Model Selection in R

Charles J. Geyer

October 28, 2003

This used to be a section of my master's level theory notes. It is a bit overly theoretical for this R course. Just think of it as an example of literate programming in R using the `Sweave` function. You don't have to absorb all the theory, although it is there for your perusal if you are interested.

This discusses the problem of choosing among many models. Although our examples will be regression problems and some of the discussion and methods will be specific to linear regression, the problem is general. When many models are under consideration, how does one choose the best? Thus we also discuss methods that apply outside the regression context.

There are two main issues.

- Non-nested models
- Many models.

The only methods for model comparison we have studied, the F test for comparison of linear regression models and the likelihood ratio test for comparison of general models, are valid only for comparing two *nested* models. We also want to test *non-nested* models, and for that we need new theory. When more than two models are under consideration, the issue of correction for multiple testing arises. If there are only a handful of models, Bonferroni correction (or some similar procedure) may suffice. When there are many models, a conservative correction like Bonferroni is too conservative. Let's consider how many models might be under consideration in a model selection problem. Consider a regression problem with k predictor variables.

- [**Almost the worst case**] There are 2^k possible submodels formed by choosing a subset of the k predictors to include in the model (because a set with k elements has 2^k subsets).
- [**Actually the worst case**] That doesn't consider all the new predictors that one might "make up" using functions of the old predictors. Thus there are potentially infinitely many models under consideration.

Bonferroni correction for infinitely many tests is undefined. Even for 2^k tests with k large, Bonferroni is completely pointless. It would make nothing statistically significant.

1 Overfitting

> *Least squares is **good** for model fitting, but **useless** for model selection.*

Why? A bigger model *always* has a smaller residual sum of squares, just because a minimum taken over a larger set is smaller. Thus least squares, taken as a criterion for model selection says "always choose the biggest model." But this is silly. Consider what the principle of choosing the biggest model says about polynomial regression.

Example 1 (Overfitting in Polynomial Regression).
Consider the regression data in the file `ex12.7.1.dat` in this directory, which is read by

```
> X <- read.table("ex12.7.1.dat", header = TRUE)
> names(X)

[1] "x" "y"

> attach(X)
```

The "simple" linear regression is done and a plot made (Figure 1) by the following

```
> par(mar = c(5, 4, 1, 1) + 0.1)
> plot(x, y)
> out <- lm(y ~ x)
> abline(out)
```

Couldn't ask for nicer data for simple linear regression. The data appear to fit the "simple" model (one non-constant predictor, which we take to be x itself).

But now consider what happens when we try to be a bit more sophisticated. How do we know that the "simple" model is o. k.? Perhaps we should consider some more complicated models. How about trying polynomial regression? But if we consider all possible polynomial models, that's an infinite number of models (polynomials of all orders).

Although an infinite number of models are potentially under consideration, the fact that the data set is finite limits the number of models that actually need to be considered to a finite subset. You may recall from algebra that any set of n points in the plane having n different x values can be interpolated (fit exactly) by a polynomial of degree $n - 1$. A polynomial that fits exactly has residual sum of squares zero (fits perfectly). Can't do better than that by the least squares criterion! Thus all polynomials of degree at least $n - 1$ will give the same fitted values and zero residual sum of squares. Although they may give different predicted values for x values that do not occur in the data, they give the same predicted values at those that do. Hence the polynomials with degree at least $n - 1$ cannot be distinguished by least squares. In fact

