Introduction to Computational BioStatistics with R: resampling

Erik Spence

SciNet HPC Consortium

30 October 2025

SciNet

Today's slides

To find today's slides, go to the "Introduction to Computational BioStatistics with R" page, under Lectures, "Resampling".

`https://scinet.courses/1391`

SciNet

Today's class

Today we will visit the following topics:

- Cross validation.
- Feature selection.
- Bootstrapping.
- Permutation tests.

With material stolen from L. Dursi.

SciNet

How do we choose the correct model?

Let's consider the problem of fitting a polynomial to noisy data.

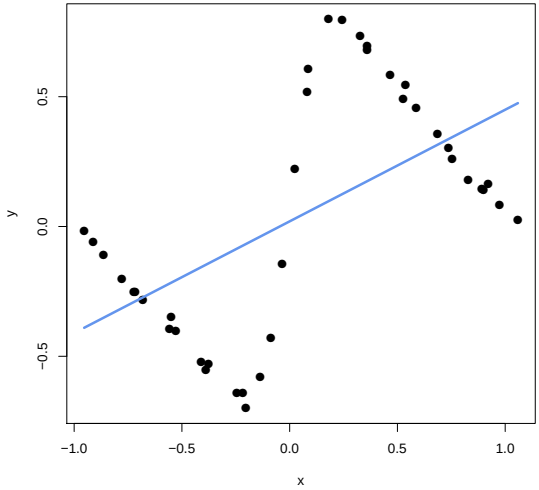
As you are likely aware, we can crank up the order of the polynomial and get a great fit to the data (even perfect!). But this won't do well on out-of-sample data.

So what do we do to choose the correct order of polynomial to fit to our data? How do we choose the correct model for our data?

SciNet

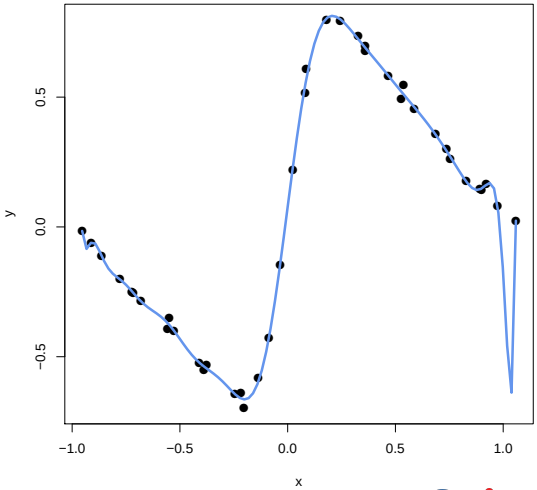
Generate some data, and fit

```
> n <- 40
>
> x <- seq(-1, 1, length = n) + 0.1 * runif(n)
> y <- tanh(8 * x) - x + 0.1 * runif(n)
>
> model <- lm(y ~ poly(x, 1))
>
> plot(x, y, pch = 16, cex = 1.4)
>
> x2 <- seq(min(x), max(x), length = 100)
>
> p.model <- predict(model, data.frame(x = x2))
>
> lines(x2, p.model, lwd = 3,
+       col = 'cornflowerblue')
>
```



Repeat with degree 20

```
>
> model20 <- lm(y ~ poly(x, 20))
>
> plot(x, y, pch = 16, cex = 1.4)
>
> p.model20 <- predict(model20,
+     data.frame(x = x2))
>
> lines(x2, p.model20, lwd = 3,
+       col = 'cornflowerblue')
>
```



It hits almost every point! What a great fit!

Training versus validation

In general, we get our data, and that's it.

- We don't have the luxury of generating more data on a whim.
- We need to do out-of-sample testing of whatever model we generate, to make sure it generalizes well to new data.
- But we often don't have any new data. What to do?
- The solution is to hold out some of the original data when we generate our model.
- Most of the data is used for training the model, the rest is used for validating it.
- These data should be chosen randomly.

It's extremely important to test your model on out-of-sample data.

SciNet

Training versus validation, continued

So we hold out some data, the 'training' data, and build our model.

- Once the model is chosen, then you can train the selected model on the entire training + validation data set.
- But you will probably still want to end your paper with a sentence like "the final model achieved 80% accuracy...".
- This can't be done using the data the model was trained on (train + validation)!
- Any data which has touched the model cannot be used for the final result.
- In this case, another chunk of data must be held out, for final testing.

In the case of training-validation-testing, a common breakdown of the data sizes might be 50%-25%-25% of the initial set. If you don't need a test data set, 2/3-1/3 is common.

