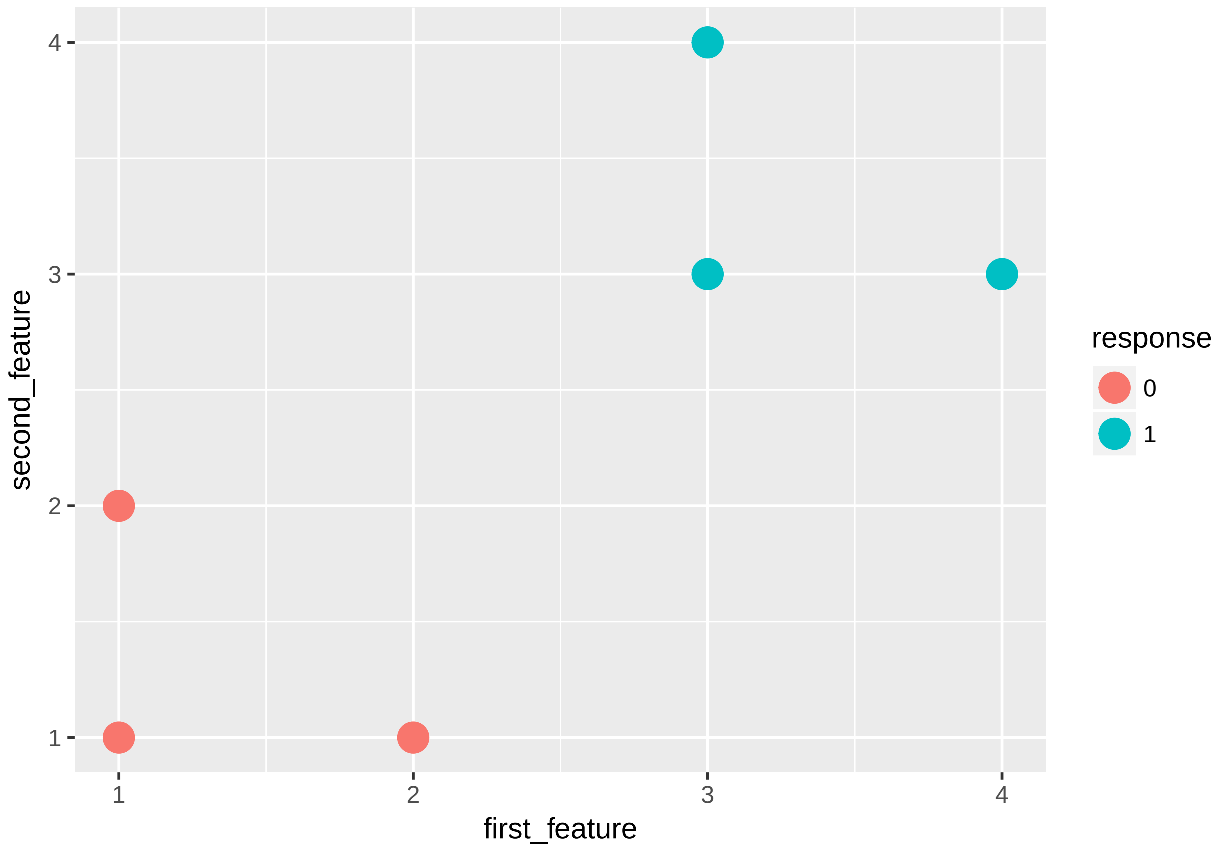
Run the code on the simple dataset above via:

```
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] 0.3625643 2.0083509 2.2473114
```

