STATISTICS 452/652: Statistical Learning and Prediction

September 23, 2020

Lecture 3: Evaluating Models—Measuring Model Error

(Reading: ISLR 2.1-2.2.2, Chapter 5)

1 Goals of lecture

- Much of Statistical Learning (SL) is about "finding the best model"
- We have to have a workable definition of "best"
  - What ideal in the population are we trying to optimize?
- We have to be able to measure that ideal with a statistic
- This turns out to be tricky
- We need to learn some clever solutions

2 Measuring Error on a Fitted Model

- The ultimate goal in SL is to predict *future* (or other unknown) Y for any possible X
  - Occasionally, we know the X values, but usually we don't, so future X will be sampled at random as well
- No way to guess future δ values, so best we can do is predict $g(\mathbb{X})$
- We have only one sample of size n
  - In some applications (e.g., internet traffic, weather), collection is continuous
  - But analysis has to happen at some fixed time

- Data in sample consists of pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ where each x has p variables in it.
- Proposed model, $f(X)$, has
  - Some partially known structure (shape)
  - Some unknown elements (parameter values)
- We fit $f(X)$ to these data, optimizing some criterion to estimate unknown elements
  - Almost always do this by minimizing SUM OF SQUARED ERRORS (SSE) via the least squares (LS) criterion,
$$\sum_{i=1}^{n}(y_i - f(x_i))^2$$
  - Fitted model can be denoted as $\hat{f}(X)$
  - SAMPLE MEAN SQUARED ERROR is the average squared distance between sample and prediction function
$$sMSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$
- LS guarantees that the version of the model we select has best SSE and sMSE for the sample

Multiple Models

- Often, we don't know what model to fit and consider several options
  - $f_1, f_2, \ldots, f_M$ may be a "family" of models of a certain type with different structures, or just different models
  - For example, polynomial regression models of "order d" are models of the form
$$f(X) = \beta_0 + \beta_1 X + \ldots + \beta_d X^d$$
    - Order "0": $f(X) = \beta_0$, a constant for all X
    - Order 1: $f(X) = \beta_0 + \beta_1 X$
    - Order 2: $f(X) = \beta_0 + \beta_1 X + \beta_2 X^2$
    - Order 3: $f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$
    - etc.
    - Can consider models up to order $M < n$
  - We can fit each one using LS
  - How do we know which model is best for making future predictions?
    - Like, how do we know which order, d, to use?
- *We need to be able to measure the average error from each model when we use it for future prediction.*

2.1 What we'd like to do

- sMSE measures how far, on average, the prediction function is from *the sample.*
- This isn't what we want. Our real goal is to find the best model for *future* predictions
  - Imagine drawing a new GIGANTIC (or infinite) new sample of X values, representing the "future"
  - Model should accurately predict Y for *these* X
    - NOT JUST THE ONES IN THE ORIGINAL SAMPLE
      - Best looking model in original sample may chase its errors
      - New Y values will have different errors δ than those
  - Call new data $(x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_{n^*}^*, y_{n^*}^*)$
  - Compute the MEAN SQUARED PREDICTION ERROR (MSPE) on the new data

$$MSPE = \frac{1}{n^*}\sum_{i=1}^{n^*}(y_i^* - \hat{f}(x_i^*))^2 \tag{1}$$

    where $\hat{f}$ is the fitted model from the *original* data
- Important point: the data that created $\hat{f}$ and the new data are *independent*
  - Knowing everything about the original sample tells you nothing about the possible δ errors in the new sample.

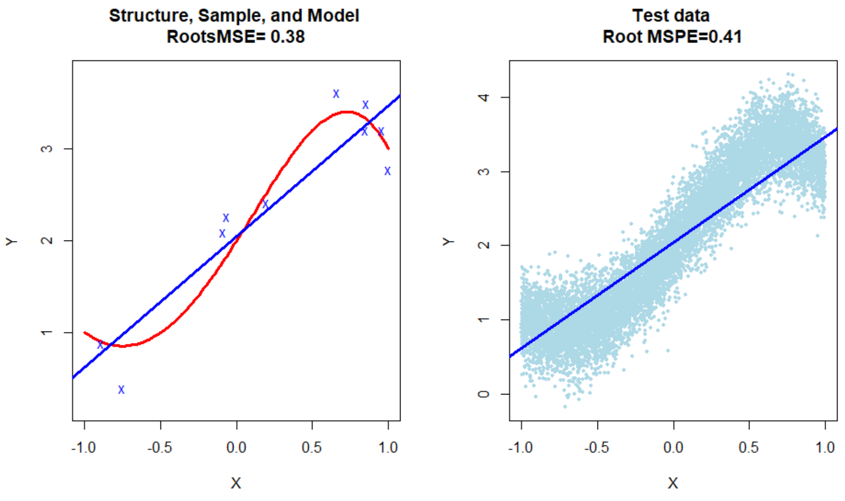
Example: Prediction Error: The goal (L3 - Measuring Error.R) In this example, recreate what is not possible in real life.

