Lab 14: Statistical Prediction

Name: Rufus Petrie

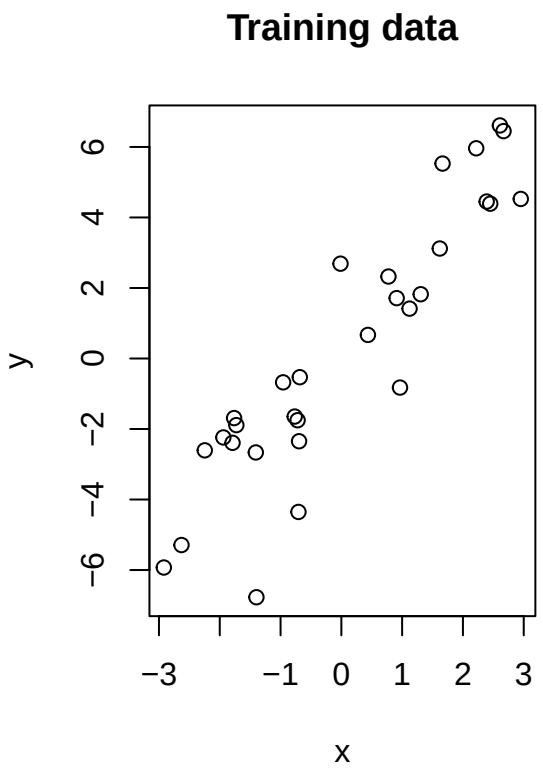
This week's agenda: understanding training and testing errors, implementing sample-splitting and cross-validation, and trying a bunch of statistical prediction methods (optional).

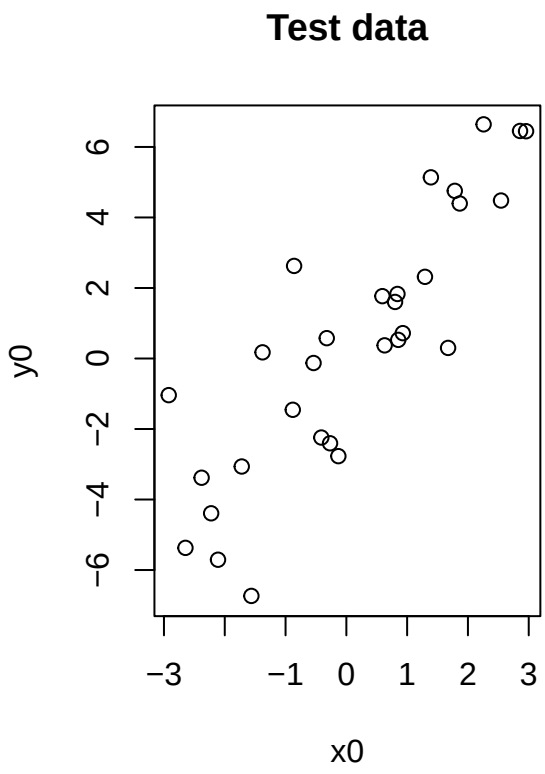
Practice with training and test errors

The code below generates and plots training and test data from a simple univariate linear model, as in lecture. (You don't need to do anything yet.)

```
set.seed(1)
n = 30
x = sort(runif(n, -3, 3))
y = 2*x + 2*rnorm(n)
x0 = sort(runif(n, -3, 3))
y0 = 2*x0 + 2*rnorm(n)

par(mfrow=c(1,2))
xlim = range(c(x,x0)); ylim = range(c(y,y0))
plot(x, y, xlim=xlim, ylim=ylim, main="Training data")
plot(x0, y0, xlim=xlim, ylim=ylim, main="Test data")
```