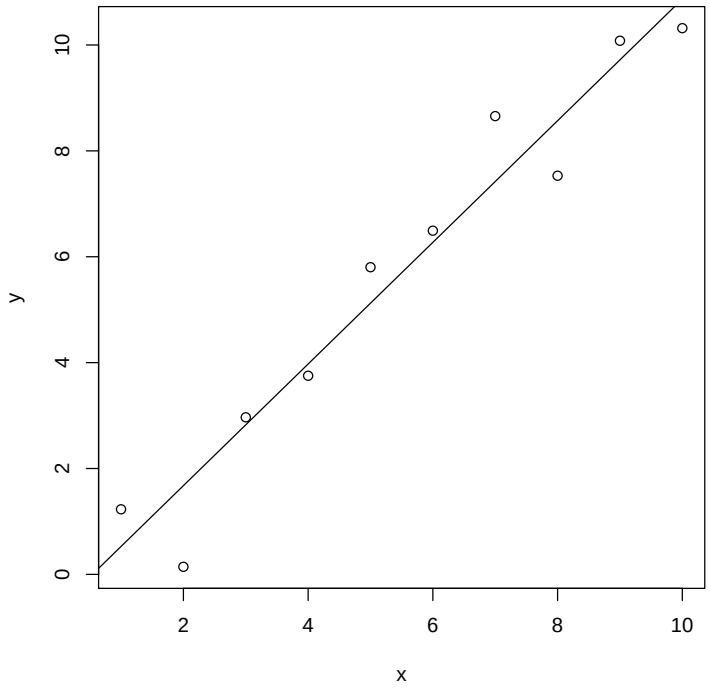

Figure 1: Some regression data. With fitted "simple" regression function of the form $y = \hat{\alpha} + \hat{\beta}x$.

the polynomials with degree more than $n - 1$ cannot be fit at all, because they have more parameters than there are equations to determine them. $\mathbf{X}'\mathbf{X}$ is a singular matrix and cannot inverted to give unique estimates of the regression coefficients.

This is a general phenomenon, which occurs in all settings, not just with linear regression. Models with more parameters than there are data points are underdetermined. Many different parameter vectors give the same likelihood, or the same empirical moments for the method of moments, or the same for whatever criterion is being used for parameter estimation. Thus in general, even when an infinite number of models are theoretically under consideration, only n models are practically under consideration, where n is the sample size.

Hence the "biggest model" that the least squares criterion selects is the polynomial of degree $n - 1$. What does it look like? The following code fits this model

```
> out.poly <- lm(y ~ poly(x, 9))
> summary(out.poly)

Call:
lm(formula = y ~ poly(x, 9))

Residuals:
ALL 10 residuals are 0: no residual degrees of freedom!

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   5.6974
poly(x, 9)1  10.4359
poly(x, 9)2  -0.6243
poly(x, 9)3  -0.8362
poly(x, 9)4   1.1783
poly(x, 9)5  -0.7855
poly(x, 9)6   0.1269
poly(x, 9)7  -1.3901
poly(x, 9)8  -0.4744
poly(x, 9)9  -1.0935

Residual standard error: NaN on 0 degrees of freedom
Multiple R-Squared:     1,        Adjusted R-squared:   NaN
F-statistic:   NaN on 9 and 0 DF,  p-value: NA
```

(Note the zero residual degrees of freedom. That's the overfitting!)

Then

```
> par(mar = c(5, 4, 1, 1) + 0.1)
> plot(x, y)
> curve(predict(out.poly, data.frame(x = x)), add = TRUE)
> abline(out, lty = 2)
```

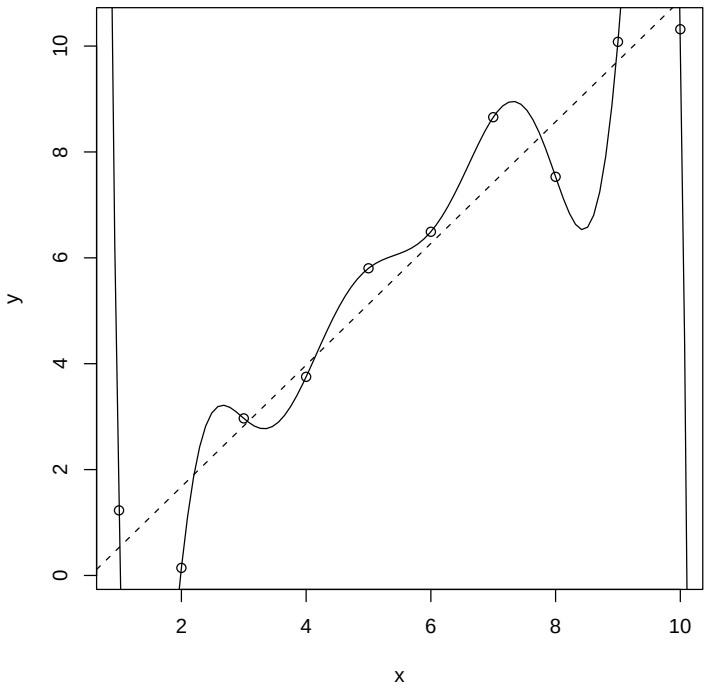

Figure 2: Some regression data. With fitted linear regression function (dashed line) and ninth degree polynomial regression function (solid curve). See also Figure 3.

plots the two regression functions for our fits (Figure 2) the best fitting (perfectly fitting!) polynomial of degree $n - 1 = 9$ and the least squares regression line from the other figure.