1. We start with the original sample of $n = 10$ observations from the same structure we used in the last lecture, a 4th-order polynomial that actually looks kind of like a third-order.
2. We fit the simple linear regression model to the sample.
3. We generate a "gigantic" independent set of data, not used for fitting the model
   (a) I used 10,000, but could have used a million, a billion, or whatever.
4. We see what "future errors" will look like.
5. We measure the sample mean squared error, sMSE, and the mean squared prediction error (MSPE) for the sample from the future

See Figure 1. What we see is:

- First of all, we notice that if we could have seen more data in the first place, we'd never have considered the straight line!

Figure 1: Sample vs. test error. Left panel: true structure (red curve), sample of $n = 10$ observations (blue "X"), and estimated linear regression (blue line). Right panel: "large" sample of new data from "future", showing how we want to evaluate the fitted regression.



- More specifically, we see how variable the data would look at any given value of X
  - This is the impact of the unknowable δ's.
  - Also points to why we don't necessarily want to just get close to the sample responses
  - We want to be able to predict the middle of this mess at each X
- The $\sqrt{sMSE}$ for the straight line is 0.38[1]
- The square root of the average prediction error from the new 10,000 draws is 0.41, pretty similar
- By the way, I know from the simulation that the square-root-MSPE for the right model should be 0.3, so our model imperfection is adding another 0.11 to this.

- The REAL criterion we want to minimize is the aggregate squared error *over the entire rest of the population*, not the sample
- This is the *Expected Mean Squared Prediction Error*, EMSPE,

$$EMSPE = E_Y[(Y - \hat{Y})^2]$$

  - "E" is the math-stat symbol for "expected value", which is nothing more than the population average with respect to a random variable

[1] We usually report measures of average squared error as square roots, to return the units of measurement to the same units as Y. This is similar to how we compute and report standard deviations from sample variances.

  - "E_Y" signifies that the average is over the population of Y values ("the future")[2]
    - As opposed to the population of possible linear regressions $\hat{Y}$ we could have observed with different original samples.

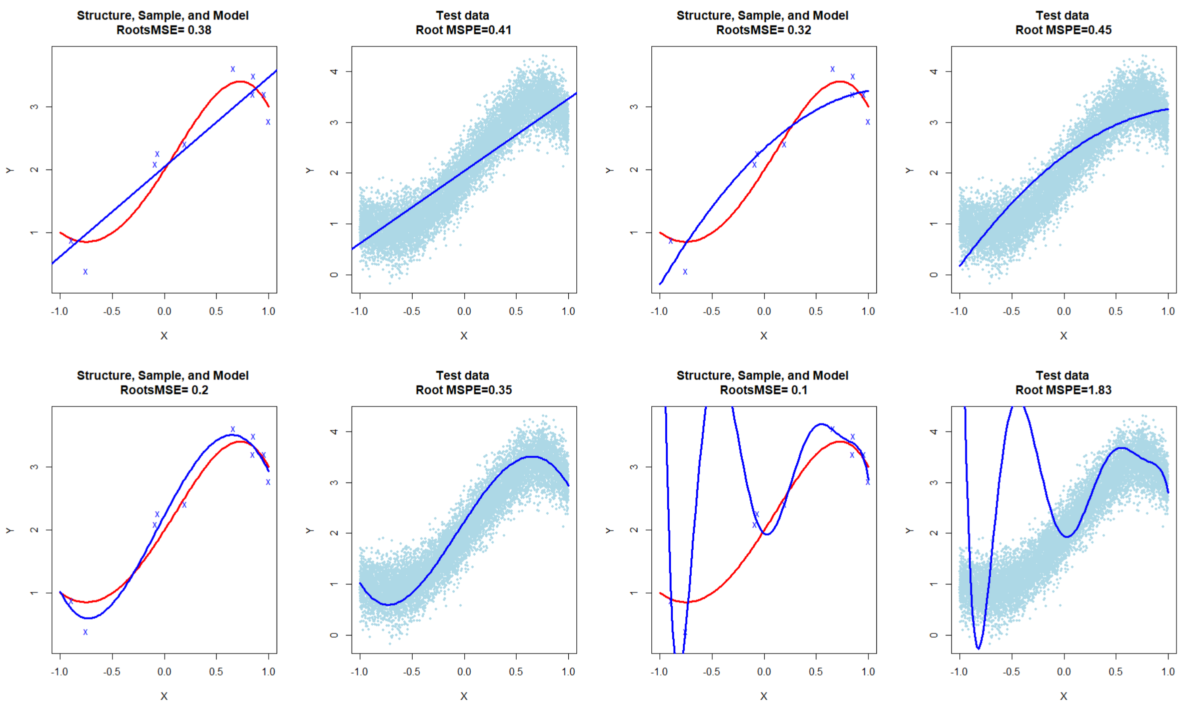
- This is literally just taking same calculation as the MSPE, except over an infinite future sample:
$$\frac{1}{N}\sum_{i=1}^{N}(Y_i^* - \hat{f}(X_i^*))^2 \tag{2}$$
for an ever-growing N observations, (X^*, Y^*)
  - I will sometimes use the notation $\hat{Y}(X) = \hat{f}(X)$ to mean the prediction function created from the original sample, computed at any value X.
- Of course, this is impossible to compute, so we have to use something else.