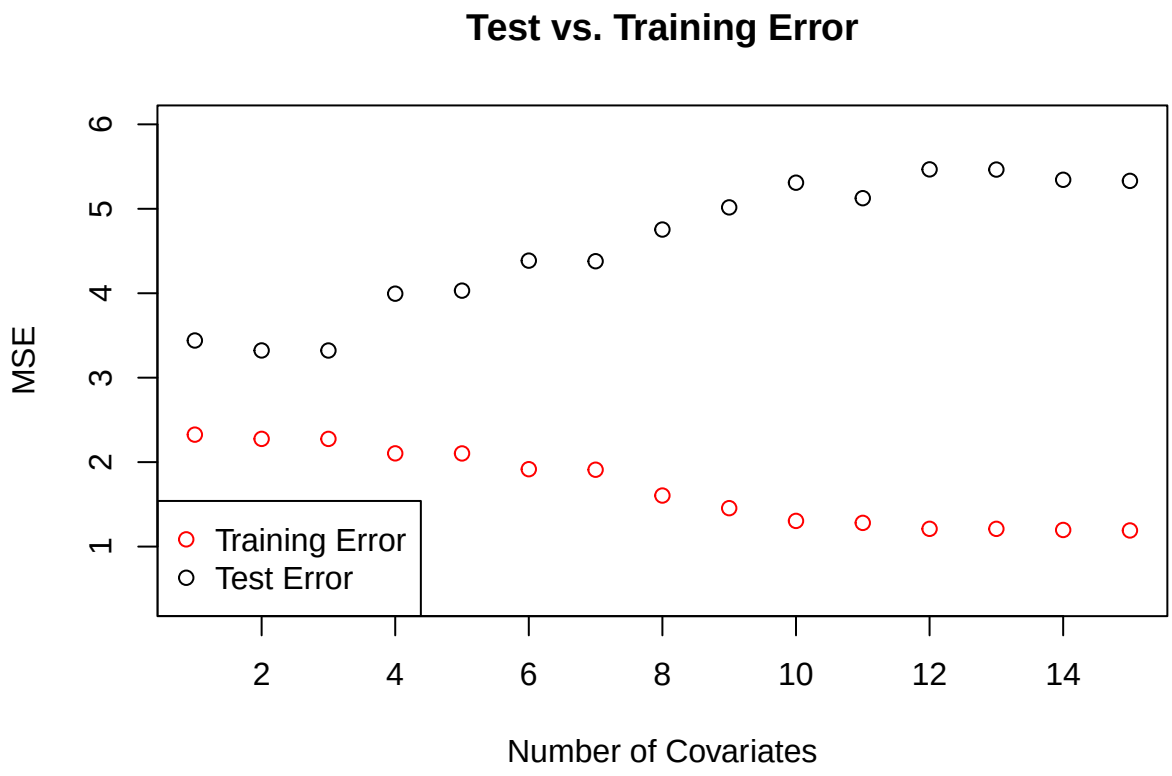




- **1a.** For every k in between 1 and 15, regress `y` onto a polynomial in `x` of degree k. Hint: look at the lecture to see how to use the `poly()` function. Then use this fitted model to predict `y0` from `x0`, and record the observed test error. Also record the observed training error. Plot the test error and training errors curves, as functions of k, on the same plot, with properly labeled axes, and an informative legend. What do you notice about the relative magnitudes of the training and test errors? What do you notice about the shapes of the two curves? If you were going to select a regression model based on training error, which would you choose? Based on test error, which would you choose?

```
k <- 15
error_matrix = matrix(0, k, 2)
for(i in 1:k){
  model <- lm(y~poly(x, i))
  training_predictions <- predict(model, data.frame(x))
  training_error <- mean((y-training_predictions)^2)
  test_predictions <- predict(model, data.frame(x0))
  test_error <- mean((y0-test_predictions)^2)
  error_matrix[i, 1] <- training_error
  error_matrix[i, 2] <- test_error
}

plot(1:15, 1:15/2.5, col="white", main = "Test vs. Training Error",
     xlab = "Number of Covariates", ylab = "MSE")
points(x = 1:15, y = error_matrix[,2])
points(x = 1:15, y = error_matrix[,1], col="red")
legend("bottomleft", legend=c("Training Error", "Test Error"),
       col=c("red", "black"), pch=21)
```



Training error is always lower than test error. Training error decreases steadily, but test error decreases then starts to increase. Based on the training error, you would select the model with 15 covariates, but based on the test error, you would select the model with 3 covariates.

- **1b.** Without any programmatic implementation, answer: what would happen to the training error in the current example if we let the polynomial degree be as large as 29?

Overfitting like this would let the training error become arbitrarily small.

