Lab 3: Cross Validation and Model Flexibility

Getting started

Click the Github Assignment link on Canvas, then open it in Posit Cloud as a new project from a GitHub repository.

Warm up

Let's warm up with some simple exercises. Open the `lab3.qmd` Quarto file. Update the YAML of your Quarto file with your information, render, commit, and push your changes. Make sure to commit with a meaningful commit message. Then, go to your repo on GitHub and confirm that your changes are visible in your `.qmd` file. If anything is missing, commit and push again.

Packages

We'll use the `tidyverse` package for data wrangling and visualization. We will also be using the `boot` package to help with cross-validation. We will also make use of the `gbm` package for one of the models we fit below called a gradient boosting machine.

```r
library(boot)
library(tidyverse)
library(gbm)
```

Data

You will simulate your own data for this lab (with our help, don't worry!). Simulation is a really powerful tool when doing data science and machine learning, in particular when we are working on developing our own tools and techniques *and* when we are learning about existing tools. When we simulate our own data, we control the process – we know the *truth*, which we are usually only trying to estimate in real world applications. ISLR uses simulation to demonstrate how and why certain machine learning techniques work – and you will do the same in this lab.

You will simulate data drawn from two univariate $f(X)$ curves: one more linear and one less linear.[1]

ϵ in each case defines the unexplained randomness in our outcome. For simplicity, we will say $\epsilon \sim \mathcal{N}(0, 1)$.

More Linear

$$\begin{aligned} Y_1 &= f_{Y_1}(X_1) + \epsilon \\ &= 2 + X_1 - 0.012X_1^2 + \epsilon \end{aligned}$$

Nonlinear

$$\begin{aligned} Y_2 &= f_{Y_2}(X_1) + \epsilon \\ &= 2 + \sin(1.5\sin(1.2X_1) + 0.5\cos(0.5(X_1 - 1)^2)) + 0.3X_1 + \epsilon \end{aligned}$$

Simulation

We will simulate 1000 observations for each Y, using the same X_1.

```r
# set seed for replicability purposes
set.seed(280)
# the number of observations in our data set
n <- 1000
# x1 ~ beta(1.1, 1.1) * 20 - 10 (constrained to between -10 and 10)
x1 <- rbeta(n, shape1 = 1.1, shape2 = 1.1) * 20 - 10
```

[1]We could do this in the multivariate case, but then it becomes much harder to visualize. But try it out on your own if you want after this lab!

```r
# Simulating n y1 values
y1 <- 2 + x1 - 0.012 * x1^2 + rnorm(n, mean = 0, sd = 1)
# And doing the same with y2
y2 <- 2 + sin(1.5 * sin(1.2 * x1) + 0.5 * cos(0.5 * (x1 - 1)^2)) + 0.3 * x1 +
  rnorm(n, mean = 0, sd = 1)


# putting all in a tibble
sim_data <- tibble(x1 = x1,
                   y1 = y1,
                   y2 = y2)
```

Models

You will use two modeling approaches: 1. A regular linear regression model. 2. A *gradient boosting machine* (GBM)

The linear model is much less flexible than the GBM. We have not talked about GBMs yet, but we will during module 7 when we talk about tree methods.

The goal of the lab is not to force you to learn how to use GBMs on your own; it is to help you see why cross-validation is important versus just a validation-test split and to give you an intuition for why less flexible methods are perfectly fine in many cases. Because of this, all of the GBM-related code will be provided for you. You'll learn to write this code on your own later in the semester!

You will be responsible for the OLS-related code. Remember that you can use the example code for this module (`example-ISLR5` in Posit Cloud) as a template for much of the code in this lab.

We will reuse the `calc_mse()` function that we made in that example code, for convenience's sake.

```r
# function takes two arguments:
## true - the true Y values
## pred - the predicted Y values
## observations in true and pred *must* be in same order
calc_mse <- function(true, pred, subset = NULL){
  error <- (true - pred)^2
  if(!is.null(subset)) error <- error[subset]
  mean(error)
}
# note that R functions don't need return statement - last object evaluated in
# function body gets returned
```

