

# Lab 3

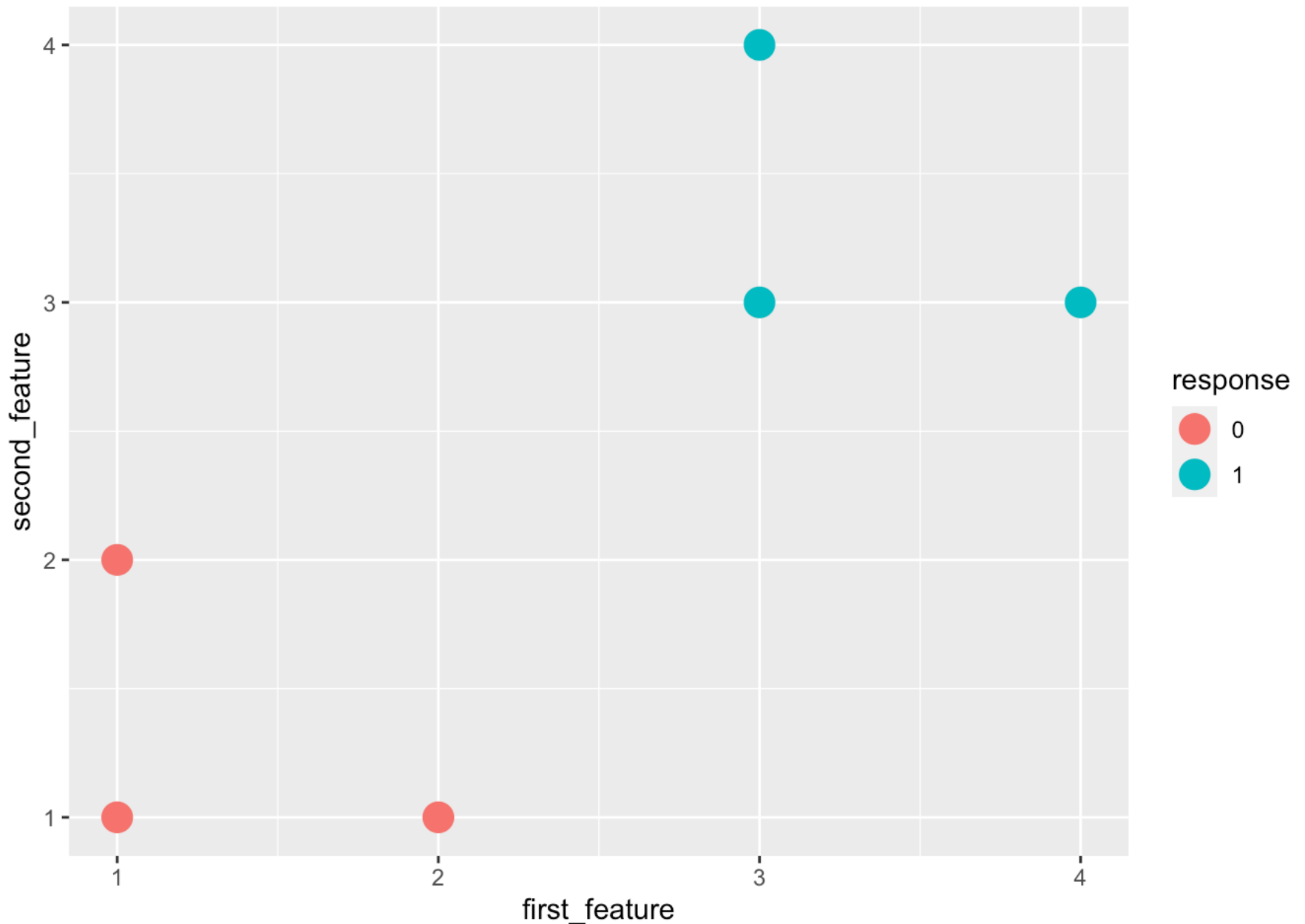
Christian Guaraca

11:59PM March 4, 2021

## Support Vector Machine vs. Perceptron

We recreate the data from the previous lab and visualize it:

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



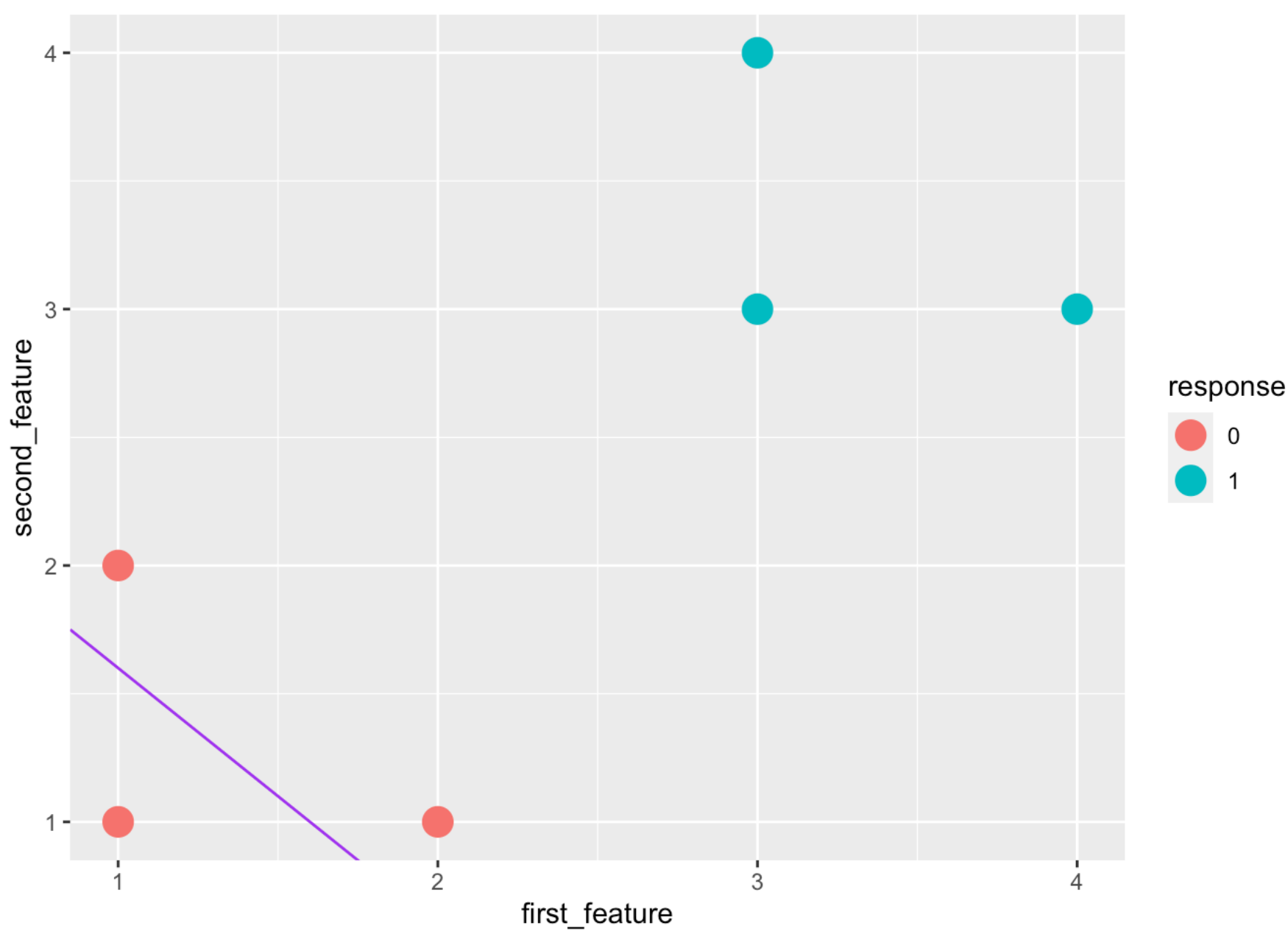
Use the `e1071` package to fit an SVM model to the simple data. Use a formula to create the model, pass in the data frame, set kernel to be `linear` for the linear SVM and don't scale the covariates. Call the model object `svm_model`. Otherwise the remaining code won't work.

```
pacman::p_load(e1071)
svm_model = svm(
  (formula = as.matrix(Xy_simple[2:3])),
  (data = as.numeric(Xy_simple$response == 1)),
  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) %*% cbind(Xy_simple$first_feature, Xy_simple$second_feature)[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_svm_line
```