- **1c.** Modify the above code for the generating current example data so that the underlying trend between `y` and `x`, and `y0` and `x0`, is cubic (with a reasonable amount of added noise). Recompute training and test errors from regressions of `y` onto polynomials in `x` of degrees 1 up to 15. Answer the same questions as before, and notably: if you were going to select a regression model based on training error, which would you choose? Based on test error, which would you choose?

```
set.seed(1)
n = 30
x = sort(runif(n, -3, 3))
y = 2*x^3 + 2*rnorm(n)
x0 = sort(runif(n, -3, 3))
y0 = 2*x0^3 + 2*rnorm(n)

k <- 15
error_matrix = matrix(0, k, 2)
for(i in 1:k){
  model <- lm(y~poly(x, i))
  training_predictions <- predict(model, data.frame(x))
  training_error <- mean((y-training_predictions)^2)
```

```
  test_predictions <- predict(model, data.frame(x0))
  test_error <- mean((y0-test_predictions)^2)
  error_matrix[i, 1] <- training_error
  error_matrix[i, 2] <- test_error
}

# plot(1:15, 1:15*6, col="white", main = "Test vs. Training Error",
#      xlab = "Number of Covariates", ylab = "MSE")
# points(x = 1:15, y = error_matrix[,2])
# points(x = 1:15, y = error_matrix[,1], col="red")
# legend("top", legend=c("Training Error", "Test Error"),
#        col=c("red", "black"), pch=21)
```

The answers are all basically the same. For training error, you would select the highest degree polynomial possible, but for test error, you would select degree 3.

Sample-splitting with the prostate cancer data

Below, we read in data on 97 men who have prostate cancer (from the book The Elements of Statistical Learning). (You don't need to do anything yet.)

```
pros.df = read.table(
  "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data")
dim(pros.df)
```

```
## [1] 97 10
```

```
head(pros.df)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

- **2a.** As we can see, the designers of this data set already defined training and test sets for us, in the last column of `pros.ds`! Split the prostate cancer data frame into two parts according to the last column, and report the number of observations in each part. On the training set, fit a linear model of `lpsa` on `lcavol` and `lweight`. On the test set, predict `lpsa` from the `lcavol` and `lweight` measurements. What is the test error?

```
pros_train <- pros.df[pros.df$train==1,]
pros_test <- pros.df[pros.df$train==0,]
dim(pros_train)
```

```
## [1] 67 10
```

```
dim(pros_test)
```

```
## [1] 30 10
```

```
pros_lm <- lm(lpsa~lcavol+lweight, data=pros_train)
predictions <- predict(pros_lm, pros_test)
mean((pros_test$lpsa-predictions)^2)
```

```
## [1] 0.4924823
```

The test error equals 0.49.

- **2b.** Using the same training and test set division as in the previous question, fit a linear model on the training set `lpsa` on `age`, `gleason`, and `pgg45`. Then on the test set, predict `lpsa` from the relevant predictor measurements. What is the test error?

```
pros_lm <- lm(lpsa~age+gleason+pgg45, data=pros_train)
predictions <- predict(pros_lm, pros_test)
mean((pros_test$lpsa-predictions)^2)
```

```
## [1] 1.022471
```

The new test error equals 1.02.

- **2c.** How do the test errors compare in the last two questions? Based on this comparison, what regression model would you recommend to your clinician friend? What other considerations might your clinician friend have when deciding between the two models that is not captured by your test error comparison?

The model from part a has the lower test error, so I would recommend that one. However, we should note that the first model uses measurements of the tumor, whereas the second model uses historical patient information, so the second model may be much cheaper to implement.

- **Challenge.** The difference between the test errors of the two linear models considered above seems significant, but we have no sense of variability of these test error estimates, since it was just performed with one training/testing split. Repeatedly, split the prostate cancer data frame randomly into training and test sets of roughly equal size, fit the two linear models on the training set, and record errors on the test set. As a final estimate of test error, average the observed test errors over this process for each of the two model types. Then, compute the standard deviation of the test errors over this process for each of the two model types. After accounting for the standard errors, do the test errors between the two linear model types still appear significantly different?