2.2 Problem with Mean Squared Error

- It is tempting to use the sMSE to estimate 2.
- This is a bad idea!
  - EMSPE assumes that errors are measured on different, independent data from sample used to create $\hat{f}$
  - sMSE uses SAME data for both...it matters!
- Recall that errors any model makes consist of combination of bias and variance
  - Too simple/stiff: high bias, low variance
  - Too complex/flexible: low bias/high variance
  - Goal is to find good compromise (bias-variance tradeoff)
- $\hat{f}$ was deliberately chosen to try to make sMSE as small as possible
  - Making f more complex (almost) ALWAYS makes $sMSE$ smaller
    - Overly complex models have sMSE driven to 0
  - Proof:
    - Start with linear regression model $f_1(X) = \beta_0 + \beta_1 X_1$
      - · Estimate parameters by LS
      - · $sMSE_1 = \frac{1}{n}\sum_{i=1}^{n}(y_i - [\hat{\beta}_0 + \hat{\beta}_1 x_{i1}])^2$
      - · This is the minimized LS criterion, divided by n

[2]This formula is oversimplified for clarity. In more proper terms, we would need to add notation to account for the joint distribution of (X, Y), and a conditioning statement that conditions on the sample we have drawn, so that $\hat{Y}$ is treated as fixed rather than random. Don't try to learn math start from this class. Take STAT 330.

Figure 2: Sample vs. test error for 4 different polynomial models: top pair: $d = 1, 2$; bottom pair: $d = 4, 7$.



- $*$ Add any additional term to it: $f_2(X) = \beta'_0 + \beta'_1 X_1 + \beta'_2 X_2$
  - · Estimate parameters by LS
  - · $sMSE_2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - [\hat{\beta}'_0 + \hat{\beta}'_1 x_{i1} + \hat{\beta}'_2 x_{i2}])^2$
  - · This is the minimized LS criterion for f_2, divided by n
- $*$ If LS estimate for $\hat{\beta}'_2 = 0$, then we can't improve the LS criterion by adding X_2
  - · The model reduces to f_1 and $sMSE_1 = sMSE_2$
- $*$ If LS decides to estimate $\hat{\beta}'_2$ to be *anything* else, it is because doing so improves the LS criterion over what f_1 can do.
  - · In this case, $sMSE_2 < sMSE_1$
- $*$ So, unless by weird luck, $\hat{\beta}_2$ is EXACTLY 0, the sMSE for the more complex model is smaller

See this for yourself in an example:

Example: Prediction Error: The problem with sample MSE (L3 - Measuring Error.R) Continuing the previous example, we make the models successively more complex by increasing the order of polynomial and watch what happens to the model fits, the sMSE and the approximate EMSPE. See Figure 2.

What we see is:

- Repeating the straight line fit, $\sqrt{sMSE} = 0.38$ and $\sqrt{EMSPE} \approx 0.41$

- Adding just a quadratic term doesn't really improve the fit visually
  - $\sqrt{sMSE} = 0.32$ and $\sqrt{EMSPE} \approx 0.45$
  - The fit to the sample looks much better than it is in the population
- Using the correct structure as the model gives us a decent looking shape
  - $\sqrt{sMSE} = 0.20$ and $\sqrt{EMSPE} \approx 0.35$
  - The *sMSE badly* underestimates the average prediction error relative to the future population
- Using an overly complex model chases errors like crazy!
  - $\sqrt{sMSE} = 0.10$ and $\sqrt{EMSPE} \approx 1.83$
  - This is an absolutely horrible fit, but the sMSE thinks it's just great!

- Thus, sMSE does not respect the bias-variance tradeoff!
  - But we don't get to generate new data just to measure EMSPE
  - Need to find ways to use just the sample to do this

3 Resampling (Sample Re-use) Methods

Some terms:

- A typical SL analysis consists of four steps:
  1. Identify some group of models as candidates
     (a) Goal is to find the one that fits best
  2. Fit the models on some data
     (a) This is called the TRAINING phase
     (b) Data used here are called TRAINING DATA or the TRAINING SET
  3. Compare the models on a new, independent set of data to see which one generalizes best to the population
     (a) This is called the VALIDATION phase
     (b) Data used for this are called VALIDATION DATA or the VALIDATION SET
     (c) The MSPE is computed as in equation 1 using the validation data as (x^*, y^*)
     (d) Find the best model according to MSPE
  4. Use a third, independent data set to estimate and report a measure of error for the final selected model
     (a) This is called the TEST phase.

(b) Data used here are called TEST DATA or TEST SET

(c) The MSPE is computed as in equation 1 using the test data as (x^*, y^*)

- It is vitally important that data used at any phase are not seen by the models in any previous phase
  - Its sample of δ errors must be a surprise to any fitted models,
  - Otherwise, if the data help in a previous phase, the models will have a chance to chase them
    - This is called ERROR LEAKAGE
    - It will result in underestimates of the error, particularly for more complex models
- How do we create three sets of data when we have only one?