SciNet

k-fold Cross Validation

There are some downsides to this approach to validation data hold-out. What if most of the data set's outliers happen to be in the training set?

Ideally, we should do several partitions of the data set, and average over the results. This is called k-fold Cross Validation:

- Partition the data set (randomly) into k sets.
- For each set:
  - Train on the remaining $k - 1$ sets.
  - Validate on the held-out set.
- Average the results, for some measure that gives you a sense of how badly the model is doing (residuals, or accuracy, usually).

The makes efficient use of the data set, and is easily automated.

k-fold Cross Validation, continued

How do we choose k?

- if k is too large - the different training sets are very highly correlated (almost all of their points are the same).
- if k is too small - we don't get very much advantage of averaging in the k validation data sets.

In practice, 10 is a very commonly-used value for k; but again, this depends on the size of your data set.

SciNet

Regression, with degree 20

```
# cross_validation.R
library(caret)

loadData <- function(n) {
  x <- seq(-1, 1, length = n) + 0.1 * runif(n)
  y <- tanh(8 * x) - x + 0.1 * runif(n)
  return(data.frame(x, y)) }

calcError <- function(my.data, d, kfolds = 10) {

  fitControl <- trainControl(method = 'cv',
    number = kfolds)

  f <- as.formula(paste("y ~ poly(x,", d, ")"))
  fit <- train(f, data = my.data, method = "lm",
     trControl = fitControl)

  return(fit$results$RMSE) }
```

```
plotErrors <- function(n, maxdegree = 20) {

  my.data <- loadData(n)

  degrees <- 1:maxdegree
  errors <- rep(0.0, length(degrees))

  for (d in degrees) {
    errors[d] <- calcError(my.data, d)
  }
  plot(degrees, errors)
  lines(degrees, errors)
}
```

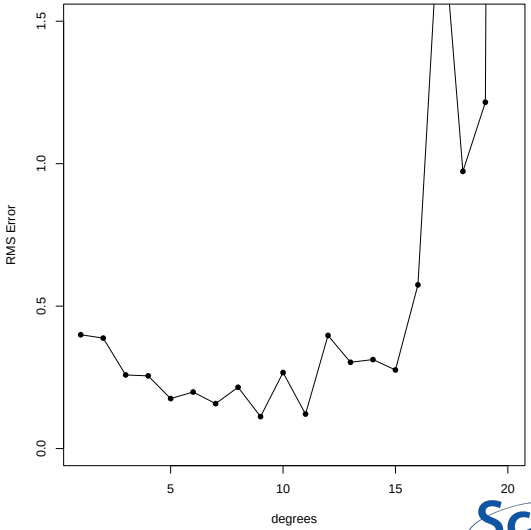
SciNet

Regression, with degree 20, continued

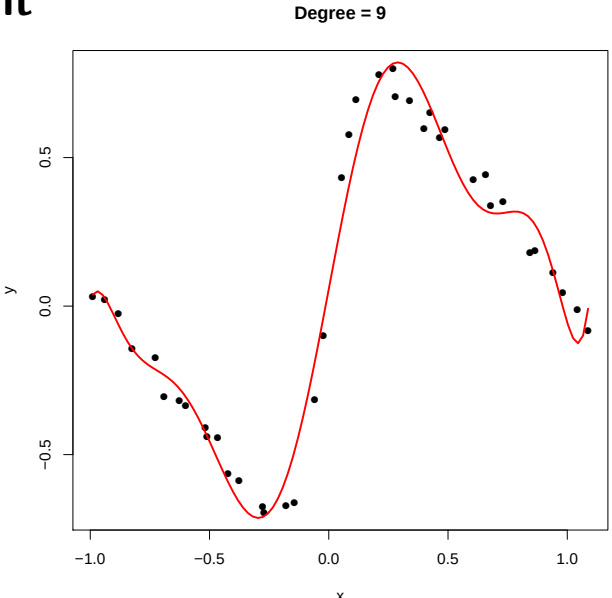
```
>
> source("cross_validation.R")
>
> plotErrors(40)
>
```

This chooses the degree to fit 40 points using 10-fold cross validation.

The error is estimated for each degree; the minimum is chosen. In practise, the simplest model that is "close enough" to the minimum is generally a good choice.



Model with fit



How, maybe, to do feature selection

The number and choice of features in our models is a hyperparameter, just not a numeric one. How might we choose the correct combination of features for our model?

- Cross-validation!