```
set.seed(0)
n <- 100
error_matrix = matrix(0, n, 2)
for(i in 1:n){
  inds = sample(rep(0:1, length=97))
  pros.df$train <- inds
  pros_train <- pros.df[pros.df$train==1,]
  pros_test <- pros.df[pros.df$train==0,]

  pros_lm <- lm(lpsa~lcavol+lweight, data=pros_train)
  predictions <- predict(pros_lm, pros_test)
  error_matrix[i, 1] <- mean((pros_test$lpsa-predictions)^2)

  pros_lm <- lm(lpsa~age+gleason+pgg45, data=pros_train)
  predictions <- predict(pros_lm, pros_test)
  error_matrix[i, 2] <- mean((pros_test$lpsa-predictions)^2)
}
```

```
mean(error_matrix[,1])
```

```
## [1] 0.5927849
```

```
sd(error_matrix[,1])
```

```
## [1] 0.08473976
```

```
mean(error_matrix[,2])
```

```
## [1] 1.271597
```

```
sd(error_matrix[,2])
```

```
## [1] 0.2395514
```

After accounting for randomness in sample splitting, the test errors between the two models still appear significant.

Sample-splitting with the wage data

Below, we read in data on 3000 individuals living in the mid-Atlantic regression, measuring various demographic and economic variables (adapted from the book An Introduction to Statistical Learning). (You don't have to do anything yet.)

```
wage.df = read.csv("http://www.stat.cmu.edu/~ryantibs/statcomp-F19/data/wage.csv",
                   skip=16, stringsAsFactors = TRUE)
dim(wage.df)
```

```
## [1] 3000   11
```

```
head(wage.df, 5)
```

```
##        year age     sex           maritl     race       education
## 231655 2006  18 1. Male 1. Never Married 1. White    1. < HS Grad
## 86582  2004  24 1. Male 1. Never Married 1. White 4. College Grad
## 161300 2003  45 1. Male       2. Married 1. White 3. Some College
## 155159 2003  43 1. Male       2. Married 3. Asian 4. College Grad
## 11443  2005  50 1. Male      4. Divorced 1. White      2. HS Grad
##                      region       jobclass         health health_ins      wage
## 231655 2. Middle Atlantic  1. Industrial      1. <=Good      2. No  75.04315
## 86582  2. Middle Atlantic 2. Information 2. >=Very Good      2. No  70.47602
## 161300 2. Middle Atlantic  1. Industrial      1. <=Good     1. Yes 130.98218
## 155159 2. Middle Atlantic 2. Information 2. >=Very Good     1. Yes 154.68529
## 11443  2. Middle Atlantic 2. Information      1. <=Good     1. Yes  75.04315
```

- **3a.** Randomly split the wage data frame into training and test sets of roughly equal size. Report how many observations ended up in each half.