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 = "red")
simple_viz_obj + simple_perceptron_line
```



Why is this line of separation not "satisfying" to you?

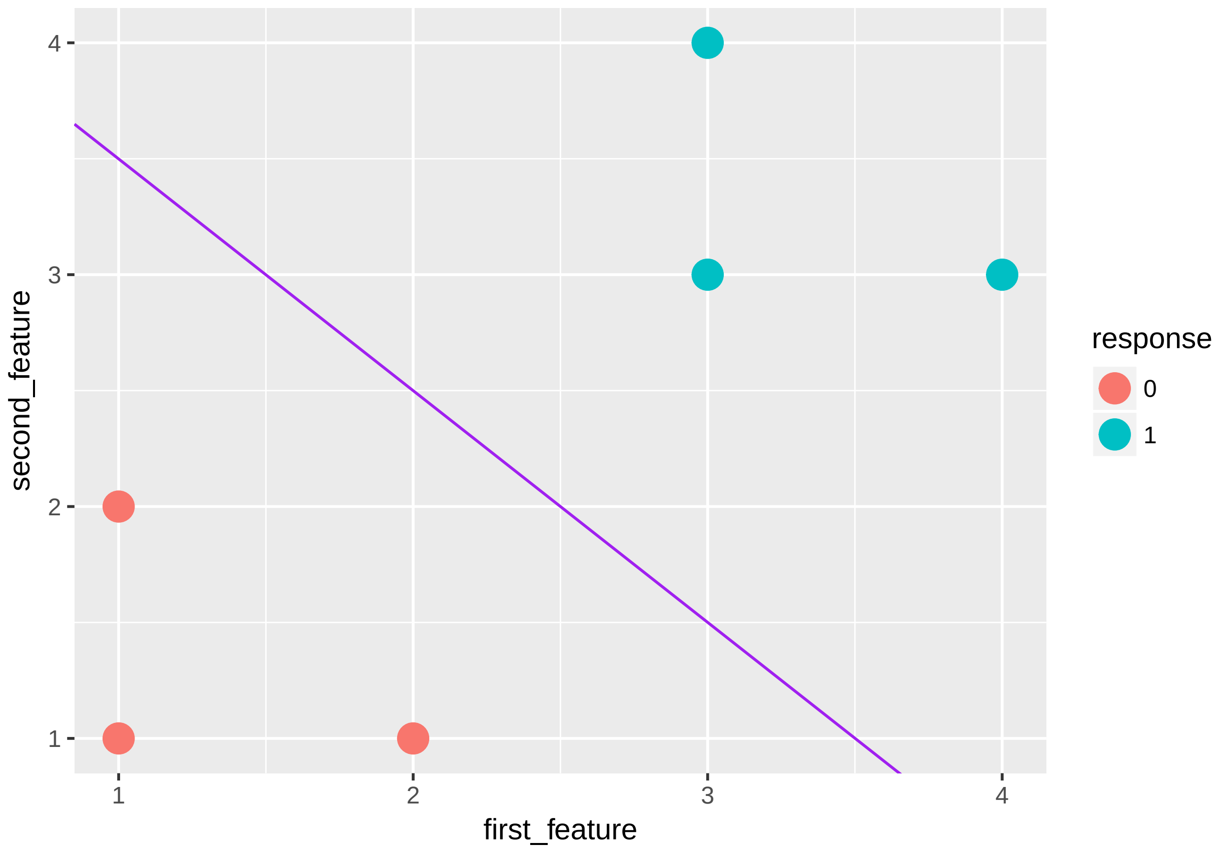
It seems unatural. I would expect a good seperation to maximize the margins.

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_model = svm(X_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?

Yes

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.

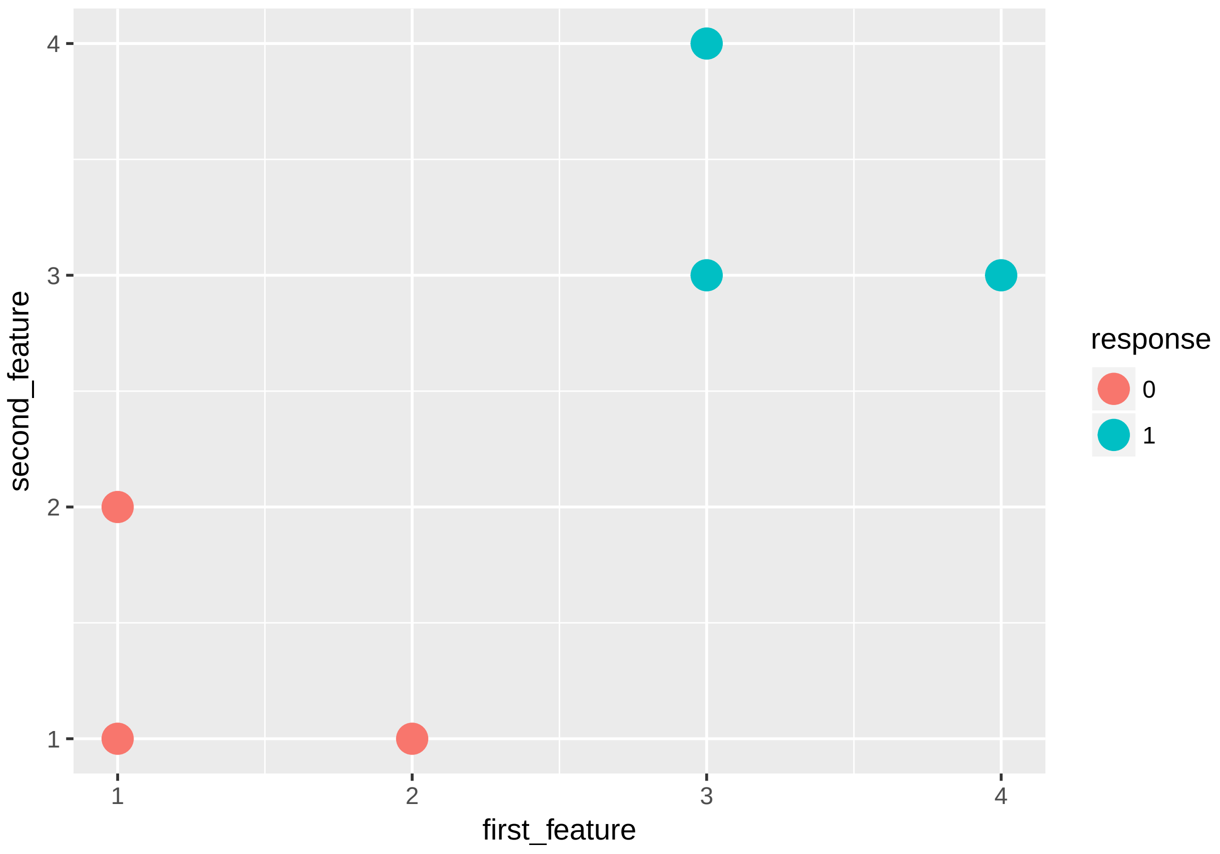
Note there are differences between this spec and the perceptron learning algorithm spec in question #1. You should figure out a way to respect the `MAX_ITER` argument value.

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){
  w = runif(ncol(Xinput)+1)

  cost = function (X, y, w, lambda){
    loss= max(0, 0.5 - (y-0.5)*(X%*%w[-1] - w[1])) + lambda*w[-1]%*%w[-1]
    loss
  }
  for (j in 1:MAX_ITER){
    for (i in 1:nrow(Xinput)){
      optim(par= w, fn= cost, X=Xinput[i, ],y = y_binary[i],lambda = lambda)
    }
  }
  w
}
```