Source the `perceptron_learning_algorithm` function from lab 2. Then run the following to fit the perceptron and plot its line in orange with the SVM's line:

```

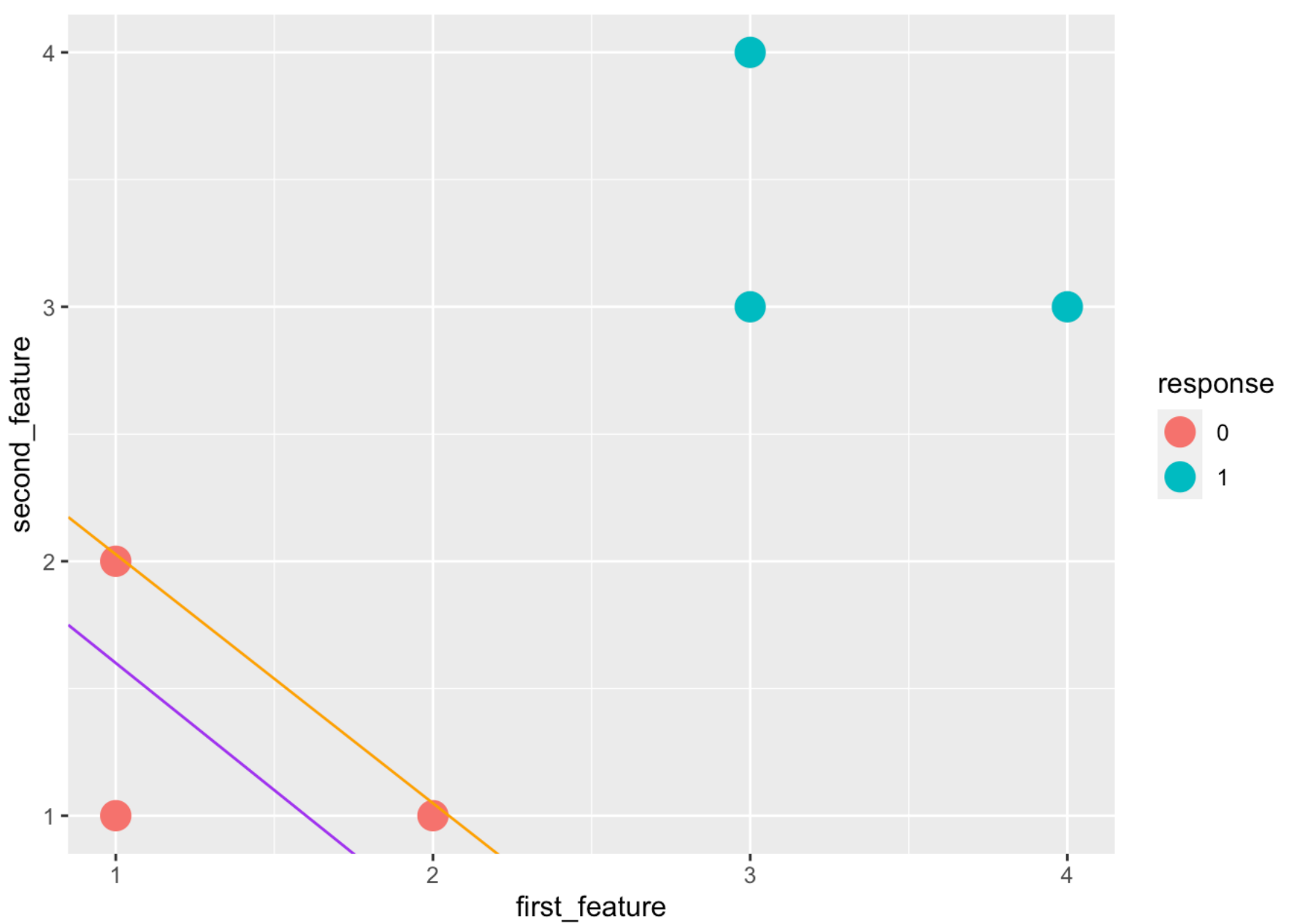
perceptron_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 1000, w = NULL)
{
  X = cbind(1, Xinput)
  n = nrow(X)
  p = ncol(X)
  w = rep(0, p)
  for (iter in 1: MAX_ITER){
    is_error = FALSE
    for (a in 1:n){
      x_a = X[a, ]
      yhat_i = ifelse (sum(X[a, ]*w)> 0, 1, 0)
      if (yhat_i == 0)
        is_error = TRUE
      for(i in 1:p){
        y_i = y_binary[a]
        w[i] = w[i] + (y_i -yhat_i * x_a[i])
      }
    }
    if(is_error == FALSE)
      return(w)
  }
  w
}

w_vec_simple_per = perceptron_learning_algorithm(
  cbind(Xy_simple$first_feature, Xy_simple$second_feature),
  as.numeric(Xy_simple$response == 1)
)

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 + simple_svm_line

```



Is this SVM line a better fit than the perceptron?

#The SVM line is a better fit than the perceptron.

Now write pseudocode for your own implementation of the linear support vector machine algorithm using the Vapnik objective function we discussed.

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.

```

#' Support Vector Machine
#
#' This function implements the hinge-loss + maximum margin linear support vector mac
hine algorithm of Vladimir Vapnik (1963).
#
#' @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's.
#' @param MAX_ITER    The maximum number of iterations the algorithm performs. Defaul
ts to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the hyperplane ve
rsus average hinge loss.
#'                    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){
  #TO-DO: write pseudo code in comments
}

```

If you are enrolled in 342W the following is extra credit but if you're enrolled in 650, the following is required. Write the actual code. You may want to take a look at the `optimx` package. You can feel free to define another function (a "private" function) in this chunk if you wish. R has a way to create public and private functions, but I believe you need to create a package to do that (beyond the scope of this course).

```

#' This function implements the hinge-loss + maximum margin linear support vector mac
hine algorithm of Vladimir Vapnik (1963).
#
#' @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's.
#' @param MAX_ITER    The maximum number of iterations the algorithm performs. Defaul
ts to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the hyperplane ve
rsus average hinge loss.
#'                    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){
  #TO-DO
}