3.1 Data Splitting

- The simplest solution is called DATA SPLITTING
  - Simply *randomly* divide the one sample into three subsamples
  - Use one for training, one for validation, and one for testing
  - Randomizing prevents biases in creating samples
    - If data are collected in any sequence, early data may be different from later data in unknown ways.
- This is used a lot, but there are obvious problems with it:
  - Smaller training set means higher variance in fitted models, especially more complex ones.
  - Maintaining large training set means potentially small validation and/or test sets
    - Remember that MSPE is a statistic and has its own variability and standard error
    - If computed on too few observations, MSPE is too variable to be useful
- Typical recommendation: 70%/15%/15%
  - But it depends on complexity of models and sample size
  - If models are all relatively simple and sample is large, can make training set smaller fraction (e.g., 40/30/30)
  - If models are complex and/or sample size is small, need to keep training set pretty large.
- Good idea to repeat splitting/MSPE calculation
  - Average results from each split
  - Reduces variability of MSPE calculation
  - Allows maintaining larger training sets.

Example: Data Splitting on the Prostate Data (L3 - Prostate Data Resampling.R) In this example, we examine how data splitting can be used to measure prediction error for three models on the prostate data.

- Models use `lcavol`, `pgg45`, or both variables in linear regression
- First we do a single 70/15/15 split
  - Train all three models on 70%
  - Measure MSPE for each model on middle 15%
  - Choose model with best MSPE
  - Compute final test error on last 15%

```
> n = nrow(prostate)
> #store sample size for easy calculations later
>
> # Set seed to get repeatable results.
> #  Rerun without resetting seed to see variability
> set.seed(120401002)
>
> # Set sampling fraction.  Here I choose 70/15/15 percents
> sf1 = 0.7
> sf2 = 0.15
> #  Logic:
> #    1. randomly permute (reorder) numbers 1,2,...,n
> #    2. take positions of first 70% as training set "set=1"
> #    3. take positions of next 15% as validation set "set=2"
> #    4. take positions of last 15% as test set "set=3
>
> reorder = sample.int(n=n, size=n, replace=FALSE)
> reorder
 [1] 56 34 20 14 36 17 70 44 59 68 52 97 64 53 75  4 86 26
[19]  5 90 85 50 27 15 31 63 48 96 11 24 54 65 91 38 49 93
[37] 89  1 95 39 13 12 81 25 45  2 83 10 87  9 16 92 42 51
[55] 66 78 47 29 33 72 37 88 28 18 46 60 94 71 21 55 30 58
[73]  6 22 84 82  7  8 62 61 41  3 77 73 35 57 23 80 67 69
[91] 43 40 32 74 19 76 79
>
> set = ifelse(test=(reorder < sf1*n), yes=1,
+    no=ifelse(test=(sf1*n < reorder & reorder < (sf1+sf2)*n),
+    yes=2, no=3))
> set
 [1] 1 1 1 1 1 1 2 1 1 2 1 3 1 1 2 1 3 1 1 3 3 1 1 1 1 1 1 3
[29] 1 1 1 1 3 1 1 3 3 1 3 1 1 1 2 1 1 1 3 1 3 1 1 3 1 1 1 2
[57] 1 1 1 2 1 3 1 1 1 1 3 2 1 1 1 1 1 1 3 2 1 1 1 1 1 1 2 2
```

```
[85] 1 1 1 2 1 2 1 1 1 2 1 2 2
>
> mod.vol.s1 = lm(lpsa ~ lcavol, data=prostate[set==1,])
> mod.pgg.s1 = lm(lpsa ~ pgg45, data=prostate[set==1,])
> mod2.s1 =  lm(lpsa ~ lcavol + pgg45, data=prostate[set==1,])
>
> pred.v.s1 = predict(mod.vol.s1, newdata=prostate[set==2,])
> pred.p.s1 = predict(mod.pgg.s1, newdata=prostate[set==2,])
> pred.2.s1 = predict(mod2.s1, newdata=prostate[set==2,])
>
> (MSPE.v.s1 = mean((prostate[set==2,"lpsa"] - pred.v.s1)^2))
[1] 0.8918803
> (MSPE.p.s1 = mean((prostate[set==2,"lpsa"] - pred.p.s1)^2))
[1] 1.882894
> (MSPE.2.s1 = mean((prostate[set==2,"lpsa"] - pred.2.s1)^2))
[1] 0.8234661
>
> #  Best model is 2-variable model, so compute test error
> #   there using all other data
> mod2.s2 =  lm(lpsa ~ lcavol + pgg45, data=prostate[set<3,])
> pred.2.s2 = predict(mod2.s2, newdata=prostate[set==3,])
> (MSPE.2.s2 = mean((prostate[set==3,"lpsa"] - pred.2.s2)^2))
[1] 0.3800876
```

We find that the validation error for the 2-variable model is slightly smaller than for the model with just `lcavol`—0.82 vs. 0.89—while both models have much smaller MSPE than the one with `pgg45`, 1.88. When I recompute the error for the best model on the test set, the result is quite different, 0.38. This seems strange, because our expectation is that error measured on an independent set of data should on average be *larger* than when it was chosen as the minimum value from a group of models. However, our test set is tiny—only 15 observations. There is lots of variability in MSPE from split to split with such small validation and test sets, as we will see next.

To see this, we repeat the first part of the analysis by splitting 100 times. This give us 100 estimates of error for each of the three models. We can take averages to get more precise estimates of error, plus we can put measures of uncertainty on these estimates.