How well does the biggest model do? It fits the observed data perfectly, but it's hard to believe that it would fit *new* data from the same population as well. The extreme oscillations near the ends of the range of the data are obviously nonsensical, but even the smaller oscillations in the middle seem to be tracking random noise rather than any real features of the population regression function. Of course, we don't actually know what the true population regression function is. It could be either of the two functions graphed in the figure, or it could be some other function. But it's hard to believe, when the linear function fits so well, that something as complicated as the ninth degree polynomial is close to the true regression function. We say it "overfits" the data, meaning it's *too* close to the data and not close enough to the true population regression function.

Let's try again on Figure 2. Perhaps we would like to see the extent of those big oscillations. Something like this should do it.

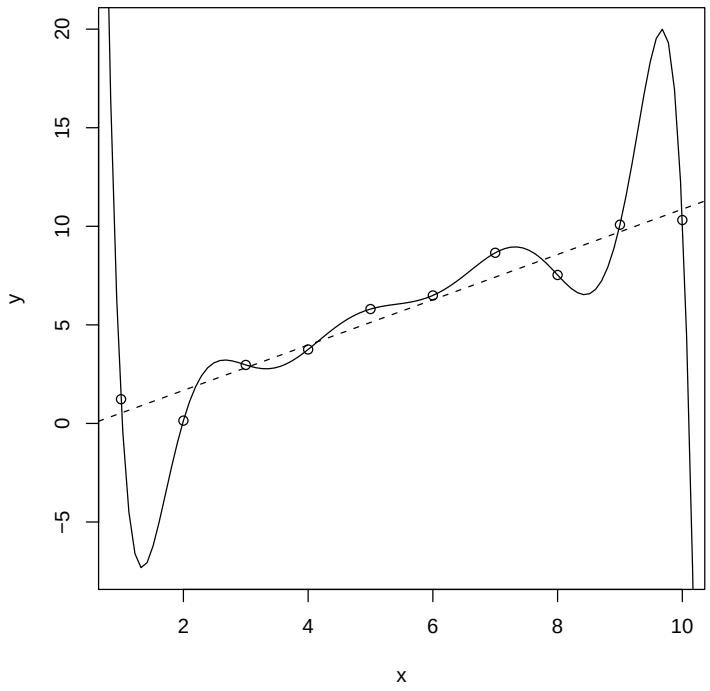

Figure 3: Same as Figure 2 except for greater vertical extent.

```
> par(mar = c(5, 4, 1, 1) + 0.1)
> xx <- seq(min(x), max(x), length = 1000)
> yy <- predict(out.poly, data.frame(x = xx))
> plot(x, y, ylim = range(yy))
> curve(predict(out.poly, data.frame(x = x)), add = TRUE)
> abline(out, lty = 2)
```

The result is shown in Figure 3.

2 Mean Square Error

So what criterion should we use for model selection if residual sum of squares is no good? One theoretical criterion is mean square error. In the regression setting, it is unclear what quantities mean square error should apply to. Do we use the mean square error of the parameter estimates? We haven't even defined mean square error for vector quantities. That alone suggests we should avoid looking at MSE of regression coefficients. There is also our slogan that regression coefficients are meaningless. Hence we should look at estimates of

the regression function.

But here too, there are still issues that need to be clarified. The regression function $h(\mathbf{x}) = E(Y \mid \mathbf{x})$ is a scalar function of $\mathbf{x}$. (Hence we don't need to worry about MSE of a vector quantity). But it is a function of the predictor value $\mathbf{x}$.

$$\text{mse}\{\hat{h}(\mathbf{x})\} = \text{variance} + \text{bias}^2 = \text{var}\{\hat{h}(\mathbf{x})\} + \Big(E\{\hat{h}(\mathbf{x})\} - h(\mathbf{x})\Big)^2 \tag{1}$$

where $h(\mathbf{x})$ is the true population regression function and $\hat{h}(\mathbf{x})$ is an estimate (we are thinking of least squares regression estimates, but (1) applies to any estimate). ("Bias?" did I hear someone say? Aren't linear regression estimates *unbiased*? Yes, they are when the model is *correct*. Here we are considering cases when the model is too small to contain the true regression function.)