```
inds = sample(rep(0:1, length=3000))
wage.df$train <- inds
sum(inds)
```

```
## [1] 1500
```

1500 observations ended up in each half.

- **3b.** On the training set, fit the following two models. The first is a linear model of `wage` on `year`, `age`, and `education`. The second is an additive model of `wage` on `year`, `s(age)`, and `education`, using the

`gam` package. For the second model, plot the effects fit to each predictor, with `plot()`. Then, use each of these two models to make predictions on the test set, and report the associated test errors. Which model predicts better? What does that tell you about the nonlinearity that we allowed in the additive model for the `age` predictor?

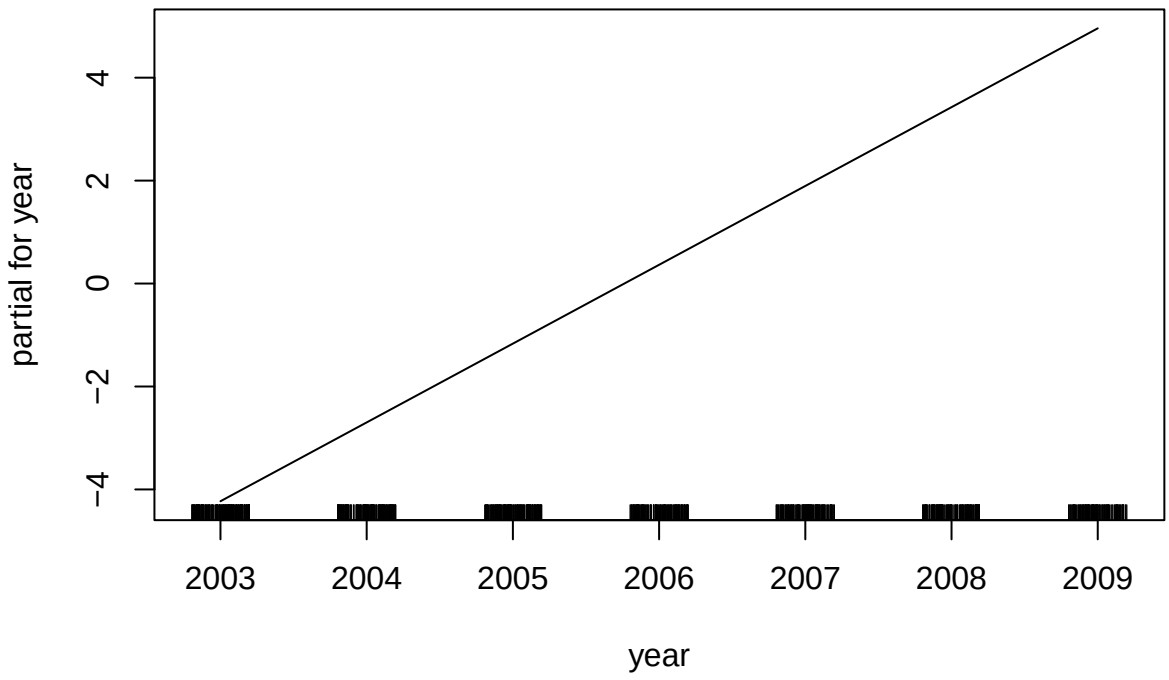
```
library(gam)
```

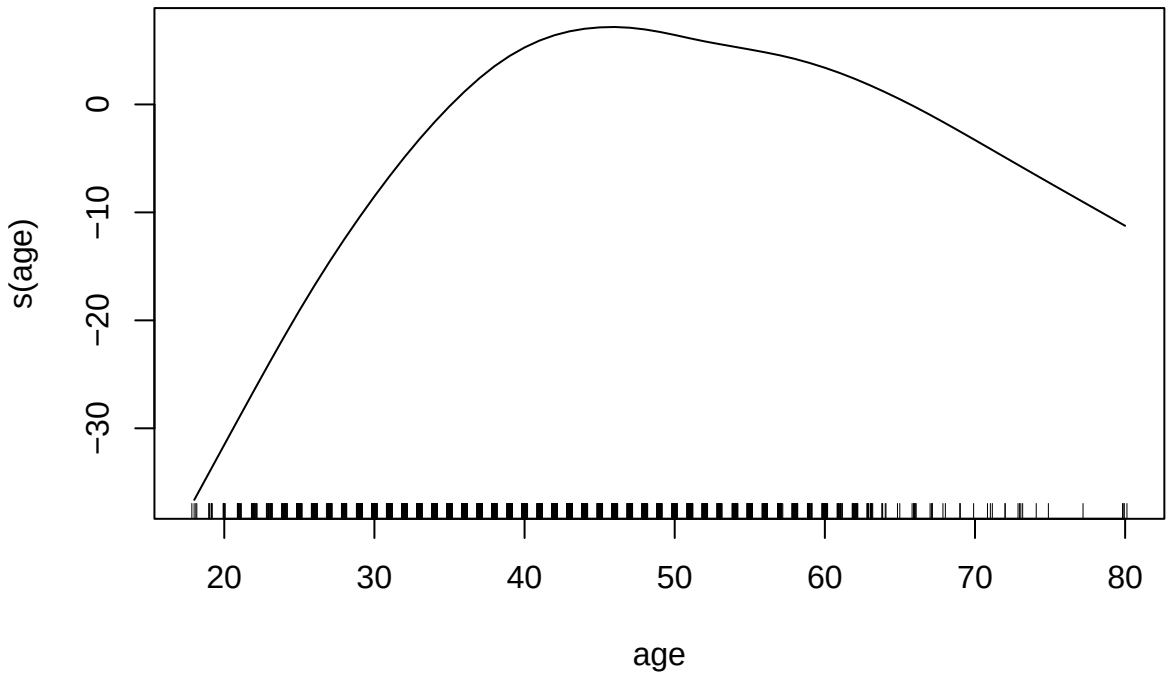
```
## Loading required package: splines
```

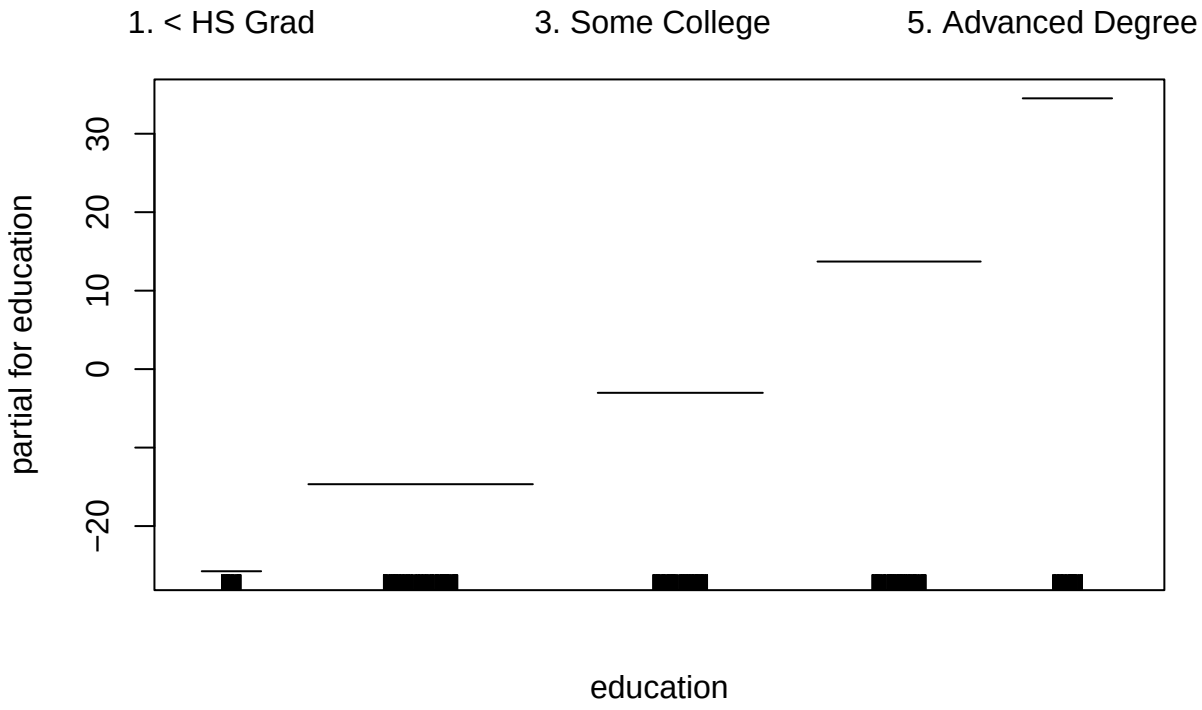
```
## Loading required package: foreach
```