This is not tricky; the code is shown in the program for this example. The results are below, where `MSPEs` is a 100×3 matrix for which each row is from one split and each column is from one model. We compute the mean and a 95% confidence interval for the MSPE from each model.

```
> head(MSPEs)
      lcavol-s  pgg45-s    both-s
[1,] 0.7169422 1.087899 0.6701023
```

```
[2,] 0.6393267 1.174864 0.5986930
[3,] 0.5473014 1.250206 0.5020446
[4,] 0.9031222 1.882240 0.9235737
[5,] 0.5731554 1.123305 0.6683715
[6,] 0.4113682 1.116847 0.4813640
>
> (MSPE.mean = apply(X=MSPEs, MARGIN=2, FUN=mean))
 lcavol-s   pgg45-s    both-s
0.6994601 1.3661100 0.7164228
> (MSPE.sd = apply(X=MSPEs, MARGIN=2, FUN=sd))
 lcavol-s   pgg45-s    both-s
0.1779951 0.3547278 0.1740420
> MSPE.CIl = MSPE.mean - qt(p=.975, df=R-1)*MSPE.sd/sqrt(R)
> MSPE.CIu = MSPE.mean + qt(p=.975, df=R-1)*MSPE.sd/sqrt(R)
> round(cbind(MSPE.CIl, MSPE.CIu),2)
         MSPE.CIl MSPE.CIu
lcavol-s     0.66     0.73
pgg45-s      1.30     1.44
both-s       0.68     0.75
```

1. First, notice how much all three models' MSPE varies from split to split. We get the feeling here that a single split is really just guessing at the true EMSPE within a pretty wide range.

2. Despite this the three models show a consistent pattern in which `pgg45` is clearly worse than the other two. The other two take tuens ourperforming each other. Each is best in 3 splits among the 6 shown. There is not a clear pattern of preference.

3. The means show this same pattern

4. The confidence intervals show that our estimates of error are around $\pm 5\%$ of the means, so the means are reasonably precisely estimated.

   (a) The two good models have heavily overlapping intervals. There is no clear favourite between them. I would probably choose the smaller model because it is easier to work with for the future—only one variable to measure.

   (b) The third model is clearly much worse than the others. The CI is WAY higher.

3.2 Cross-Validation

- Cross-validation (CV) is just an organized way to do repeated data splitting

- **CV Algorithm**:
  1. Break the data up randomly into V roughly equal subsets, called "folds" (for example, 10-fold cross-validation is commonly used, meaning that $V = 10$)
  2. Delete one fold (like doing a $V - 1 : 1$ split)
     (a) Develop the model on the remaining data, size $n_1 = n * (V - 1)/V$
     (b) Predict the responses for the deleted fold and compute MSPE, $n^* = n/V$
  3. *Repeat for each fold.*
     (a) Average MSPEs
- Thus, each subset serves as a test set one time, and serves as a part of the training set for predicting each of the $V - 1$ other subsets.
  - *Every* observation gets a prediction and MSPE is computed between the predicted values and the observations using all n observations
  - This is the CV estimate of prediction error, which I might call $MSPE_{CV}$ or CV-error or something.
- Can use CV in validation phase, test phase or both
- If M models are to be compared in validation, requires fitting VM models

1. Cross-validation can give estimates of prediction error that have little bias, but only if used properly.
   - In general, when using CV for estimating prediction error of any complex analysis, using knowledge from any "whole set" analysis is forbidden within the folds.
     - That is, the v^{th} model fit must not have any chance to "see" the test data before they are predicted.
2. CV errors are still quite variable unless n is very large.
   - Can repeat CV R times and average results
   - Now requires RVM model fits
   - CV is now based on Rn squared errors
3. Choice of V is of relatively minor importance
   - Potential values are anything from 2 to n
     - 2-fold CV is just a single 50-50 split, but with fitting and validation done in both directions
     - n-fold CV makes each observation its own fold.
       * Equivalently, leave one observation out and use $n - 1$ to fit model
       * Special case with special name: leave-one-out CV (LOOCV)

- Choice of V is another form of bias-variance tradeoff!
  - Smaller V means that models are fit on smaller fractions of the data
    - Underestimating MSPE from full data
  - Larger V means that models are fit on increasingly overlapping data sets
    - Individual squared errors are more correlated because of model similarity
    - Means of positively correlated RVs have more variability than means of less-correlated RVs.
- Also a matter of computation time
  - Need to fit VM models, so prefer V to be smaller
- On balance, V is almost always chosen to be 5 or 10

Example: Cross-validation on the Prostate Data (L3 - Prostate Data Resampling.R) We repeat the previous example using cross-validation. The key is in how the `sample.int()` function is used to create V groups of approximately equal size rather than two or three as before. The code is in the example.