But this is a huge cross-validation (CV) problem:

- Try all $\mathbf{2}^{\boldsymbol{p}}$ possible combinations of features, where $\boldsymbol{p}$ is the number of features.
- Regress on them.
- Of the possibilities, find best one, based on some metric, or combination of metrics.
- This actually has a name: "Best Subset Selection" (or sometimes, Best Subset Regression).
- This will work for modest $\boldsymbol{p}$.
- (But for modest $\boldsymbol{p}$ we don't really need to do much feature selection.)

SciNet

Aside: Danger, Danger!

In general, flags should be going off inside you when facing the possibility of doing tens of thousands of tests on your data.

For feature selection this is ok. But for small variations on this theme things can quickly go horribly, inexorably, off the rails.

- "Let's look through all pairwise combinations of my 100 features, looking for statistically significant correlations!"
- This can come up in cases where it's not necessarily obvious - looking for correlations between pixels in images.
- If you are doing 10,000 hypothesis tests, using as your significance $p < 0.05$, you should expect 500 significant results — even if the data is just random noise.

Huge numbers of tests will naturally result in false positives. Be careful.

SciNet

Forward and backward selection

Exhaustive search is safe, but infeasible in the most urgent cases. ($p = 100$ implies 10^{30} model tests, and 100 isn't a huge number of features.) What other options have we?

Three greedy methods are in common use, but can get caught in local minima:

- Forward selection: starting from nothing,
  - For each remaining feature, include it, and calculate CV error, for some metric.
  - If for the best feature, the error drops enough, select it and continue,
  - Else terminate and report the selected features.
  - R's "step" function can be used for this, as well as the "stepAIC" function from the "MASS" package.
- Backward selection: same, but start with a model with all features, and drop until error rises too much.
  - Use "step" with "direction = 'backward' ".
- Stepwise selection: combination of Forward and Backward selection.

SciNet

Cross-validation and bootstrapping

Cross-validation is closely related to a more fundamental method, bootstrapping.

Let's say you want to find some statistic on some other statistic of your data.

- What is the standard deviation of the 5th quantile of your data?
- What is the mean and standard deviation of an estimation error for a given model?

You'd like new sets of data that you could calculate your statistic on, and then to look at the distribution of that statistic.

SciNet

Non-parametric Bootstrapping

The key insight to the non-parametric bootstrap is that you already have an unbiased description of the process that generated your data - the data itself.

The approach for the non-parametric bootstrap is:

- Generate synthetic data sets from the original data set by resampling;
- Calculate the statistic of interest on these synthetic data sets, and get the distribution of that particular statistic.

Cross-validation is a particular case: CV takes k (sub)samples of the original data set, applied a function (fit the data set to part, calculate error on the remainder), and calculates the mean of the residuals.

Bootstrapping can be used far more generally: any time you need to estimate statistics on a quantity whose statistics aren't automatically calculated.

SciNet

Non-parametric Bootstrapping, example

Suppose you want to get statistics on the median of your data. How would you get the uncertainty on the median?

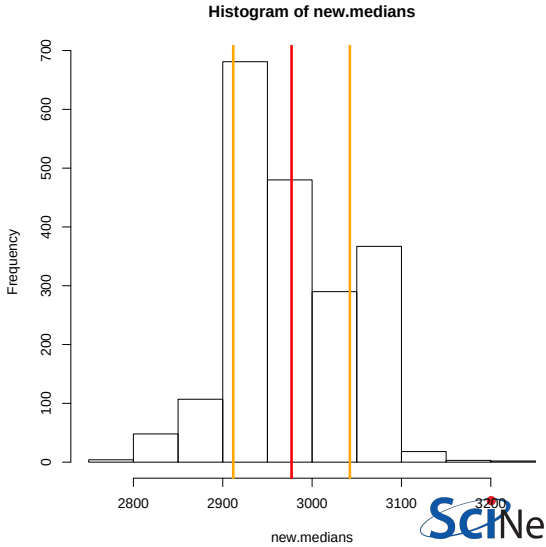
- Randomly sample from your data to create a fake data set.
- Be sure to set "replace = TRUE", so that you are sampling from the full population.
- Do this many times.
- Calculate statistics on the resulting distribution.

```
> library(MASS)
> bwt.median <- function(x, my.data) {
+   new.data <- sample(my.data$bwt,
+       size = nrow(my.data), replace = TRUE)
+   return(median(new.data))
+ }
>
> new.medians <- sapply(1:2000, bwt.median,
+                       birthwt)
>
> median(birthwt$bwt)
[1] 2977
> mean(new.medians)
[1] 2976.945
> sd(new.medians)
[1] 65.2638
>
```

Non-parametric Bootstrapping, example, continued

```
>
> hist(new.medians)
>
> m.medians <- mean(new.medians)
> sd.medians <- sd(new.medians)
>
> abline(v = m.medians, col = 'red', lwd = 3)
> abline(v = m.medians + sd.medians,
+    col = 'orange', lwd = 3)
> abline(v = m.medians - sd.medians,
+    col = 'orange', lwd = 3)
>
```

We can use this distribution to get a confidence interval on the median.