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 = "purple")
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'
#' @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){

  Edist = function(X1, X2){
  sum = 0
  for (i in 1:length(X1)){
    sum = sum + (X1[i]-X2[i])^2
  }
  sqrt(sum)
}

  min_dist = Edist(Xinput[1,], Xtest)
  y_hat = y_binary[1]
  for (j in 1:nrow(Xinput)){
    if (Edist(Xinput[i,], Xtest)< min_dist){
      min_dist = Edist(Xinput[i,], Xtest)
      y_hat = y_binary[i]
```

```r
    }
  }
  y_hat
}
```

Write a few tests to ensure it actually works:

```r
data(iris)
Xy = iris
Xy = Xy[Xy$Species != "virginica",]
y = as.numeric(Xy$Species == "versicolor")
X= as.matrix(Xy[, c(1, 2, 3, 4)])
ytest= c()
for (i in 1:nrow(X)){
  ytest = c(ytest, nn_algorithm_predict(X, y, X[i, ]))
}

expect_equal(y, ytest)
```

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
#' 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.
#' @param k           The
#' @return            The predictions as a n* length vector.

Edist = function(X1, X2){
  sum = 0
  for (i in 1:ncol(X1)){
    sum = sum + (X1[i]-X2[i])^2
  }
  sqrt(sum)
}

nn_algorithm_predict = function(Xinput, y_binary, Xtest, k){
  k_nearset= matrix(rep(1000, 2*k), ncol= k)
  for (j in 1:nrow(Xinput)){
    if (Edist(Xinput[i,], Xtest) < max(k_nearest[1, ])){
      q =
      min_dist = Edist(Xinput[i,], Xtest)
      y_hat = y_binary[i]
    }
  }
  y_hat
}
```