```

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 argument here
  slope = -svm_model_weights[2] / svm_model_weights[3],
  color = "brown")
simple_viz_obj + my_svm_line
```

Is this the same as what the `e1071` implementation returned? Why or why not?

TO-DO

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

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

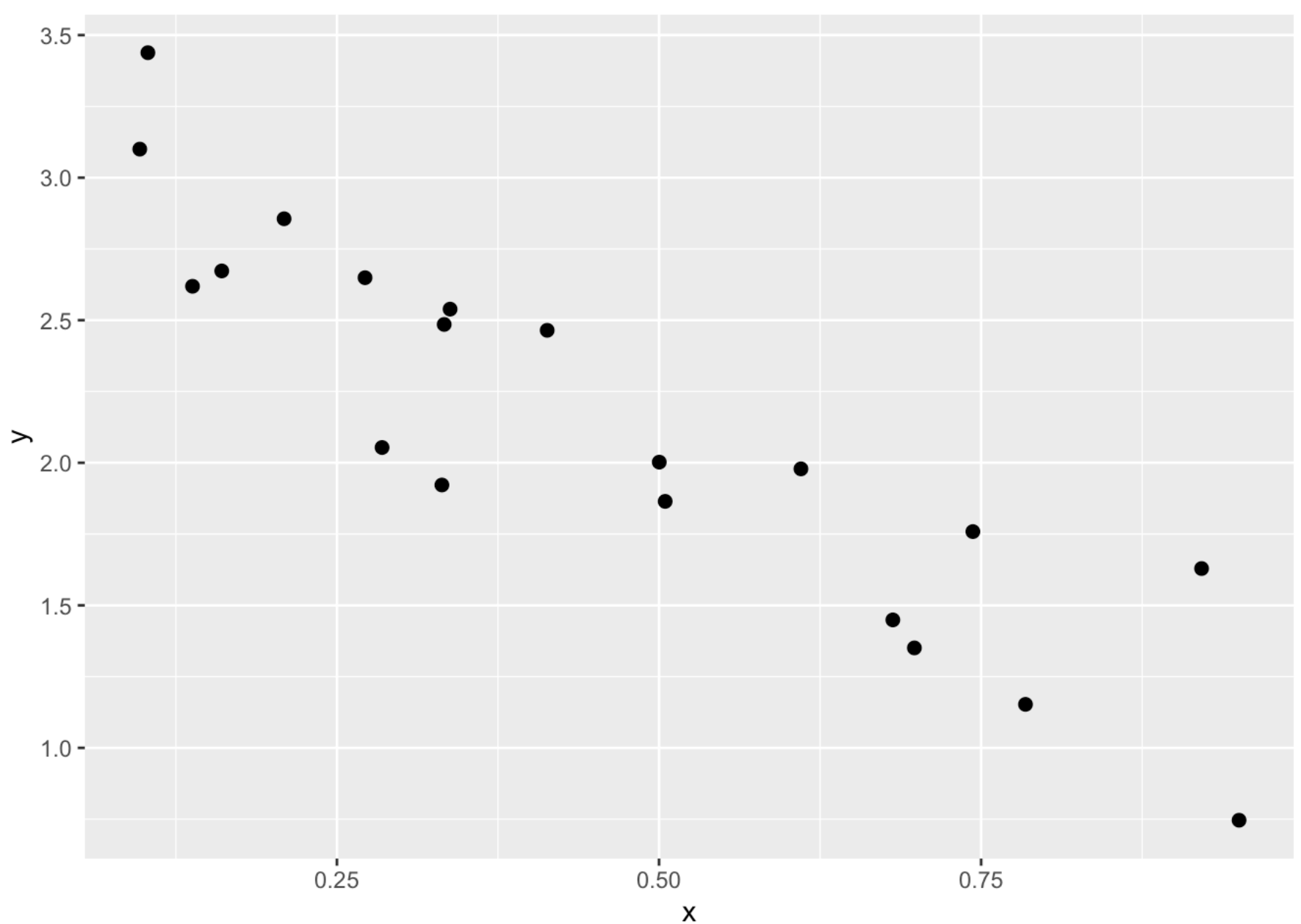
```
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
```

Compute  $h^*(x)$  as `h_star_x`, then draw  $\epsilon \sim N(0, 0.33^2)$  as `epsilon`, then compute `y`.