To make a criterion that we can minimize to find the best model, we need a single scalar quantity, not a function. There are several things we could do to make a scalar quantity from (1). We could *integrate* it over some range of values, obtaining so-called *integrated mean squared error*. A simpler alternative is to sum it over the *design points*, the $\mathbf{x}$ values occurring in the data set under discussion. We'll discuss only the latter.

Write $\boldsymbol{\mu}$ for the true population means of the responses given the predictors, defined by $\mu_i = h(\mathbf{x}_i)$. Let m index models. The m-th model will have design matrix $\mathbf{X}_m$ and hat matrix

$$\mathbf{H}_m = \mathbf{X}_m(\mathbf{X}_m'\mathbf{X}_m)^{-1}\mathbf{X}_m'. \tag{2}$$

It is called the hat matrix because it "puts the hat on" $\mathbf{y}$, that is, $\hat{\mathbf{y}}_m = \mathbf{H}_m\mathbf{y}$ is the vector of predicted values for the m-th model. The expected value of the regression predictions is

$$E(\hat{\mathbf{y}}_m) = \mathbf{H}_m\boldsymbol{\mu}.$$

Hence the bias is

$$E(\hat{\mathbf{y}}) - E(\mathbf{y}) = -(\mathbf{I} - \mathbf{H}_m)\boldsymbol{\mu}. \tag{3}$$

If the m-th model is correct, then $\boldsymbol{\mu}$ is in the range of $\mathbf{H}_m$ and the bias is zero. Models that are incorrect have nonzero bias. The bias (3) is, of course, a vector. Its i-th element gives the bias of $\hat{y}_i$. What we decided to study was the sum of the MSEs at the design points, the "bias" part of which is just the sum of squares of the elements of (3), which is the same thing as the squared length of this vector

$$\text{bias}^2 = \|(\mathbf{I} - \mathbf{H}_m)\boldsymbol{\mu}\|^2 = \boldsymbol{\mu}'(\mathbf{I} - \mathbf{H}_m)^2\boldsymbol{\mu} = \boldsymbol{\mu}'(\mathbf{I} - \mathbf{H}_m)\boldsymbol{\mu}. \tag{4}$$

The variance is

$$\begin{aligned}
\operatorname{var}(\hat{\mathbf{y}}) &= E\left\{\left\|\hat{\mathbf{y}} - E(\hat{\mathbf{y}})\right\|^2\right\} \\
&= E\left\{\left\|\hat{\mathbf{y}} - \mathbf{H}_m\boldsymbol{\mu}\right\|^2\right\} \\
&= E\left\{\left\|\mathbf{H}_m(\mathbf{y} - \boldsymbol{\mu})\right\|^2\right\} \\
&= E\left\{\mathbf{H}_m(\mathbf{y} - \boldsymbol{\mu})'(\mathbf{y} - \boldsymbol{\mu})\mathbf{H}_m\right\} \\
&= \sigma^2\mathbf{H}_m^2 \\
&= \sigma^2\mathbf{H}_m
\end{aligned}$$

This is, of course, a matrix. What we want though is just the sum of the diagonal elements. The i-th diagonal element is the variance of $\hat{y}_i$, and our decision to focus on the sum of the MSEs at the design points says we want the sum of these. The sum of the diagonal elements of a square matrix $\mathbf{A}$ is called its *trace*, denoted $\operatorname{tr}(\mathbf{A})$. Thus the "variance" part of the MSE is

$$\text{variance} = \sigma^2 \operatorname{tr}(\mathbf{H}_m). \tag{5}$$

And

$$\text{mse}_m = \sum_{i=1}^{n} \text{mse}(\hat{y}_i) = \sigma^2 \operatorname{tr}(\mathbf{H}_m) + \boldsymbol{\mu}'(\mathbf{I} - \mathbf{H}_m)\boldsymbol{\mu}.$$

3 The Bias-Variance Trade-Off

Generally, one cannot reduce both bias and variance at the same time. Bigger models have less bias but *more* variance. Smaller models have less variance but *more* bias. This is called the "bias-variance trade-off."

Example 2.
Consider the regression data in the file `ex12.3.2.dat` in this directory, which is read by

```
> X <- read.table("ex12.3.2.dat", header = TRUE)
> names(X)

[1] "x" "y"

> attach(X)
```

Figure 4 shows the scatter plot.

A mere glance at the plot shows that y is decidedly not a linear function of x, not even close. However, there is nothing that prevents us from making up more predictor variables. The data set itself has no other variables in it, just x and y. So any new predictor variables we make up must be functions of x. What functions?