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)

sum1 =0
sum2 =0

for (i in 1:n){
  sum1 = sum1 + x[i]*y[i]
  sum2 = sum2 +x[i]^2

}
b_1 = (sum1 -n*y_bar*x_bar)/(sum2 - n*x_bar^2)
b_0 = y_bar - b_1*x_bar
```

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`.

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

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

```
library(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
```

```
dim(Galton)
```

```
## [1] 928   2
```

Galton data frame has 2 columns and 928 rows. For this data, n = 928, p=1. The regressowr is the height of parent (measured in inches). The response is the height of the child (measured in inches).

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

```
n = 928
avg_height = (sum(Galton[,1]) + sum(Galton[,2]))/(2*n)
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.

```
y = Galton[,2]
x= Galton[,1]
linear_model = lm(y ~ x)
coef(linear_model)
```

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

```
summary(linear_model)$r.squared
```

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

```
summary(linear_model)$sigma
```

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

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

The intercept, b_0, is the predicted height for a child whose parent is 0 inches tall. The slope, b_1 is the height that each additional inch of the parent adds to the height of the child. R^2 is 0.21, which is far than 1. Yet, it is greater thann zero. It means than 0.2 of the varience in the child's height is explained by the parent's height. RMSE is 2.23. Which means That 95% of the prediction made will fall within 4.47 inches abov or below the predicted value.

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

Depends onthe purpose of the model. We managed to explain some of the variation in height. Considering that the average height is 68.2 inches, RMSE of 2.23 is not terrible. The predictions are useful as rough estimates.

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)

x = Galton$parent
```

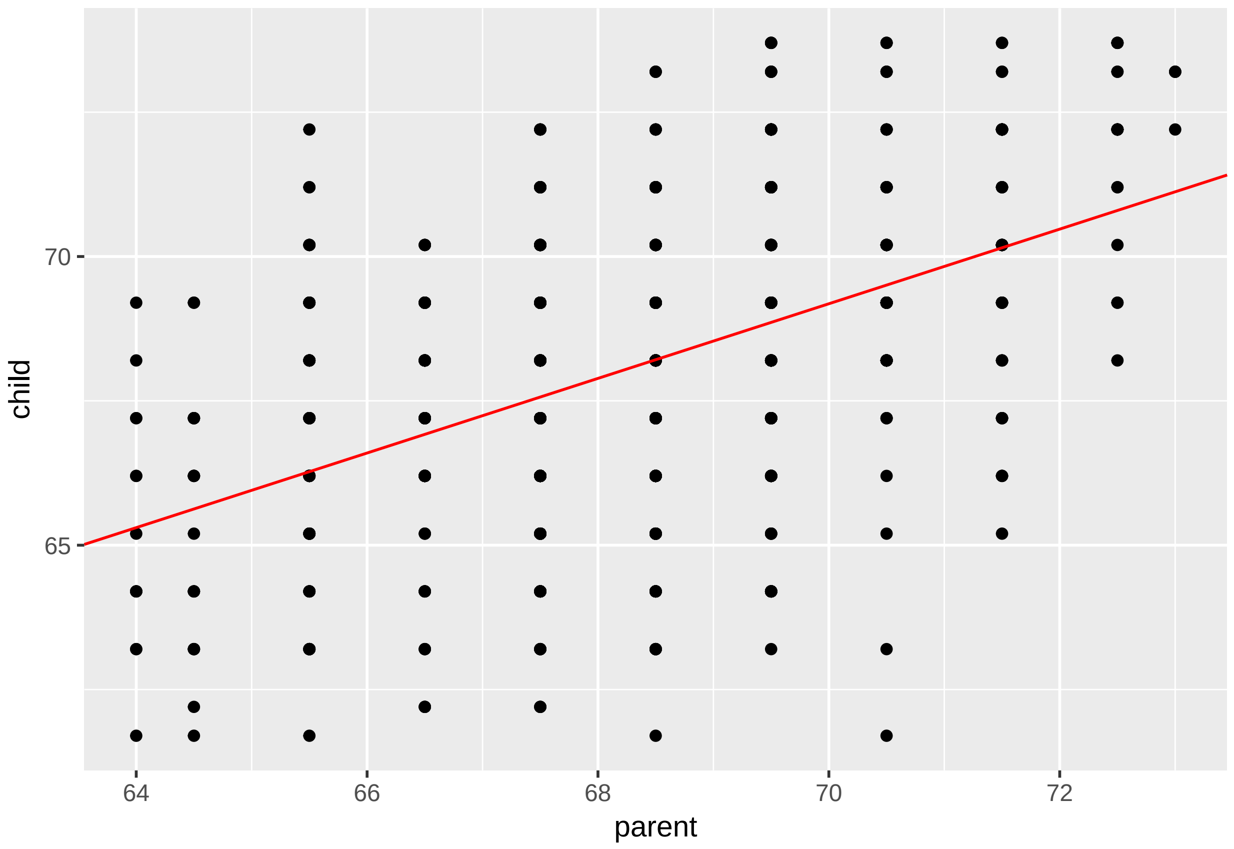
```r
y = Galton$child