Exercises

Data Visualization

Q1

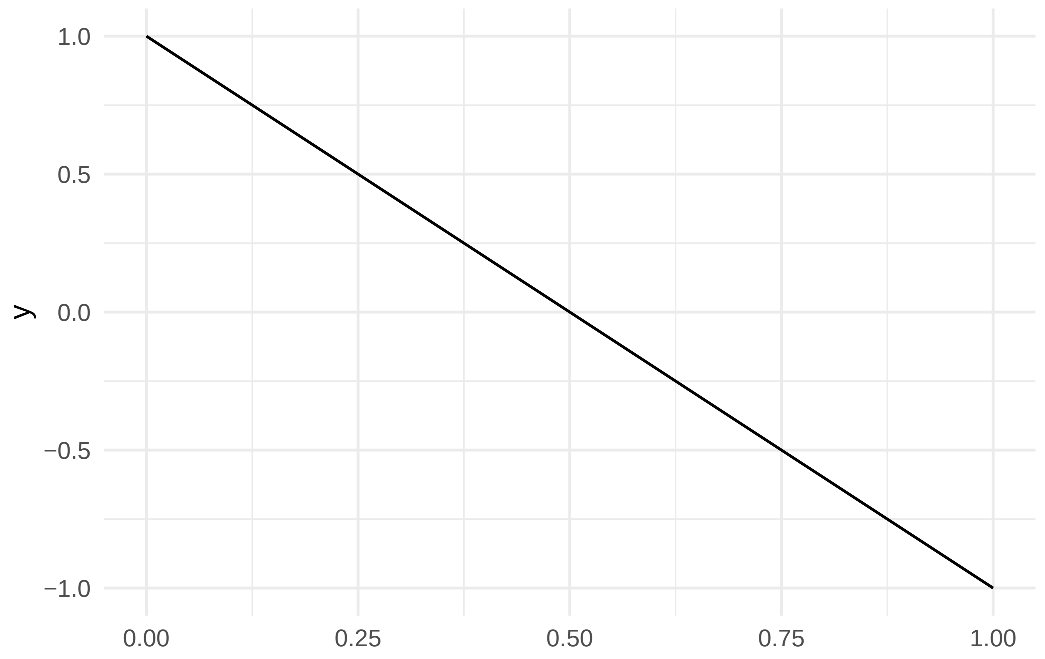
Visualize the relationship between `x1` and `y1` and `y2`. Use a *hollow-point* scatter plot onto which you superimpose the *true* $f(x)$. Increase the `linewidth` of the true function to 1.25. Describe the two relationships.

Hints: - **ggplot2** has the `geom_function()` geom, which allows you to specify a mathematical function in the `fun =` argument. This function should take one argument and return one output and should match the true function defined in the Simulations section. For example, the function $Y = -2X + 1$ would take the form of the R function

```
example_fun <- function(x){
  -2*x + 1
}
# to demonstrate Y = -2 * 3 + 1 = -5
example_fun(3)
```

```
[1] -5
```

```
# to give an example of how geom_function() can work
ggplot() +
  geom_function(fun = example_fun) +
  theme_minimal()
```



- `geom_point()` has a `shape =` argument that allows you to change what the points. There are 15 possible shapes - take a look at `?shape` to see the help file for this.

Validation Set Variance

In this section, we will see how choosing different validation sets can result in different estimates of test error.

Remember that the validation set approach involves randomly setting aside a portion of the data (the validation set), fitting the model on the remaining data (the training set), and then calculating prediction error on the held out data (the validation set, which the model hasn't seen before).

Q2

Use the `sample()` function to create a validation, training set split for our data, where 25% of the data goes into the validation set and the remaining 75% go into the test set.

Use `set.seed(1)`.

Q3

Fit models for both `y1` and `y2` using the `glm` function. Then calculate the validation set MSEs to estimate the test MSE for these two fits. Store these as `y1_linear_mse` and `y2_linear_mse`.

Then run the code in the `predictions-gbm` chunk. This code generates the predicted values for the validation set - use these to calculate the MSEs for the GBM models. Store these as `y1_gbm_mse` and `y2_gbm_mse`.

You will have then have four sets of MSEs. Display and compare the four different estimated test MSEs. What do you notice? Which model fits the data better in each case?

Hint: While you may not know how GBMs work, you *do* know how to interpret MSEs.

```
# function to calculate predicted values
predict_gbm <- function(y_train, x_train, x_val){

  df_test <- data.frame(y = y_train,
                        x = x_train)

  boost_model <- gbm(y ~ x,
                     data = df_test,
                     interaction.depth = 1,
                     n.minobsinnode = as.integer(.05 * length(y_train)),
                     n.trees = 500,
                     shrinkage = .1,
                     bag.fraction = 1,
                     distribution = "gaussian")

  df_val <- data.frame(x = x_val)

  boost_pred <- predict(boost_model, newdata = df_val,
                        n.trees = 500)
  boost_pred
}

# fitting the two gbm models
y1_gbm_pred <- predict_gbm(sim_data$y1[train_index],
                           sim_data$x1[train_index],
                           sim_data$x1[-train_index])
y2_gbm_pred <- predict_gbm(sim_data$y2[train_index],
                           sim_data$x1[train_index],
                              sim_data$x1[-train_index])
```

Q4

We can use the `predict_gbm()` function as a function to plot the fitted functions on top of the plots we made in Q1. This can help us compare how the two model fare against the *true* function. Fill in the code in the next chunk. You have to copy your code from Q1 and also add a regression line in each case. The drawing of the GBM fit is provided to you.