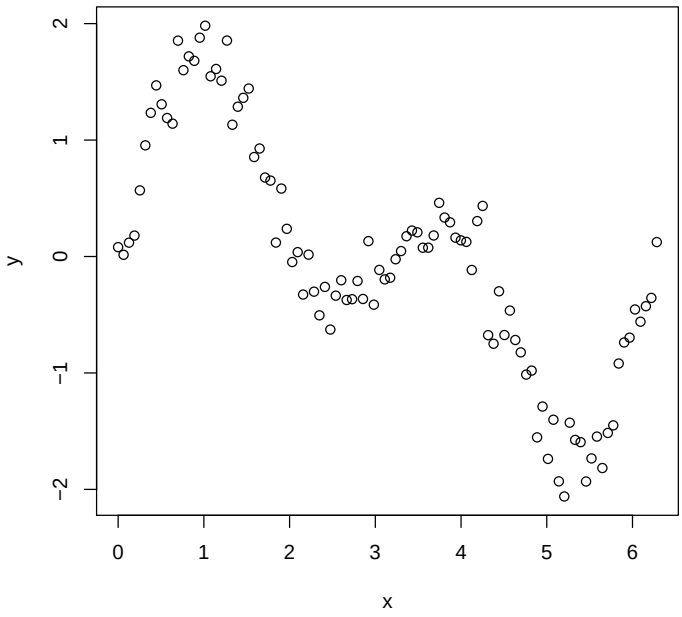

Figure 4: Some regression data.

Since we are told nothing about the data, we have no guidance as to what functions to make up. In a real application, there might be some guidance from scientific theories that describe the data. Or there might not. Users of linear regression often have no preconceived ideas as to what particular functional form the regression function may have. One thing that is often done, more or less for lack of any better ideas, is to try a polynomial.

The following fits a bunch of polynomials and compares them.

```
> out1 <- lm(y ~ x)
> out2 <- update(out1, . ~ . + I(x^2))
> out3 <- update(out2, . ~ . + I(x^3))
> out4 <- update(out3, . ~ . + I(x^4))
> out5 <- update(out4, . ~ . + I(x^5))
> out6 <- update(out5, . ~ . + I(x^6))
> out7 <- update(out6, . ~ . + I(x^7))
> anova(out1, out2, out3, out4, out5, out6, out7)

Analysis of Variance Table

Model 1: y ~ x
Model 2: y ~ x + I(x^2)
Model 3: y ~ x + I(x^2) + I(x^3)
Model 4: y ~ x + I(x^2) + I(x^3) + I(x^4)
Model 5: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5)
Model 6: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6)
Model 7: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7)
  Res.Df    RSS Df Sum of Sq        F Pr(>F)
1     98 38.455
2     97 38.434  1     0.021   0.5072 0.4782
3     96 33.075  1     5.359 129.1406 <2e-16 ***
4     95 33.007  1     0.068   1.6364 0.2040
5     94  8.833  1    24.174 582.4921 <2e-16 ***
6     93  8.820  1     0.013   0.3048 0.5822
7     92  3.818  1     5.002 120.5270 <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

(no particular reason for stopping at 7).

It is clear from the printout that odd degree polynomials fit way better than the even ones (strictly speaking, better than the immediately preceding even one to which they are compared). In hindsight, this is obvious. A glance at Figure 4 shows that the regression function is bending down on the left side and up on the right. This is the behavior of an odd degree polynomial. An even degree polynomial heads the same way at both sides (both up or both down). Thus we need only consider odd degree polynomials. Let's add some more.

```
> out9 <- update(out7, . ~ . + I(x^8) + I(x^9))
> out11 <- update(out9, . ~ . + I(x^10) + I(x^11))
```

```
> out13 <- update(out11, . ~ . + I(x^12) + I(x^13))
> anova(out1, out3, out5, out7, out9, out11, out13)

Analysis of Variance Table

Model 1: y ~ x
Model 2: y ~ x + I(x^2) + I(x^3)
Model 3: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5)
Model 4: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7)
Model 5: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) +
    I(x^8) + I(x^9)
Model 6: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) +
    I(x^8) + I(x^9) + I(x^10) + I(x^11)
Model 7: y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) +
    I(x^8) + I(x^9) + I(x^10) + I(x^11) + I(x^12) + I(x^13)
  Res.Df    RSS Df Sum of Sq        F  Pr(>F)
1     98 38.455
2     96 33.075  2     5.381  63.8192 < 2e-16 ***
3     94  8.833  2    24.242 287.5378 < 2e-16 ***
4     92  3.818  2     5.015  59.4796 < 2e-16 ***
5     90  3.788  2     0.030   0.3545 0.70256
6     88  3.788  2 0.0001081   0.0013 0.99872
7     87  3.667  1     0.121   2.8627 0.09424 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Hmmmmm. Seems we stopped in the right place. Model 4 here, which is the polynomial of degree 7 is clearly fits much better than model 3. But models 5, 6, and 7 don't fit significantly better than model 4. Thus it is fairly clear that model 4 is the one to go with.

