

## Lab 3

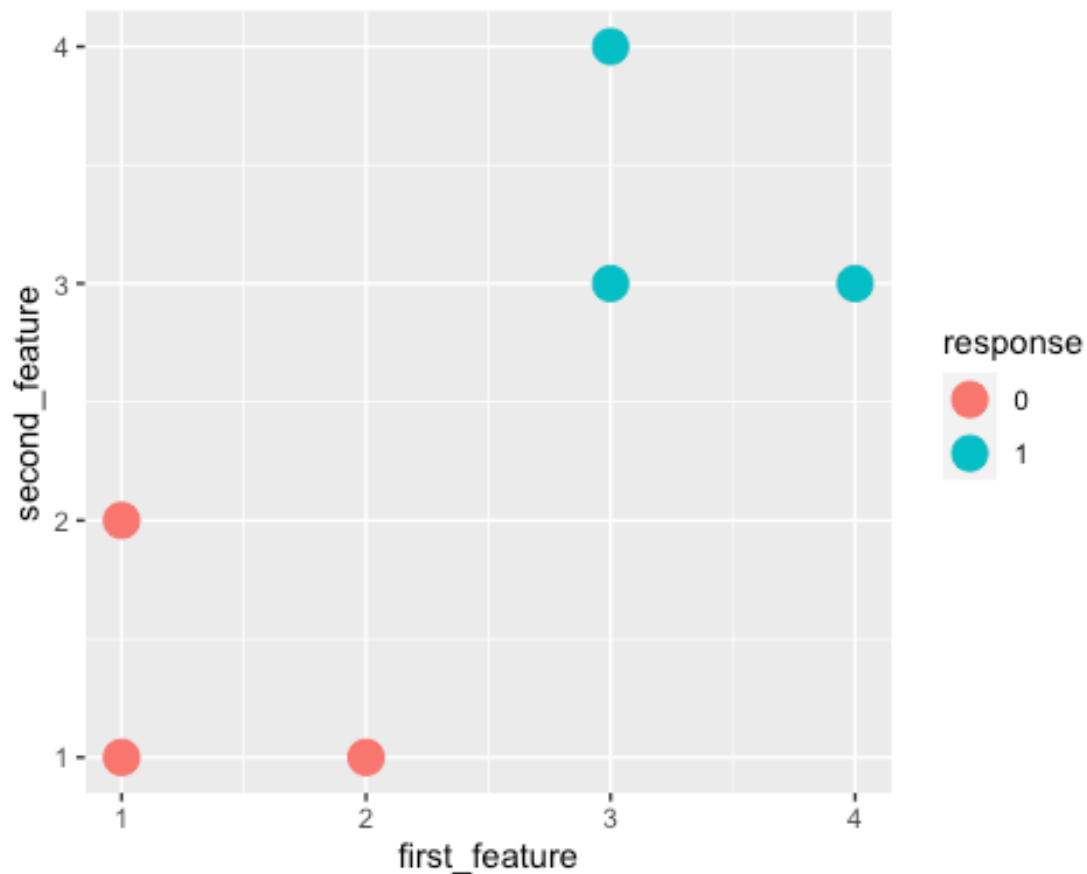
Shoara Chowdhury

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)),
  first_feature = c(1, 1, 2, 3, 3, 4),
  second_feature = c(1, 2, 1, 3, 4, 3)
)
simple_viz_obj = ggplot(Xy_simple, aes(x = first_feature, y = second_feature,
color = response)) +
  geom_point(size = 5)
simple_viz_obj
```

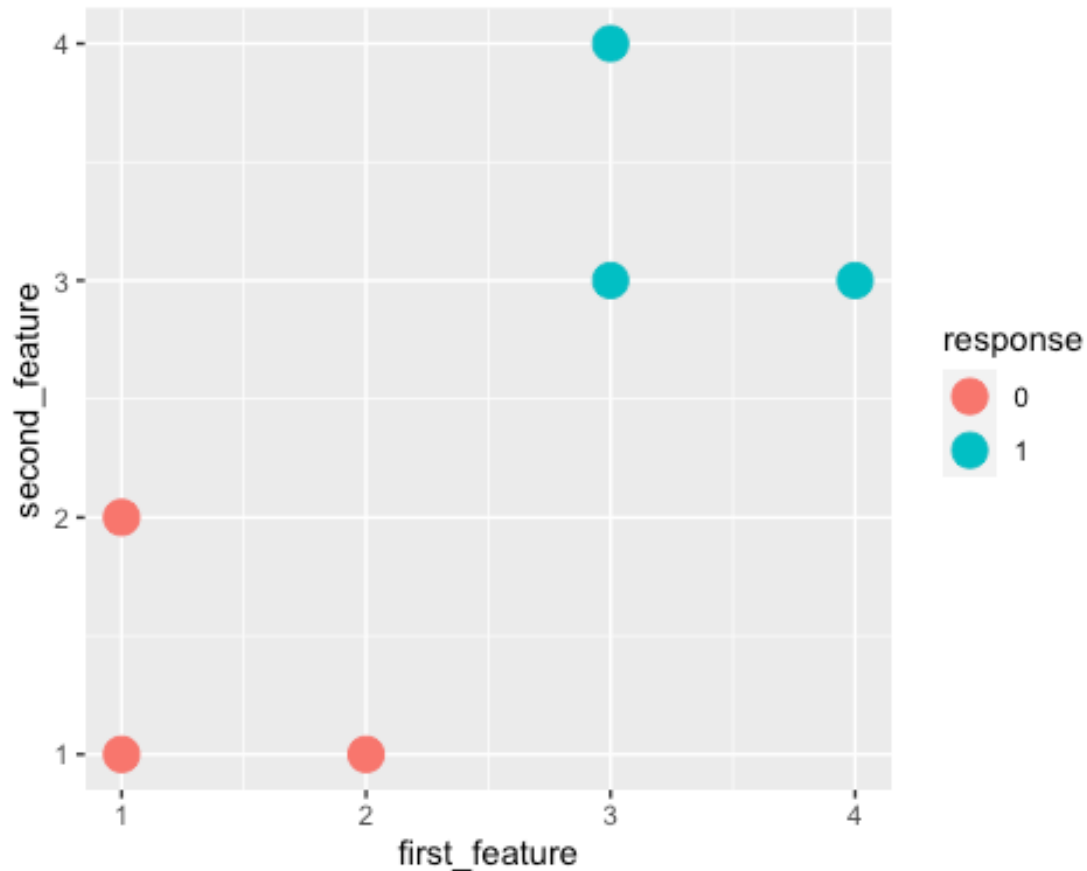


Use the `e1071` package to fit an SVM model to the simple data. Use a formula to create the model, pass in the data frame, set kernel to be linear for the linear SVM and don't scale the covariates. Call the model object `svm_model`. Otherwise the remaining code won't work.

```
pacman::p_load(e1071)
svm_model = svm(
  formula = second_feature ~ .,
  data = Xy_simple,
  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:

```
#TO DO
```

```
''' ''' Is this SVM line a better fit than the perceptron?
```

No.

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 machine 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. Defaults to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the
hyperplane versus 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 machine 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. Defaults to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the
hyperplane versus 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:

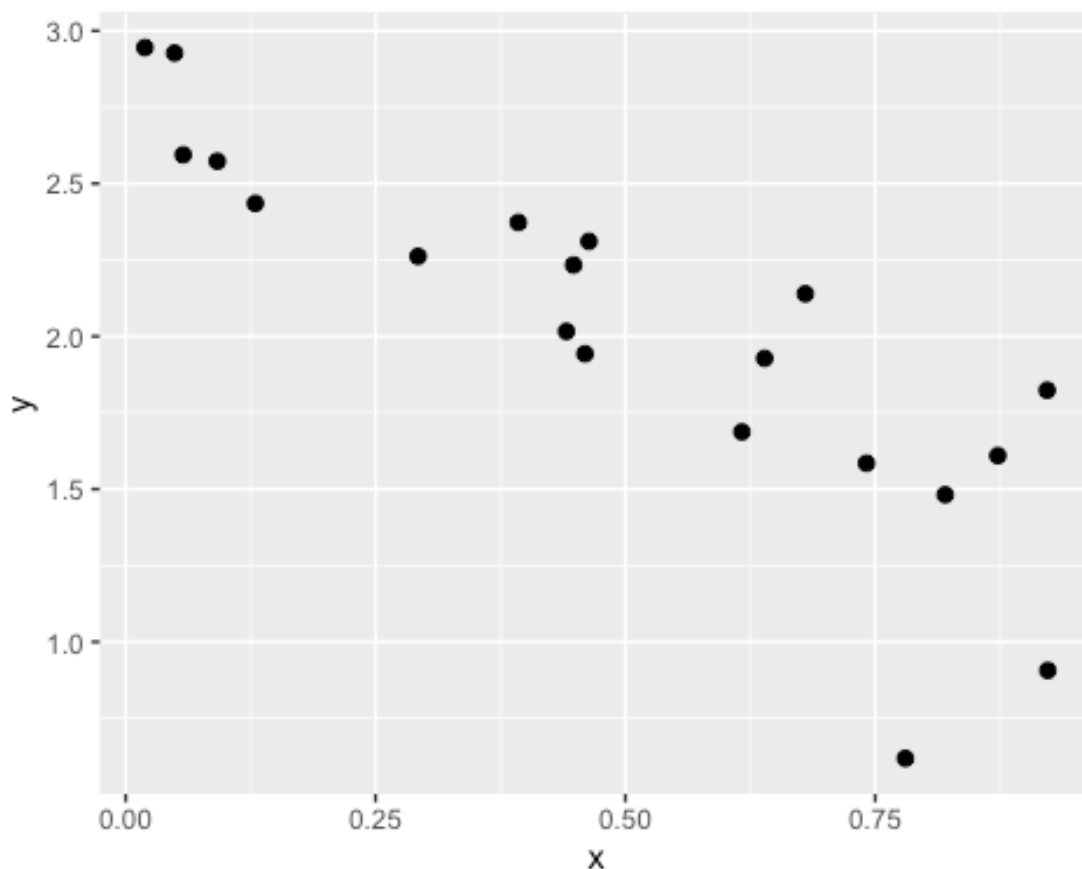
```
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
```

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  $\beta_1$ ? Yes.

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 `Rsq` (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 Rxygen documentation here.

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

  n = length(y)
  x_bar= sum(x)/n
  y_bar= sum(y)/n
  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
  y_hat= b_0 +b_1*x
  e = y- y_hat
  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,y_hat= y_hat,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_mod = #TO DO
#my_simple_ols_mod = # TO DO
#run the tests to ensure the function is up to spec
#pacman::p_load(testthat)
#expect_equal(my_simple_ols_mod$b_1, as.numeric(coef(lm_mod)[2]), tol = 1e-4)
##expect_equal(my_simple_ols_mod$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.

```
#mean(my_simple_ols_mod$e)
#expect_equal(mean(my_simple_ols_mod$e), 0)
```

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

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

```
#TO-DO
```

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){
  #TO-DO
}
```

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

```
#expect_equal(g(my_simple_ols_obj, 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 = #TO-DO
#errors = 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

  #TO-DO
  # errors[i] = sum((beta - b)^2)
#}
#errors
```

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
```

```
Column type frequency:
numeric          2
```

```
Group variables  None
```

#### Variable type: numeric

skim_variab le	n_missin g	complete_ra te	mea n	sd	p0	p2 5	p5 0	p7 5	p10 0	hist
parent	0	1	68.3	1.7	64.	67.	68.	69.	73.0	■ ■ ■ ■
			1	9	0	5	5	5		■
child	0	1	68.0	2.5	61.	66.	68.	70.	73.7	■ ■ ■ ■
			9	2	7	2	2	2		■

TO-DO

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

*#TO-DO*

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

*#TO-DO*

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  $R^2$ .

*#TO-DO*



Interpret all four quantities:  $b_0$ ,  $b_1$ , RMSE and  $R^2$ . Use the correct units of these metrics in your answer.

TO-DO

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

TO-DO

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.

TO-DO

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?

TO-DO

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)
```

Fill in the following sentence:

TO-DO: Children of short parents became ... on average and children of tall parents became ... on average.

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

TO-DO

Why should this effect be real?

TO-DO

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.

TO-DO

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

```
x = #TO-DO
y = #TO-DO
Xy = data.frame(x = x, y = y)
```

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 = #TO-DO
#y = #TO-DO
Xy = data.frame(x = x, y = y)
```

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 `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 `Rsqr` (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 Rxygen documentation here.

```
my_ols = function(X, y){
  #TO-DO
}
```

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).

```
#TO-DO
```

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){
  #TO-DO
}
```