What do you notice?

```
# linear function
# insert code from Q1 here
# don't forget the +'s!
# drawing GBM predicted curve
  geom_function(aes(color = "GBM Model"),
                    fun = predict_gbm,
                args = list(x_train = sim_data$x1[train_index],
                            y_train = sim_data$y1[train_index]),
                linewidth = 1.25)

# nonlinear function
# insert code from Q1 here
# don't forget the +'s!
# drawing GBM predicted curve
  geom_function(aes(color = "GBM Model"),
                    fun = predict_gbm,
                args = list(x_train = sim_data$x1[train_index],
                            y_train = sim_data$y2[train_index]),
                linewidth = 1.25)
```

Q5

Repeat Q3, but this time use a different training/validation split where you use `set.seed(3)`. What do you notice about the MSEs? Do they tell a different story than in Q3?

! Remember to Commit!

Render, commit, and push your changes to GitHub with an appropriate commit message.

K-Fold Cross Validation

Q6

Given what you found in Q5, why does it seem a good idea to use cross-validation here?

Q7

Let's try some K-fold cross validation. Here again, a lot of the GBM code will be provided for you so that you can spend most of your time understanding the logic behind cross validation and applying your knowledge of OLS. Because it is fairly clear that we know that the GBM is better than the linear model in the nonlinear data, we will focus on the more linear data here.[2]

We will choose $K = 10$. Use the `cv.glm()` function to do cross-validation of the linear model for `y1`. Use `set.seed(4)` at the beginning of the chunk.

Run the code in chunk `cv-gbm` to calculate the cross-validation MSE for the GBM model.

```
# gbm has bug when cross-validating with a single predictor
# so we are adding a non-informative predictor
# gbm throws a warning, but it's their bug!
sim_data$x2 <- 0

set.seed(4)
y1_gbm_cv <- gbm(y1 ~ x1 + x2,
                     data = sim_data,
                     interaction.depth = 1,
                     n.minobsinnode = 5,
                     n.trees = 500,
                     shrinkage = .1,
                     bag.fraction = 1,
                     cv.folds = 10,
                     distribution = "gaussian")$cv.error[500]
```

Compare the two. According to cross-validation, which seems to fit better? By a lot or only by a little?

Given that the GBM takes longer to fit, do you think it makes sense to prefer it over the linear model in this case? Relate this to what you learned in ISLR Ch. 2 on flexibility vs interpretability.

[2]Feel free to try this out with the nonlinear data in your free time if you are curious/want practice!

❗ Remember to Commit!

Render, commit, and push your changes to GitHub with an appropriate commit message.

Final Steps

When you have made your final edits, remember to render, stage the changed files, then commit, and push to the repo.

Don't forget to submit your pdf on Canvas.

Grading

Please put your names on your answer document and add the date. You won't get points for doing this, but you will *lose* points for not doing so.

Remember to comment your code. You don't get points for commenting, but you will *lose* points for not commenting your code.

For clarity's sake, you should label your code chunks (this helps make debugging easier). You will *lose* points for not naming chunks.

If an exercise asks a question, you should answer it in narrative form and not just rely on the code. You will be *penalized* otherwise.

Please suppress warnings and messages. You will *lose* points if you do not suppress warnings and messages.

If you are asked to make a visualization, make sure that it is presentable, professional in appearance, has a title, and has properly labeled axes. You will *lose* points if you do not do this.

Points Breakdown

The different components of this lab are worth the following points:

- Q1: 8 pts (4 pts for each plot)
- Q2: 1 pt
- Q3: 6 pts (2 pts for fitting linear models, 2 pts for calculating all MSEs, 2 pts for displaying and comparing MSEs)
- Q4: 5 pts (2 pts for each plot, 1 pt for interpretation)
- Q6: 2 pts

- Q7: 4 pts (1 pt for calculating linear CV MSE, 1 pt for comparison, 2 pt for interpretation)
- Workflow (includes making sufficient commits): 4 pts

Total: 30 pts