Double Hmmmmm. The 1 degree of freedom for comparing models 6 and 7 is wrong. The only reason I can think of for this is that using the monomials as predictors is very unstable. Let's try again.

```
> out3 <- lm(y ~ poly(x, 3))
> out5 <- lm(y ~ poly(x, 5))
> out7 <- lm(y ~ poly(x, 7))
> out9 <- lm(y ~ poly(x, 9))
> out11 <- lm(y ~ poly(x, 11))
> out13 <- lm(y ~ poly(x, 13))
> anova(out1, out3, out5, out7, out9, out11, out13)

Analysis of Variance Table

Model 1: y ~ x
Model 2: y ~ poly(x, 3)
Model 3: y ~ poly(x, 5)
```

```
Model 4: y ~ poly(x, 7)
Model 5: y ~ poly(x, 9)
Model 6: y ~ poly(x, 11)
Model 7: y ~ poly(x, 13)
  Res.Df    RSS Df Sum of Sq        F Pr(>F)
1     98 38.455
2     96 33.075  2     5.381  63.4536 <2e-16 ***
3     94  8.833  2    24.242 285.8905 <2e-16 ***
4     92  3.818  2     5.015  59.1388 <2e-16 ***
5     90  3.788  2     0.030   0.3524 0.7040
6     88  3.788  2 0.0001081   0.0013 0.9987
7     86  3.646  2     0.142   1.6739 0.1936
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

No difference at all except for that last comparison, which `poly` gets right!

Figure 5 shows the scatter plot again with the regression function for the 7th degree polynomial added.

From a data analytic point of view, there's nothing more to be said. From a theoretical point of view, we haven't even started. Theoretically, we haven't yet looked at the bias-variance tradeoff, or for that matter at the bias or variance. These are theoretical quantities that cannot be discovered by looking at fitted models.

To know the bias we need to know the true population regression function. I do happen to know that here because this is simulated data. The true population regression function is trignometric, given by

$$\mu_i = \sin(x_i) + \sin(2x_i) \tag{6a}$$

$$\sigma = 0.2 \tag{6b}$$

Again we consider various polynomial regression models. Since we are not being theoretical, we use only the x values and ignore the y values.

Since the true population regression curve is a trigonometric rather than a polynomial function, *no* polynomial is unbiased. This is typical of real applications. No model under consideration is exactly correct.

The R function `lm` returns the design matrix if asked, and from it we can construct the hat matrix, for example

```
> out7 <- lm(y ~ poly(x, 7), x = TRUE)
> hat7 <- out7$x %*% solve(t(out7$x) %*% out7$x) %*% t(out7$x)
> all.equal(predict(out7), as.numeric(hat7 %*% y))
```

```
[1] TRUE
```

Then (integrated) bias squared and variance are given by

```
> mu <- sin(x) + sin(2 * x)
> sigma <- 0.2
```

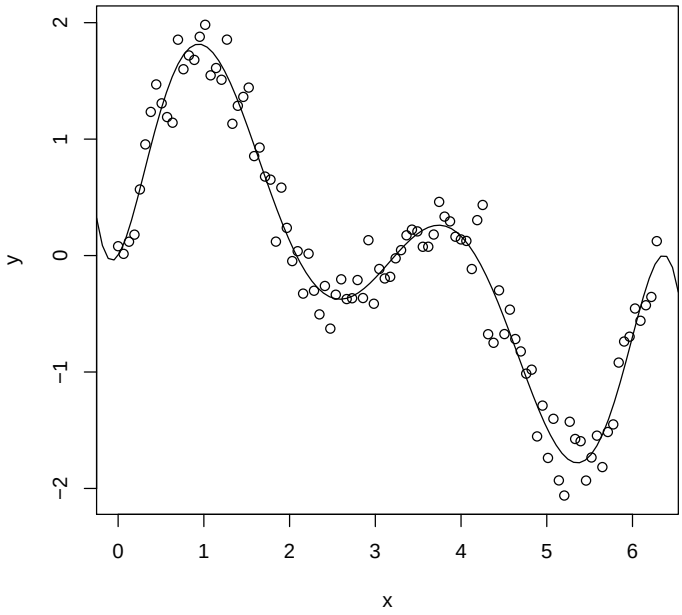

Figure 5: Same regression data as in Figure 4. Now with sample regression function in the model of all polynomials of degree seven.