I did 5-fold CV. The results are below, where `MSPES.cv` are the MSPEs from the 5 folds. Notice that, since we have 5 estimates of error, each on 1/5 of the data, we can use these to make a confidence interval for the EMSPE.

```
> MSPEs.cv
      lcavol-c  pgg45-c    both-c
[1,] 0.5126019 1.230918 0.5547818
[2,] 0.8052891 1.538687 0.7608947
[3,] 0.6216238 1.013638 0.5966328
[4,] 0.4728313 1.064475 0.4266302
[5,] 0.9526411 1.394193 0.9157161
>
> (MSPEcv = apply(X=MSPEs.cv, MARGIN=2, FUN=mean))
 lcavol-c   pgg45-c    both-c
0.6729974 1.2483823 0.6509311
> (MSPEcv.sd = apply(X=MSPEs.cv, MARGIN=2, FUN=sd))
 lcavol-c   pgg45-c    both-c
0.2025673 0.2206636 0.1902019
> MSPEcv.CIl = MSPEcv - qt(p=.975, df=R-1)*MSPEcv.sd/sqrt(V)
> MSPEcv.CIu = MSPEcv + qt(p=.975, df=R-1)*MSPEcv.sd/sqrt(V)
> round(cbind(MSPEcv.CIl, MSPEcv.CIu),2)
         MSPEcv.CIl MSPEcv.CIu
lcavol-c       0.49       0.85
pgg45-c        1.05       1.44
both-c         0.48       0.82
```

The 5 folds show the same sort of pattern that we saw with the 100 splits in the previous example. The program also shows how to repeat the CV calculations. I did this 15 times, because it gave about the same total number of predictions as we had in the previous example. The results are quite similar and not shown here.

3.3 Bootstrap

- Bootstrap is similar tool to CV but based on different theory
  - Main use is estimating sampling distributions
  - Can be adapted to measure test/validation error
- **Bootstrap Algorithm:** Bootstrap "resamples" data *with replacement*
  - Draw one observation at random from original data
  - Copy it into new data set (the "resample"), replace it, and draw again
  - Keep going until there are n observations in new data set
  - Fit model to data in resample
  - Compute MSPE on any data that were never selected into resample
- Observations may be drawn 0, 1, 2, 3,... times, with pure chance determining how many of each observation are in resample[3]
  - On average, about 2/3 of observations get included at least once
  - Remaining observations (random, but about 1/3 on average) are the validation/test sample
  - These are sometimes called "out-of-bag" data
- Model fit based on n observations
  - Multiple copies don't really bother the process!
  - So variability of model is similar to original data
- Like with other methods, can repeat R times

Example: Bootstrapping on the Prostate Data (L3 - Prostate Data Resampling.R) We repeat the previous example using bootstrapping. Again, the key is in how the `sample.int()` function is used to resample with replacement. The code is in the example. I also reran the bootstrap for $R = 40$ resamples, again to create a similar total number of predictions. The results are similar to what we have seen in previous examples and are not shown here.

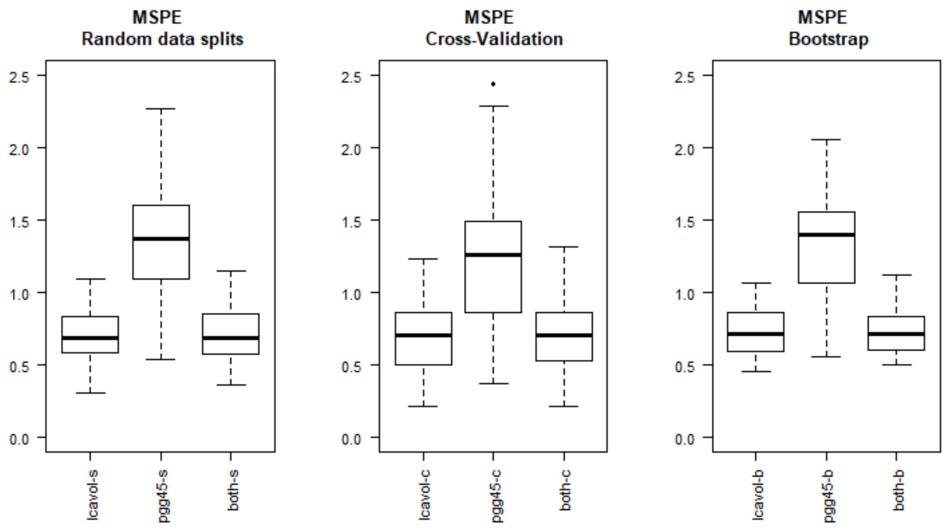
---

[3]Technical detail: Bootstrap resampling is simply an application of the multinomial distribution. You have n identical trials and probability $1/n$ on each of n "categories" (the observation indexes $1, 2, \ldots, n$).