```
h_star_x = beta_0 + beta_1 * x
epsilon = rnorm(n, 0, 0.33)
y = h_star_x + epsilon
```

Graph the data by running the following chunk:

```
pacman::p_load(ggplot2)
simple_df = data.frame(x = x, y = y)
simple_viz_obj = ggplot(simple_df, aes(x, y)) +
  geom_point(size = 2)
simple_viz_obj
```



Does this make sense given the values of  $\beta_0$  and  $\beta_1$  ?

Write a function `my_simple_ols` that takes in a vector `x` and vector `y` and returns a list that contains the `b_0` (intercept), `b_1` (slope), `yhat` (the predictions), `e` (the residuals), `SSE`, `SST`, `MSE`, `RMSE` and `Rsqr` (for the R-squared metric). Internally, you can only use the functions `sum` and `length` and other basic arithmetic operations. You should throw errors if the inputs are non-numeric or not the same length. You should also name the class of the return value `my_simple_ols_obj` by using the `class` function as a setter. No need to create ROxygen documentation here.



```

my_simple_ols = function(x, y){
  n = length(y)
  x_bar = sum(x)/n
  y_bar = sum(y)/n
  if (length(x) != n) {
    stop("x and y need to be the same length")
  }
  if (class(x) != 'numeric' && class(x) != 'integer') {
    stop("x needs to be numeric")
  }
  if (class(y) != 'numeric' && class(y) != 'integer') {
    stop("y needs to be numeric")
  }
  if (n <= 2) {
    stop("n must be more than 2")
  }

  b_1 = (sum(x*y) - n*x_bar*y_bar) / (sum(x^2) - n*x_bar^2)
  b_0 = y_bar - b_1*x_bar
  yhat = b_0 + b_1*x
  e = y - yhat
  SSE = sum(e^2)
  SST = sum((y - y_bar)^2)
  MSE = SSE / (n-2)
  RMSE = sqrt(MSE)
  Rsq = 1 - SSE / SST

  model = list(b_0 = b_0, b_1 = b_1, yhat = yhat, e = e, SSE = SSE, SST = SST, MSE = MSE, RMSE = RMSE, Rsq = Rsq)

  class(model) = "my_simple_ols_obj"
  model
}

```

Verify your computations are correct for the vectors `x` and `y` from the first chunk using the `lm` function in R:

```

?lm
lm_mod = lm(y~x)
my_simple_ols_mod = my_simple_ols(x,y)
#run the tests to ensure the function is up to spec
pacman::p_load(testthat)
expect_equal(my_simple_ols_mod$b_0, as.numeric(coef(lm_mod)[1]), tol = 1e-4)
expect_equal(my_simple_ols_mod$b_1, as.numeric(coef(lm_mod)[2]), tol = 1e-4)
expect_equal(my_simple_ols_mod$RMSE, summary(lm_mod)$sigma, tol = 1e-4)
expect_equal(my_simple_ols_mod$Rsq, summary(lm_mod)$r.squared, tol = 1e-4)

```

Verify that the average of the residuals is 0 using the `expect_equal`. Hint: use the syntax above.

```

expect_equal(mean(my_simple_ols_mod$e), 0)

```

Create the  $X$  matrix for this data example. Make sure it has the correct dimension.

```
cbind(1,x)
```

```
##           x
## [1,] 1 0.78442972
## [2,] 1 0.16059915
## [3,] 1 0.50019584
## [4,] 1 0.13790659
## [5,] 1 0.69820769
## [6,] 1 0.92112074
## [7,] 1 0.41318732
## [8,] 1 0.61016147
## [9,] 1 0.33324362
## [10,] 1 0.95021723
## [11,] 1 0.33782936
## [12,] 1 0.68148476
## [13,] 1 0.09698472
## [14,] 1 0.50477196
## [15,] 1 0.27181249
## [16,] 1 0.28503717
## [17,] 1 0.10326282
## [18,] 1 0.20898536
## [19,] 1 0.33147864
## [20,] 1 0.74359693
```

Use the `model.matrix` function to compute the matrix `x` and verify it is the same as your manual construction.

```
model.matrix(~x)
```

```
##      (Intercept)          x
## 1          1 0.78442972
## 2          1 0.16059915
## 3          1 0.50019584
## 4          1 0.13790659
## 5          1 0.69820769
## 6          1 0.92112074
## 7          1 0.41318732
## 8          1 0.61016147
## 9          1 0.33324362
## 10         1 0.95021723
## 11         1 0.33782936
## 12         1 0.68148476
## 13         1 0.09698472
## 14         1 0.50477196
## 15         1 0.27181249
## 16         1 0.28503717
## 17         1 0.10326282
## 18         1 0.20898536
## 19         1 0.33147864
## 20         1 0.74359693
## attr(,"assign")
## [1] 0 1
```

Create a prediction method `g` that takes in a vector `x_star` and `my_simple_ols_obj`, an object of type `my_simple_ols_obj` and predicts `y` values for each entry in `x_star`.

```
g = function(my_simple_ols_obj, x_star){
  y_star = my_simple_ols_obj$b_0 + my_simple_ols_obj$b_1 * x_star
}
```

Use this function to verify that when predicting for the average `x`, you get the average `y`.