```
> n <- length(x)
> t(mu) %*% (diag(n) - hat7) %*% mu

            [,1]
  [1,] 0.1632783

> sigma^2 * sum(diag(hat7))

[1] 0.32
```

Put the above in a loop.

```
> kmax <- 20
> bsq <- double(kmax)
> v <- double(kmax)
> for (k in 1:kmax) {
+     out <- lm(y ~ poly(x, k), x = TRUE)
+     hat <- out$x %*% solve(t(out$x) %*% out$x) %*% t(out$x)
+     bsq[k] <- t(mu) %*% (diag(n) - hat) %*% mu
+     v[k] <- sigma^2 * sum(diag(hat))
+ }
> foo <- cbind(1:kmax, bsq, v, bsq + v)
> dimnames(foo) <- list(NULL, c("degree", "bias^2", "variance",
+     "mse"))
> foo

      degree       bias^2 variance        mse
 [1,]      1 3.338474e+01     0.08 33.4647442
 [2,]      2 3.338474e+01     0.12 33.5047442
 [3,]      3 2.909238e+01     0.16 29.2523829
 [4,]      4 2.909238e+01     0.20 29.2923829
 [5,]      5 4.855222e+00     0.24  5.0952222
 [6,]      6 4.855222e+00     0.28  5.1352222
 [7,]      7 1.632783e-01     0.32  0.4832783
 [8,]      8 1.632783e-01     0.36  0.5232783
 [9,]      9 1.899415e-03     0.40  0.4018994
[10,]     10 1.899415e-03     0.44  0.4418994
[11,]     11 9.794280e-06     0.48  0.4800098
[12,]     12 9.794280e-06     0.52  0.5200098
[13,]     13 2.588958e-08     0.56  0.5600000
[14,]     14 2.588958e-08     0.60  0.6000000
[15,]     15 3.868572e-11     0.64  0.6400000
[16,]     16 3.868739e-11     0.68  0.6800000
[17,]     17 5.607338e-14     0.72  0.7200000
[18,]     18 5.551309e-14     0.76  0.7600000
[19,]     19 1.951232e-14     0.80  0.8000000
[20,]     20 2.140150e-14     0.84  0.8400000
```

The most interesting fact here is that the best model in terms of mean square error has degree 9 (not the degree 7 model chosen by the data analysis). Of course, to know the theoretically best model, we need to know the true population regression function `mu` and when we are doing a real data analysis, we don't. The point is that there is a theoretically best model and the data analysis need not find it.

Also very interesting is that we don't want to be less biased, much less unbiased. Many people exposed to bad statistics teaching think unbiasedness is absolutely wonderful. But here we see that less bias is bad when we go beyond the point of optimum bias-variance tradeoff. This is quite a general point. Linear regression is only unbiased when all of the model assumptions are correct: the true population regression function is among those under consideration (exactly) and the errors are IID (again exactly). In real life, these assumptions are rarely if ever known to be correct. So unbiasedness is mythical.

Of minor interest is the property that the even degree models have the same bias as the next lower odd degree model. This is because the true population regression is an odd function, satisfying $f(x) = -f(-x)$, which makes the true population regression coefficients on the even degree polynomials, which are even functions, satisfying $f(x) = f(-x)$, exactly zero.

Figure 6 shows both the true regression function used to simulate the response values (6a) and the sample ninth-degree polynomial regression function. The R statements making the figure are