```
## Loaded gam 1.20
```

```
wage.lm <- lm(wage ~ year + age + education, data=wage.df[wage.df$train==1,])
wage.gam <- gam(wage ~ year + s(age) + education, data=wage.df[wage.df$train==1,])
plot(wage.gam)
```


partial for year
4
2
0
−2
−4
2003
2004
2005
2006
2007
2008
2009
year


s(age)
0
−10
−20
−30
20
30
40
50
60
70
80
age




```
lm_preds <- predict(wage.lm, wage.df[wage.df$train==0,])
gam_preds <- predict(wage.gam, wage.df[wage.df$train==0,])
lm_mse <- mean((lm_preds-wage.df[wage.df$train==0,]$wage)^2)
gam_mse <- mean((gam_preds-wage.df[wage.df$train==0,]$wage)^2)
lm_mse
```

```
## [1] 1361.808
```

```
gam_mse
```

```
## [1] 1309.025
```

The GAM has slightly lower test MSE. This indicates that the nonlinearity allowed by the GAM more accurately characterizes the age effects than the linear model.

- **Challenge.** Sample-splitting can be done for logistic regression too, but it just requires us to be a bit more careful about how we choose the training and testing sets. In particular, we want to ensure that the ratio of 0s to 1s (assuming without a loss of generality that the response variable takes these two values) ends up being roughly equal in each of the training and testing sets. For the current wage data set, consider as the response variable the indicator that `wage` is above 250 (recall, this is measured in thousands of dollars!). Discard for the moment all observations that correspond to an education level of less than HS graduate (recall, the reason for this important step was explained in Lab 10f.) Then split the remaining observations into training and testing sets, but do so in a way that maintains equal ratios of 0s to 1s in the two sets, as best as possible. Once you have done this, fit two models on the training set. The first is a logistic model of `I(wage>250)` on `year`, `age`, and `education`. The second is an additive logistic model of `I(wage>250)` on `year`, `s(age)`, and `education`.

  Now, on the test set, use each of the two models to predict the probabilities that `wage` is above 250 for each of the test points. From these probabilities, produce the predicted outcomes—0s and 1s—according

to the following rule: 0 if the probability is below π, and 1 if the probability is above π, where π is the observed proportion of 1s in the training set. From these predicted outcomes, compute and compare test errors. Note that test error, here, is just an average of the number of times we would have predicted the response (`wage` above or below 250) incorrectly. What does this tell you about the nonlinearity allowed in the second model for `age`?