3.4 Using resampling methods to select models

- Suppose we have M candidate models we are considering, $f_1, \dots, f_M$
- We use some appropriate method from above to estimate their true prediction error, EMSPE
  - $MSPE_1, MSPE_2, \dots, MSPE_M$
- Simplest thing to do is select the model with the smallest $MSPE$
  - Call this model $\hat{m}$
  - The "hat" is deliberate, because this is an estimate that could change if a different sample were drawn
- The problem is that we don't have a measure of uncertainty on the $MSPE_m$'s
  - Is $\hat{m}$ is a clear choice or could several models give "similar" results
- Solution: If you can afford computing time, repeat the error computations R times.
  - Fit all M models on each split
  - Average each model's results
  - Compute SE and confidence intervals if $R$is large enough
  - Make boxplots of MSPEs for each method
- **Better way to use these results**
  - In each split, rescale MSPEs *relative to best*
    - In split r, compute $MSPE_{1,r}, MSPE_{2,r}, \dots, MSPE_{M,r}$
    - Suppose that model $\hat{m}_r$ has the smallest MSPE in that split
    - *Divide all MSPEs in that split by the smallest MSPE*
    - Call these the RELATIVE MSPEs for model m in split r, $RMSPE_{m,r}$
    - Smallest RMSPE is 1
    - All other RMSPEs are above 1 according to how much larger they are than the "best"
      - e.g., $RMSPE_{m,r} = 1.1$ means that model m had 10% higher MSPE in split r than best
      - $RMSPE_{m,r} = 2.0$ means that model m had double the MSPE of the best model in split r.
  - *Make a boxplot of the RMSPEs.*
    - This works because some splits give comparably high or low MSPEs for all methods
    - You don't care about this!
    - You only care about which ones are smallest in a given split
- I will do this *constantly* to help understand model performance. **Get used to it!!!!!**

Figure 3: Boxplots of MSPEs from three linear regression models estimated using multiple repetitions of each splitting process. For random splits, $R = 100$, with $n^* = 15$ in each test set. For CV, there $R = 15$ with $n^* = 97$ in each rep. For bootstrap, $R = 40$, with random n^* in each rep, but average is about $n^* = 36$



Example: Model Comparisons on the Prostate Data (L3 - Prostate Data Resampling.R) We show how to use the results of the repeated splits to get a visual impression of model comparisons. We show model comparison result from all three splitting methods. Thus, we have three boxplots so show, with MSPEs from each model. We maintain the same Y-axis in each plot to make them more comparable.

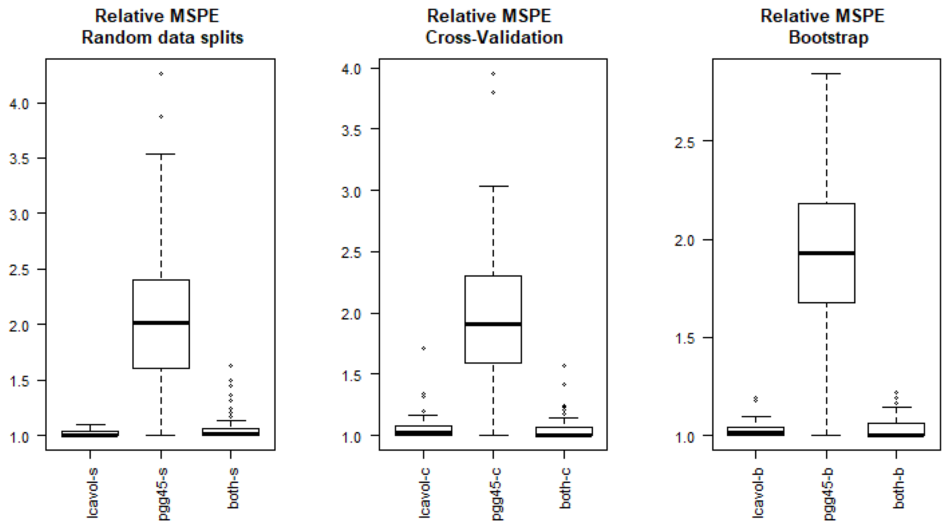
First, Figure 3 gives a clear visual depiction of what we have seen before. Remember that we want *low* values of MSPE.

- There is a lot of variability from one split to the next, regardless of method.
- For all three splitting methods, the `pgg45` model provides a much worse fit than the two models with `lcavol` in them.
- The other two models have boxes that look like they were possibly sampled from the same distribution. There's no clear winner between them.

Next, we compute the *relative* MSPEs by dividing each MSPE by the minimum value among the three models. The code for this is below. The `apply` function tells R to do a summary calculation on rows ("1") or columns ("2") of a matrix. Here we need to get the minimum MSPE (`min()` function) in each row ("1") among the three models. We do this separately for each of the three results matrices, `MSPEs`, `MSPEs.cv15`, and `MSPEs.b40`. Then when we make the boxplot, we divide the matrix by this vector of minimum values. The results are in Figure 4.

```
> par(mfrow=c(1,3))
> low.s = apply(MSPEs, 1, min)
```

Figure 4: Boxplots of *relative* MSPEs from same scenario as in Figure 3.