```
pacman::p_load(testthat)
expect_equal(g(my_simple_ols_mod, mean(x)), mean(y))
```

In class we spoke about error due to ignorance, misspecification error and estimation error. Show that as  $n$  grows, estimation error shrinks. Let us define an error metric that is the difference between  $b_0$  and  $b_1$  and  $\beta_0$  and  $\beta_1$ . How about  $h = ||b - \beta||^2$  where the quantities are now the vectors of size two. Show as  $n$  increases, this shrinks.

```

beta_0 = 3
beta_1 = -2
beta = c(beta_0, beta_1)

ns = 10^(1:6)
error_in_b = array(NA, length(ns))
for (i in 1 : length(ns)) {
  n = ns[i]
  x = runif(n)
  h_star_x = beta_0 + beta_1 * x
  epsilon = rnorm(n, mean = 0, sd = 0.33)
  y = h_star_x + epsilon

  mod = my_simple_ols(x,y)
  b = c(mod$b_0, mod$b_1)

  error_in_b[i] = sum((beta - b)^2)
}
log(error_in_b, 10) # last number should be close to zero

```

```
## [1] -0.2675793 -4.5817217 -4.1039910 -4.1823332 -4.4554674 -5.4950106
```

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 up.

```
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`.  $p$  is 1 and  $n$  is 928 the number of observations

```
pacman::p_load(skimr)
skim(Galton)
```



Data summary

Name	Galton
Number of rows	928
Number of columns	2
<hr/>	

Column type frequency:

numeric	2
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
parent	0	1	68.31	1.79	64.0	67.5	68.5	69.5	73.0	
child	0	1	68.09	2.52	61.7	66.2	68.2	70.2	73.7	

#Its a prediction of a childs height based on their parents and it does not seem to differ to much because you can tell from the standard deviation.

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

```
avg_height = mean(c(Galton$parent, Galton$child))
```

If you were predicting child height from parent height and you were using the null model, what would the RMSE be of this model be?

```
n = nrow(Galton)
SST = sum((Galton$child - mean(Galton$child))^2)
sqrt(SST/(n-1))
```

```
## [1] 2.517941
```

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  $R^2$ .

```
mod = lm(child~parent, Galton)
b_0 = coef(mod)[1]
b_1 = coef(mod)[2]
summary(mod)$sigma
```

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

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

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

Interpret all four quantities:  $b_0$ ,  $b_1$ , RMSE and  $R^2$ . Use the correct units of these metrics in your answer.  $b_0$  is the intercept of the fit in range(ridiculous)  $b_1$  is the increase in child height for one inch increase in average mother/father height. RMSE would show you how average of a child's height is next to the best fit line  $R^2$  tells you how close the child's height is in respect with the fitted regression line.

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

This model is pretty good because it gives approximations that child height does not really differ from parents height.

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

It is reasonable to assume that parents and their children have the same height, because genetics is passed from the parents to the offspring and that includes height.

If they were to have the same height and any differences were just random noise with expectation 0, what would the values of  $\beta_0$  and  $\beta_1$  be?