```
set.seed(0)
wage.df <- wage.df[wage.df$education!="1. < HS Grad",]
inds = sample(rep(0:1, length=2732))
wage.df$train <- inds
wage.df$high_wage <- (wage.df$wage>250)

wage.glm <- glm(high_wage ~ year + age + education, family="binomial",
                data=wage.df[wage.df$train==1,])
wage.gam <- gam(high_wage ~ year + s(age) + education, family="binomial",
                data=wage.df[wage.df$train==1,])

p_obs <- sum(wage.df[wage.df$train==1,]$high_wage) / length(wage.df[wage.df$train==1,]$high_wage)

glm_preds <- predict(wage.glm, wage.df[wage.df$train==0,])
glm_preds <- exp(glm_preds) / (1+exp(glm_preds))
glm_preds <- (glm_preds > p_obs)
gam_preds <- predict(wage.gam, wage.df[wage.df$train==0,])
gam_preds <- exp(gam_preds) / (1+exp(gam_preds))
gam_preds <- (gam_preds > p_obs)
mean((glm_preds-wage.df[wage.df$train==0, "high_wage"])^2)
```

```
## [1] 0.2196193
```

```
mean((gam_preds-wage.df[wage.df$train==0, "high_wage"])^2)
```

```
## [1] 0.261347
```

For these models, letting age have a nonlinear effect actually increased the test error. This means that it's better to have a linear age variable when predicting the likelihood of high wages after removing the group of observations with little education.

Cross-validation with the prostate cancer data

- **4a.** Let's revisit the prostate cancer data. Randomly split the prostate cancer data frame into $k = 5$ folds of roughly equal size. (Make sure your code is general enough to handle an arbitrary number of folds k; you will be asked to change the number of folds in questions that follow.) Report the number of observations that fall in each fold.

```
pros.df = read.table(
  "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data")

k = 5
set.seed(0)
inds = sample(rep(1:k, length=97))
pros.df$fold <- inds
table(pros.df$fold)
```

```
##
##  1  2  3  4  5
```

```
## 20 20 19 19 19
```

- **4b.** Over the folds you computed in the previous question, compute the cross-validation error of the linear model that regresses `lpsa` on `lcavol` and `lweight`.

```
error_matrix = matrix(0, 5, 1)
for(i in 1:5){
  model <- lm(lpsa~lcavol+lweight, data=pros.df[pros.df$fold!=i,])
  preds <- predict(model, pros.df[pros.df$fold==i,])
  error <- mean((pros.df[pros.df$fold==i,]$lpsa-preds)^2)
  error_matrix[i,] <- error
}
mean(error_matrix)
```

```
## [1] 0.6076543
```

- **4c.** Write a function `pros.cv()`, which takes three arguments: `df`, a data frame of prostate cancer measurements, with a default of `pros.df`; `k`, an integer determining the number of cross-validation folds, with a default of 5; and `seed`, an integer to be passed to `set.seed()` before defining the folds, with a default of NULL (meaning no seed shall be set). Your function should split up the given data `df` into `k` folds of roughly equal size, and using these folds, compute the cross-validation error of the linear model that regresses `lpsa` on `lcavol` and `lweight`. Its output should simply be this cross-validation error.

```
pros.cv <- function(df = pros.df, k = 5, seed = NULL){
  if(!(is.null(seed))){
    set.seed(seed)
  }
  inds = sample(rep(1:k, length=length(pros.df$lpsa)))
  pros.df$fold <- inds
  error_matrix = matrix(0, k, 1)
  for(i in 1:k){
    model <- lm(lpsa~lcavol+lweight, data=pros.df[pros.df$fold!=i,])
    preds <- predict(model, pros.df[pros.df$fold==i,])
    error <- mean((pros.df[pros.df$fold==i,]$lpsa-preds)^2)
    error_matrix[i,] <- error
  }
  return(mean(error_matrix))
}
```

- **4d.** Investigate the result of `pros.cv()` for different values of `k`, specifically, for `k` equal to 2, 5, 10, and 97. For each value, run `pros.cv()` some large number of times (say, 50) and report the average of the cross-validation error estimates, and the standard deviation of these estimates. Then, plot them in an informative way (say, a box plot with `boxplot()`). What do you notice? Is this surprising?