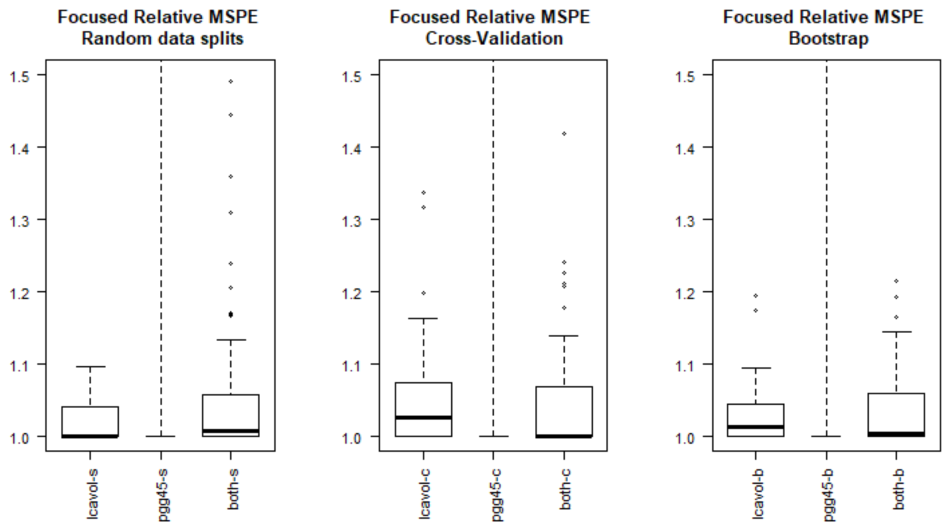
```
> boxplot(MSPEs/low.s, las=2,
+         main="Relative MSPE \n Random data splits")
> low.c = apply(MSPEs.cv15, 1, min)
> boxplot(MSPEs.cv15/low.c, las=2,
+         main="Relative MSPE \n Cross-Validation")
> low.b = apply(MSPEs.b40, 1, min)
> boxplot(MSPEs.b40/low.b, las=2,
+         main="Relative MSPE \n Bootstrap")
```

- In this kind of plot, whenever a particular model "wins" a split, its value is 1.
  - Otherwise, when it loses, it is above 1 by the amount it loses by.
- In this case we see that the first and third models seem to share winning almost all the time.
  - Even when they don't win, they have similar MSPEs to the winner.
- The `pgg45` model is clearly usually much worse than the winner.
  - But it still wins sometimes!!!
  - Imagine if you did only one replicate and had the awful luck to draw a split that made this look like the winner!

In this plot, the middle model is *so* bad, that it is hard to compare the other two—they are all squished done at the bottom. So I find it often helpful to limit the scale of the Y-axis so that we can focus on the good models that we care about. In this case, I chose a range from 1–1.5 to capture all of the good boxes except for a couple of outliers. This is in Figure 5.

It remains hard to choose a winner between the two good models.

Figure 5: Repeat of boxplots from Figure 4, limiting Y-axis to narrower range to highlight comparison of best models.



4 Exercises

Concepts

1. Which aspect of model error is affected by sample size, **bias or variance**?
2. When a model is fit that is too simple (not flexible enough) does this cause **bias or variance** compared to a model with just the right complexity?
3. Suppose we were to drastically increase the sample size used to fit the models in Figure 2. What would happen to the estimated MSPEs of each of the four models? Would they increase a lot, increase a little, not change much, decrease a little, or decrease a lot? **Answer this separately for each of the four polynomial models.**
4. Same question, but **what would happen to the sMSEs?**

Application

Refer to the Air Quality data described previously. With `Ozone` as the response variable, we have three potential explanatory variables, `Temp`, `Wind`, and `Solar.R`. We will invent 5 few models to compare:

- The three separate simple linear regression models
- The linear regression model with all three variables (`formula = Ozone~Temp+Wind+Solar.R`)
- A model that allows curvature and interactions (`formula = Ozone~Temp+Wind+Solar.R+I(Temp^2)+I(Wind^2)+I(Solar.R^2) +Temp*Wind+Temp*Solar.R+Wind*Solar.R`)[4]

We want to know which of these five models is best. (We will learn better techniques for variable selection soon.) Note that `Temp` is easiest to measure and `Solar.R` is hardest. Given a choice, we prefer simpler models that are easy to measure.

1. First, in order to use this set for modeling, we need to deal with its missing data. Since this is not a class on imputation, we deal with them in the easiest but most biased way: case deletion. When you have missing data in real life, *learn how to deal with them better than this!!!* **Run the code below to clean the data and report the results.**

   ```
   AQ = na.omit(airquality[,1:4])
   dim(AQ)
   ```

2. Set the seed to be 4099183 and run a single 75/25 split using `reorder = sample.int(n=n, size=n, replace=FALSE)`, where `n` is the number of rows in `AQ`. (This can be abbreviated as `reorder = sample.int(n)`, since the other arguments are the defaults of the `sample.int()` function.) **Which observations are in the validation set?** Note

[4]If you wish, you can write this using a shortcut, `formula=(Temp+Wind+Solar.R)^2+I(Temp^2)+I(Wind^2)+I(Solar.R^2`

that there is no test set here. We will not try to estimate the final MSPE of the chosen model because the data set is so small. Doing a good job would require nested resampling. We are not there yet...

3. Fit each model on the training data and **report the MSPEs from the validation data**.

   (a) **Which model wins this competition?**

4. Now use 10-fold CV to estimate the MSPEs for the 5 models. **Report the mean and 95% confidence intervals for the 5 models.**

   (a) Identify **which model(s) might be clearly good or clearly bad.**

5. Finally, repeat CV 20 times.

   (a) **Show box plots of MSPE. Comment on what they tell you about the models.**

   (b) **Repeat for RMSPE, and narrow focus if necessary to see best models better. Which model(s) give the best results most often?**

6. Based on what you've done, and considering practical concerns described in the preamble for the problem, **which one model would you suggest using?**