Notes on Bootstrapping

Bootstrapping strengths:

- Allows you to get information on a statistic when the true distribution of the statistic is unknown.

Bootstrapping weaknesses:

- If the statistic of interest is at the edge of parameter space (minimum, maximum, for example) the bootstrapped distribution does not converge to the true distribution.
- If you have too few data points to begin with, bootstrapping will not magically make things better. Your data must be a true representation of the population from which it is drawn.
- If your data's probability distribution has a long tail, or infinite moments, bootstrapping will fail, or give wildly inaccurate results. Examples include the Cauchy distribution, and non-central Student t distribution with 2 degrees of freedom.

SciNet

Don't write your own

As with most things R, there's already a package that does that.

- The 'boot' command in the 'boot' package will run the bootstrap for you.
- You need to specify the function which calculates the statistic.
- The function's 'i' argument is a vector of indices.
- The 'boot.ci' function calculates confidence intervals, using different methods.

```
> library(boot)
> my.med <- function(my.data, i) return(median(my.data[i]))
> b <- boot(data = birthwt$bwt, statistic = my.med,
+           R = 2000)
> boot.ci(b)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 2000 bootstrap replicates
CALL :
boot.ci(boot.out = b)

Intervals :
Level  Normal          Basic
95%    (2843, 3106 )   (2864, 3118 )

Level  Percentile      BCa
95%    (2836, 3090 )   (2807, 3062 )
Calculations and Intervals on Original Scale
```

Parametric Bootstrapping

If you know the form of the distribution that describes your data, you can simulate new data sets:

- Fit the distribution to the data;
- Generate synthetic data sets from the now-known distribution to your heart's content;
- Calculate the statistics on these synthetic data sets, and get their distribution.

This works perfectly well if you know a model that will correctly describe your data; and indeed if you do know that, it would be madness *not* to make use of it in your analysis.

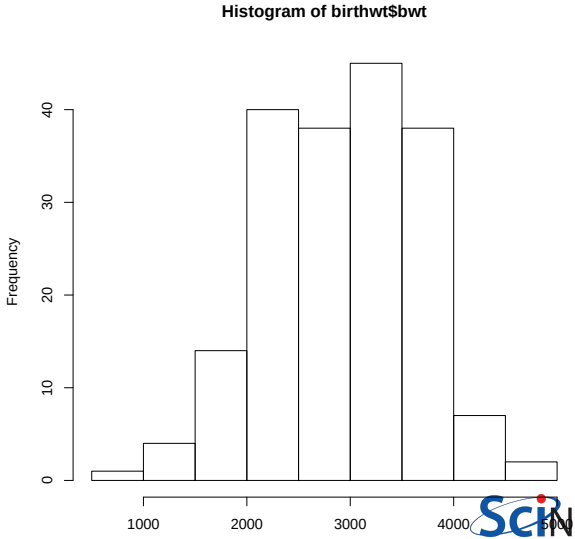
SciNet

Parametric Bootstrapping, example

Suppose we want to do a parametric bootstrap on our data, instead of non-parametric.

The data look pretty Gaussian, let's pretend that we know that the data are Gaussian.

```
>
> hist(birthwt$bwt)
>
```



Parametric bootstrapping, example, continued

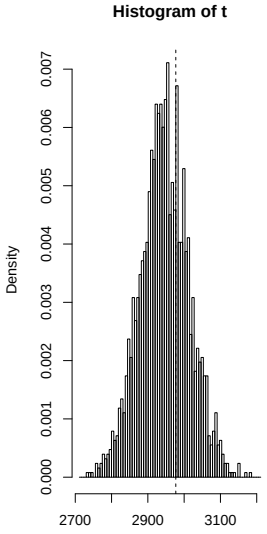
We assume that the data is Gaussian, and proceed as before.