r = cor(x, y)
s_x = sd(x)
s_y = sd(y)
ybar = mean(y)
xbar = mean(x)

b_1 = r * s_y / s_x
b_0 = ybar - b_1 * xbar
b_0
```

```
## [1] 23.94153
```

```r
b_1
```

```
## [1] 0.6462906
```

```r
simple_viz_obj = ggplot(Galton, aes(x = parent, y = child)) + geom_point()
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.

Height is a genetic trait. It is inherited from a person's the biologic parents. It makes sense to assume a child will havde similar genetic makeup to one of its parents.

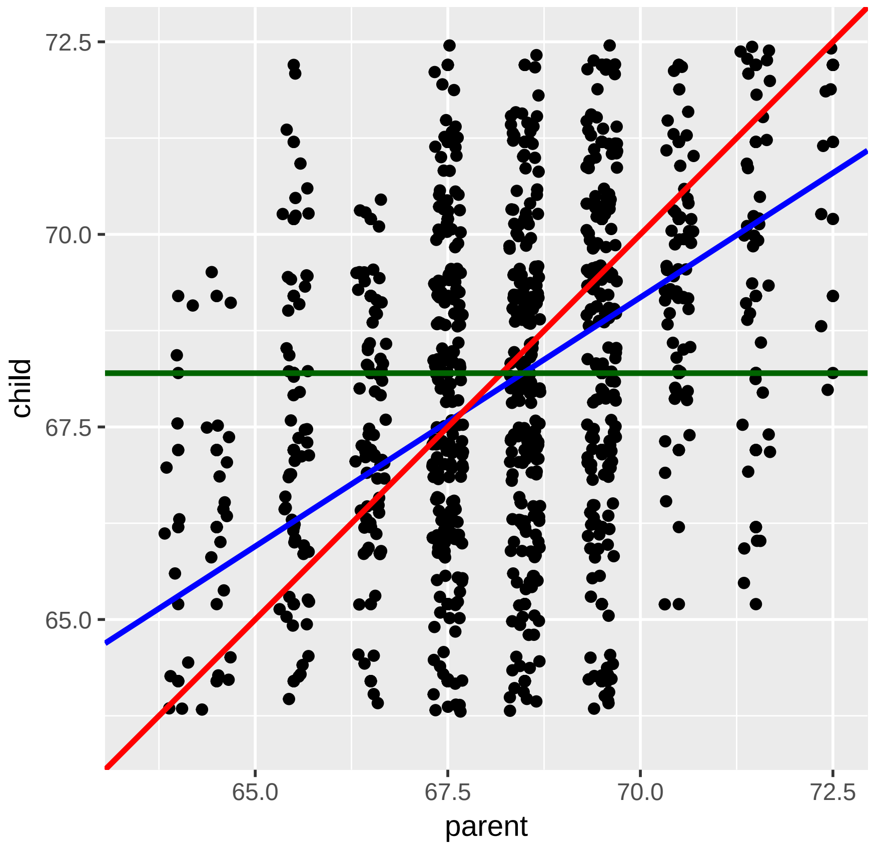
If they were to have the same height and any differences were just random noise with expectation 0, what would the values of β_0 and β_1 be?

β_0 Should be 0, and β_1 should be 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_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 88 rows containing missing values (geom_point).
```



Fill in the following sentence:

TO-DO: Children of short parents became taller then 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"?

There's a tendency towards the average value of the sample.

Why should this effect be real?

Maybe because the most common height gene is dominant.

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.

Linear relationship fitting.