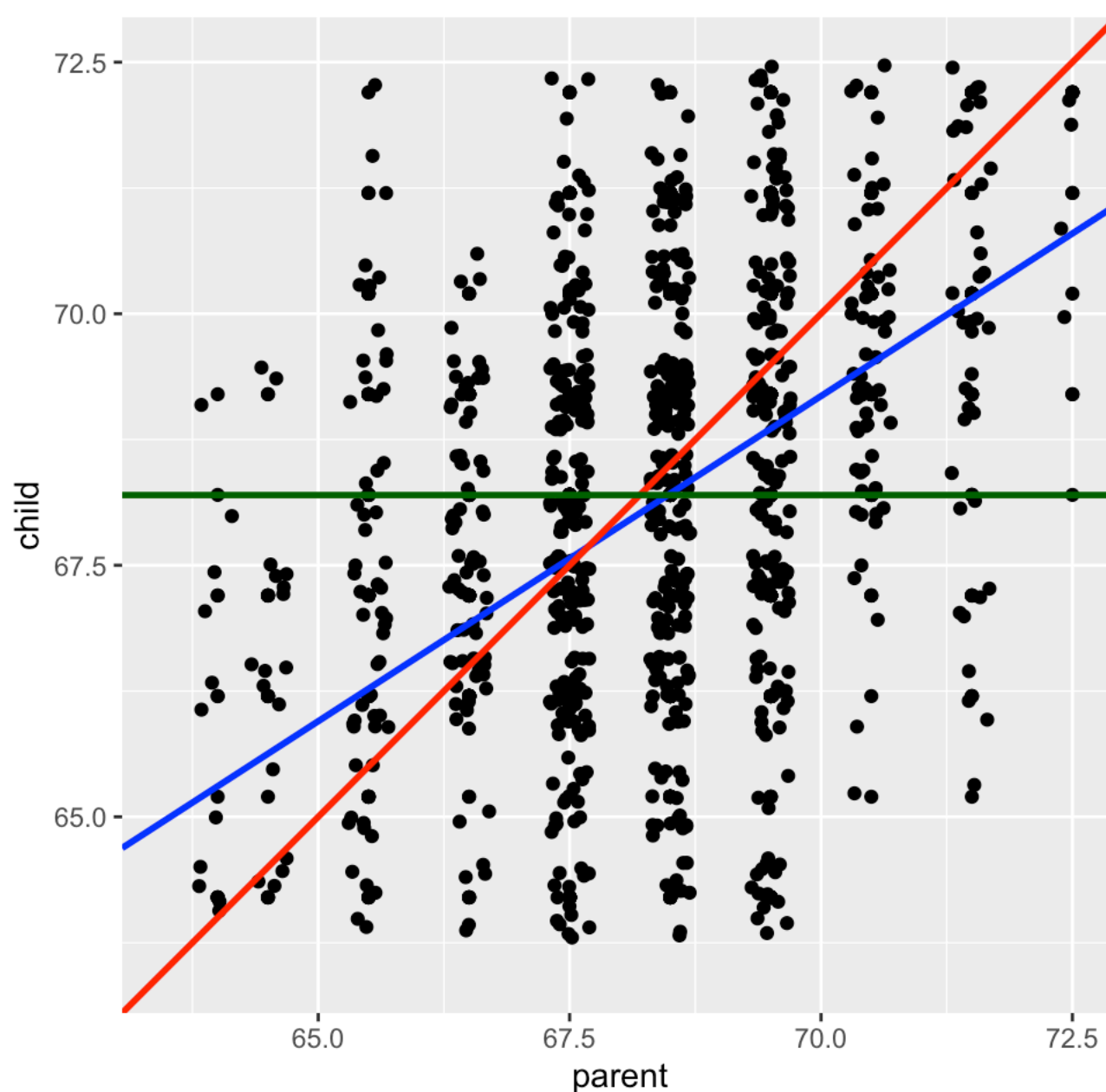
The values would become linear.

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  $\beta_0$  and  $\beta_1$  if the parent-child height equality held in red and (d) the mean height in green.

```
pacman::p_load(ggplot2)
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:

#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 it is what it says the height of the average person.

Why should this effect be real?

#This should be real because on average people's height are close to their parents height.

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.

# $H = X(X^T X)^{-1} X^T$

You can now clear the workspace. Create a dataset  $\mathbb{D}$  which we call  $x_y$  such that the linear model as  $R^2$  about 50% and RMSE approximately 1.

```
x = 1:10
y = x^9
Xy = data.frame(x = x, y = y)
M = lm(x~y)
summary(M)$r.squared
```

```
## [1] 0.5144727
```

```
summary(M)$sigma
```

```
## [1] 2.237633
```

Create a dataset  $\mathbb{D}$  which we call `xy` such that the linear model as  $R^2$  about 0% but `x`, `y` are clearly associated.

```
x = 1:100
y = x^50
Xy = data.frame(x = x, y = y)
M = lm(y~x)
summary(M)$r.squared
```

```
## [1] 0.1148204
```

Extra credit: create a dataset  $\mathbb{D}$  and a model that can give you  $R^2$  arbitrarily close to 1 i.e. approximately  $1 - \epsilon$  but RMSE arbitrarily high i.e. approximately  $M$ .

```
epsilon = 0.01
M = 1000
#TO-DO
```

Write a function `my_ols` that takes in `x`, a matrix with with `p` columns representing the feature measurements for each of the `n` units, a vector of `n` responses `y` and returns a list that contains the `b`, the `p + 1`-sized column vector of OLS coefficients, `yhat` (the vector of `n` predictions), `e` (the vector of `n` residuals), `df` for degrees of freedom of the model, `sse`, `sst`, `mse`, `rmse` and `rsq` (for the R-squared metric). Internally, you cannot use `lm` or any other package; it must be done manually. You should throw errors if the inputs are non-numeric or not the same length. Or if `x` is not otherwise suitable. You should also name the class of the return value `my_ols` by using the `class` function as a setter. No need to create ROxygen documentation here.



```

my_ols = function(X, y){
  if(class(X) != "numeric" && class(y)){
    stop("X or y is not numeric")
  }
  n = length(X)
  if(n != length(y)){
    stop("X or y has to be same length")
  }

  X_bar = sum(X) / length(X)
  y_bar = sum(y) / length(y)
  sX_squared = (1/(n-1) * sum((X - X_bar)^2))
  s_Xy= (1/(n-1)) * sum((X - X_bar)*(y - y_bar))
  b_1 = s_Xy/sX_squared
  b_0 = y_bar - b_1*X_bar
  y_hat = b_0 + b_1*X

  e = y - y_hat
  SSE = sum(e^2)
  SST = sum((y - y_bar)^2)
  Rsq = 1 - SSE/SST
  MSE = SSE / (n-2)
  RMSE = sqrt(MSE)

  model = list(
    b_0 = b_0,
    b_1 = b_1,
    y_hat = y_hat,
    e = e,
    SSE = SSE,
    SST = SST,
    Rsq = Rsq,
    MSE = MSE,
    RMSE = RMSE
  )
  class(model) = "my_ols"
  model
}

```

Verify that the OLS coefficients for the `Type` of cars in the cars dataset gives you the same results as we did in class (i.e. the ybar's within group).