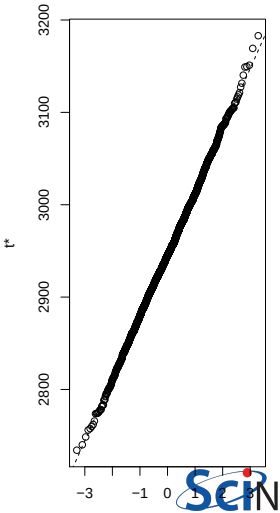
- Create a function which creates new data for you, based on the functional form that you are assuming. Tell boot what function it is.
- Tell boot that you're doing parametric bootstrapping.
- Boot requires that the data be passed to it, even if you don't use it.
- You can use the plot command to plot the results.

```
>
> my.median <- function(my.data)
+                   return(median(my.data))
>
> gen.data <- function(my.data, mle) {
+         return(rnorm(length(my.data),
+                mean = mean(my.data),
+                sd = sd(my.data)))
+ }
>
> b <- boot(birthwt$bwt,
+           statistic = my.median,
+           sim = 'parametric', R = 2000,
+           ran.gen = gen.data)
>
```

Parametric Bootstrapping, example, continued more

```
>
> plot(b)
>
```





Jackknifing

Another resampling technique is 'jackknifing'.

- This is a special case of non-parametric bootstrapping.
- Generally used to estimate the bias and variance of a particular statistic.
- In this use-case, the statistic of interest repeatedly recalculated while leaving out one data point. The distribution of the statistic is then analyzed.
- Less computationally intensive than bootstrapping, since random numbers are left out of the calculation.
- Not as common as bootstrapping.
- The 'bootstrap' package contains functionality to perform jackknifing.
- This approach has nice statistical properties, so it is sometimes seen in a more theoretical context.

We won't do an example of this, but you need to be aware that it exists.

SciNet

Permutation tests

Another resampling tool is the permutation test.

- Permutation tests commonly appear when we are interested in the null hypothesis of no difference between two treatment groups.
- Like non-parametric bootstrapping, we build distributions by sampling from our existing data set. In permutation tests, this is done by "shuffling" the observations in the data (move the data from group A to group B).
- In this case, the permutation test exactly represents the inference process we are testing.
- Why? Because the null hypothesis is that there's no difference between the two groups. Thus, if we change the outcome of a particular subject from category A to B, the statistics shouldn't change if the null hypothesis is true.
- The two-sample t test is also used for testing this null hypothesis.

SciNet

Permutation tests, continued

How does it work, exactly?

- A full permutation test would consider every single possible permutation of the data (shuffling group A and group B data).
- This gets out of hand quickly, even for small data sets. Shuffling 20 data points would mean $\binom{20}{10}$ combinations, (assuming two equally-sized groups) which is 184,756.
- We instead perform an "approximate permutation test" by randomly sampling from the space of all possible permutations.
- For each permutation, we calculate the statistic that we're after, and thus get a distribution. We then compare the distribution to the original value of the statistic (usually the mean).

SciNet

Permutation test, example

Consider again the birthwt data set from the MASS library.

- Let's look at the birthwt data from smoking and non-smoking mothers.
- First lets do a two-sample t test.

```
> smoking <- birthwt$bwt[birthwt$smoke == 1]
> non.smoking <- birthwt$bwt[birthwt$smoke == 0]
>
> t.test(smoking, non.smoking)

Welch Two Sample t-test

data:  smoking and non.smoking
t = -2.7299, df = 170.1, p-value = 0.007003
alternative hypothesis:  true difference in means is
not equal to 0
95 percent confidence interval:
-488.97860 -78.57486
sample estimates:
mean of x mean of y
2771.919 3055.696
>
```

Permutation test, example, continued

Let's do a permutation test.

- The permTS stands for "two-sample permutation test".
- The 'alternative' flag specifies the alternative hypothesis.
- The 'method' flag indicates to do Monte Carlo sampling of the permutation space, not the full permutation.
- The control flag does exactly that
  - nmc: number of permutation samplings.
  - tsmethod indicates how to calculate the two-sided p-values.

```
> library(perm)  # you need to install this
> permTS(smoking, non.smoking,
+     alternative = "two.sided",
+     method = "exact.mc", control =
+     permControl(nmc = 2000, tsmethod = "central"))
Exact Permutation Test Estimated by Monte Carlo

data:  smoking and GROUP 2
p-value = 0.008996
alternative hypothesis:  true mean smoking - mean
GROUP 2 is not equal 0
sample estimates:
mean smoking - mean GROUP 2
-283.7767

p-value estimated from 2000 Monte Carlo replications
99 percent confidence interval on p-value:
0.003120622 0.013374953
```

SciNet

Summary

Some things to remember:

- Split your data into training, testing, and optionally, validation data sets. Train using the training data, test the model on the test data.
- Use cross-validation to determine the free parameters of your models! Bootstrapping can be used to get statistics on statistics.
- Use non-parametric bootstrapping if you don't know the distribution of your data. Use parametric if you do.
- Permutation tests are a family of resampling techniques which perform tests on data, by shuffling the data sets. They can be used to complement other tests.

SciNet