```
set.seed(1000)
vals <- c(2,5,10,97)
error_matrix = matrix(0, 50, length(vals))
for(i in 1:length(vals)){
  for(j in 1:50){
    error_matrix[j,i] <- pros.cv(i)
  }
}
colMeans(error_matrix)
```

```
## [1] 0.5789342 0.5789932 0.5715051 0.5728769
```

```
apply(error_matrix,2,sd)
```

```
## [1] 0.01687522 0.01797943 0.01702870 0.01679166
```

Results vary depending on the seed, but in general there appears to be a bit of a bias-variance tradeoff with selection a certain number of folds. In particular, increasing the number of folds seems to decrease the test error of the estimator, but it also increases the standard deviation of the estimates. This isn't particularly surprising because models that use one fold will be very overfit, and models that use many folds will be underfit, and both of these extremes will have a high variance in their estimates.

- **Challenge.** In general, is 2-fold cross-validation the same as sample-splitting? Why or why not?

It's a little bit difference because when you use two folds, you will still train your model on both folds and evaluate the test error on both folds, whereas with sample splitting you will strictly use one fold for training and one fold for evaluation.

- **Challenge.** In general, what can you say about the differences in cross-validation as the number of folds varies? What is different about cross-validation with 2 folds, versus 5 folds, versus n folds (with n being the number of observations in the data set)?

Increasing the number of folds will decrease the likelihood of overfitting your model but also increase the variance of your estimates after a certain point.

- **Challenge.** Modify your function `pros.cv()` so that it takes another argument: `formula.str`, a string in the format of a formula specifying which linear model is to be evaluated by cross-validation, with the default being "lpsa ~ lcavol + lweight". Demonstrate the use of your function for different formulas, i.e., different linear regression models.

Making predictions with the HIV data set (optional)

Below, we read in some data on HIV from Rhee et al. (2003), "Human immunodeficiency virus reverse transcriptase and protease sequence database". There are 1073 observations of the following nature. The response variable (first column) is a measure of drug resistance, for a particular HIV drug. The 241 predictor variables (all but first column) are each binary indicators of the presence/absence of mutation at a particular gene mutation site. The goal is to predict HIV drug resistance from this genetic mutation information. (You don't have to do anything yet.)

```
hiv.df = read.table("http://www.stat.cmu.edu/~ryantibs/statcomp-F19/data/hiv.dat")
dim(hiv.df)
```

```
## [1] 1073  241
```

```
hiv.df[1:5, c(1,sample(2:ncol(hiv.df),8))]
```

```
##           y p162 p143 p163 p9 p79 p88 p119 p61
## 1 14.612804    1    0    0  0   0   0    0   0
## 2 25.527251    0    0    0  0   0   0    0   0
## 3  0.000000    0    0    0  0   0   0    0   0
## 4  7.918125    0    0    0  0   0   0    0   0
## 5 11.394335    0    0    0  0   0   0    0   0
```

- **5a.** Use 5-fold cross-validation to estimate the test error of a linear model that regresses the drug resistance on all of the genetic mutation indicators. (You will likely get some warnings about the linear model encountering a rank-deficient fit: why do these occur?)
- **5b.** Use 5-fold cross-validation to estimate the test error of a regression tree with the drug resistance measure as response and the genetic mutation indicators as predictors. To fit a regression tree, you can

use the function `rpart()` from the package `rpart`. (Its notation is similar to `lm()` both for training and prediction.) In terms of prediction accuracy, does the regression tree improve on the linear model?

- **5c.** Use 5-fold cross-validation to estimate the test error of a gradient boosted machine with the drug resistance measure as response and the genetic mutation indicators as predictors. To fit a gradient boosted machine, you can use the function `xgboost()` from the package `xgboost`. (This might require a bit of tinkering to set up; if you'd like a concrete place to start with the boosting settings, then you can try `max.depth=20`, `nround=10`.) In terms of prediction acccuracy, how does boosting fare, compare to a single tree?
- **5d.** Implement your own function for k-nearest neighbors regression. Then, run 5-fold cross-validation to estimate test error, for a few choices of k. Discuss your findings in comparison to those for the linear model, regression tree, and boosting.