```

cars = MASS::Cars93
anova_mod = lm(Price~Type, cars)
coef(anova_mod)

```

## (Intercept)	TypeLarge	TypeMidsize	TypeSmall	TypeSporty	TypeVan
## 18.212500	6.087500	9.005682	-8.045833	1.180357	0.887500

```
result = my_ols(as.numeric(data.matrix(data.frame(cars$Type))), cars$Price)
```

```
## Warning in b + 0 + b_1 * X: longer object length is not a multiple of shorter  
## object length
```

```
head(result)
```

```

## $b_0
## [1] 22.87102
##
## $b_1
## [1] -1.001939
##
## $y_hat
## [1] -1.008438899 -5.004163681 1.997377428 -5.004163681 -0.006500123
## [6] -5.004163681 0.995438652 -4.002224905 -0.006500123 -4.002224905
## [11] -0.006500123 -3.000286130 1.997377428 -7.008041232 -0.006500123
## [16] -8.009980007 -3.012316450 -4.002224905 -2.010377674 -4.002224905
## [21] 1.997377428 -4.002224905 -1.008438899 -6.006102456 1.997377428
## [26] -8.009980007 -0.006500123 -7.008041232 -1.008438899 -4.002224905
## [31] -1.008438899 -6.006102456 1.997377428 -7.008041232 -2.010377674
## [36] -8.009980007 -0.006500123 -4.002224905 -1.008438899 -7.008041232
## [41] -2.010377674 -6.006102456 1.997377428 -6.006102456 -1.008438899
## [46] -7.008041232 -0.006500123 -5.004163681 -0.006500123 -5.004163681
## [51] -0.006500123 -4.002224905 -1.008438899 -6.006102456 1.997377428
## [56] -8.009980007 -2.010377674 -3.000286130 -0.006500123 -7.008041232
## [61] -0.006500123 -6.006102456 -0.006500123 -6.006102456 1.997377428
## [66] -8.009980007 -0.006500123 -3.000286130 -0.006500123 -8.009980007
## [71] 0.995438652 -7.008041232 -1.008438899 -3.000286130 -2.010377674
## [76] -5.004163681 0.995438652 -3.000286130 -1.008438899 -6.006102456
## [81] -1.008438899 -3.000286130 -1.008438899 -6.006102456 -2.010377674
## [86] -5.004163681 -3.012316450 -6.006102456 -3.012316450 -3.000286130
## [91] -2.010377674 -3.000286130 -0.006500123
##
## $e
## [1] 16.908439 38.904164 27.102623 42.704164 30.006500 20.704164 19.804561
## [8] 27.702225 26.306500 38.702225 40.106500 16.400286 9.402623 22.108041
## [15] 15.906500 24.309980 19.612316 22.802225 40.010378 22.402225 13.802623
## [22] 33.502225 10.208439 17.306102 11.302623 27.009980 15.606500 32.808041
## [29] 13.208439 23.302225 8.408439 16.106102 9.302623 22.908041 16.010378
## [36] 27.909980 20.206500 24.902225 9.408439 19.508041 21.810378 18.106102
## [43] 15.502623 14.006102 11.008439 17.008041 13.906500 52.904164 28.006500
## [50] 40.204164 34.306500 40.102225 9.308439 17.606102 14.502623 27.109980
## [57] 34.510378 34.900286 61.906500 21.108041 14.906500 16.306102 26.106500
## [64] 17.806102 13.702623 27.109980 21.506500 16.500286 16.306500 27.509980
## [71] 19.704561 21.408041 10.008439 14.100286 19.710378 23.504164 23.404561
## [78] 31.700286 12.108439 14.406102 11.908439 22.500286 9.608439 15.806102
## [85] 20.410378 23.204164 25.712316 15.106102 22.712316 23.000286 25.310378
## [92] 25.700286 26.706500
##
## $SSE
## [1] 55339.15
##
## $SST
## [1] 8584.021

```

Create a prediction method `g` that takes in a vector `x_star` and the dataset  $\mathbb{D}$  i.e. `x` and `y` and returns the OLS predictions. Let `x` be a matrix with `p` columns representing the feature measurements for each of the `n` units

```
g = function(x_star, X, y){  
  my_ols(X,y)$b_0 + my_ols(X,y)$b_1 * x_star  
}  
predict(anova_mod, cars[1,])
```

```
##           1  
## 10.16667
```