```
> par(mar = c(5, 4, 1, 1) + 0.1)
> plot(x, y)
> curve(sin(x) + sin(2 * x), add = TRUE)
> curve(predict(out9, data.frame(x = x)), add = TRUE, lty = 2)
```

The result is shown in Figure 6.

We see that the sample regression function does not estimate the true regression function perfectly (because the sample is not the population), but we now know from the theoretical analysis in this example that no polynomial will fit better. A lower degree polynomial will have less variance. A higher degree will have less bias. But the bias-variance trade-off will be worse for either.

4 Model Selection Criteria

The theory discussed in the preceding section gives us a framework for discussing model selection. We want the model with the smallest MSE, the model which makes the optimal bias-variance trade-off.

Unfortunately, this is useless in practice. Mean square error is a theoretical quantity. It depends on the unknown true regression function and the unknown error variance. Furthermore, there is no obvious way to estimate it. Without knowing which models are good, which is exactly the question we are trying to resolve, we can't get a good estimate of the true regression function (and without that we can't estimate the error variance well either).

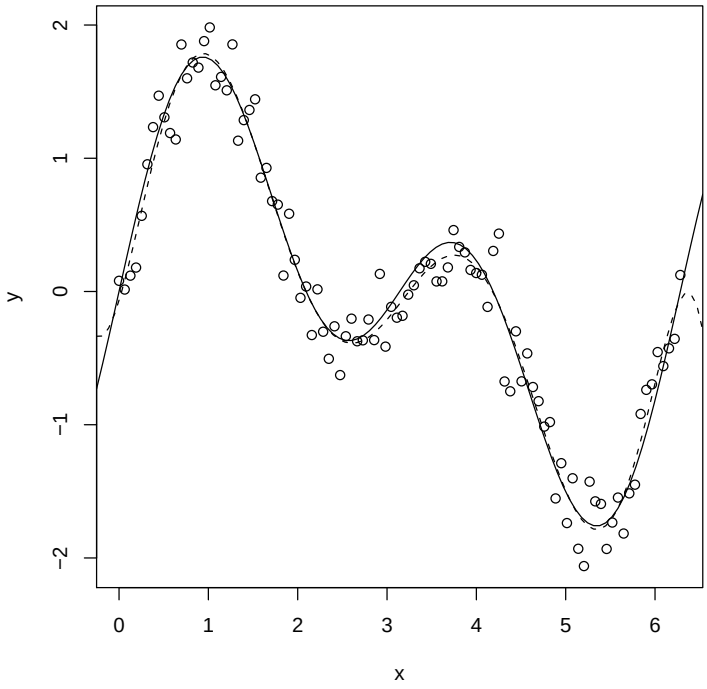

Figure 6: Some regression data with true regression function (solid line) and sample regression function (dashed line).

There are several quantities that have been proposed in the literature as estimates of MSE. No one is obviously right. We shall not go into the theory justifying them (it is merely heuristic anyway). One fairly natural quantity is

$$\sum_{i=1}^{n} \hat{e}_{(i)}^2 \tag{7}$$

where $\hat{e}_{(i)}$ is the so-called leave-one-out residual

$$\hat{e}_{(i)} = y_i - \hat{y}_{(i)}$$

where $\hat{y}_{(i)}$ is the predicted value at the point x_i from the model fitted to the data that *leaves out* y_i. The idea is that this gives an "honest" estimate of the error because y_i is not used in calculating $\hat{y}_{(i)}$. Hence $\hat{y}_{(i)}$ can't be "too close" to y_i since it doesn't "know" about y_i.

It turns out that it is not necessary to do n different regressions (each leaving out one y_i). All of the leave-one-out residuals can be calculated from the fit in the full regression model using

$$\hat{e}_{(i)} = \frac{\hat{e}_i}{1 - h_{ii}} \tag{8}$$

where h_{ii} is the i, i element of the hat matrix. This nifty algebraic fact makes (7) much easier to calculate than it appears at first sight and consequently makes (7) a much more useful criterion.

Unlike SSResid (the sum of the $\hat{e}_i^2$), this does not always favor the biggest model. Models that overfit tend to do a bad job of their leave-one-out predictions. (7) is called the *predicted residual sum of squares* (PRESS) or the *cross-validated sum of squares* (CVSS), cross-validation being another term used to describe the leave-one-out idea.

The idea of CVSS or other criteria to be described presently is to pick the model with the smallest value of the criterion. This will not necessarily be the model with the smallest MSE, but it is a reasonable estimate of it.

Another criterion somewhat easier to calculate is Mallows' C_p defined by

$$\begin{aligned} C_p &= \frac{\text{SSResid}_p}{\hat{\sigma}^2} + 2p - n \\ &= \frac{\text{SSResid}_p - \text{SSResid}_k}{\hat{\sigma}^2} + p - (k - p) \\ &= (k - p)(F_{k-p,n-k} - 1) + p \end{aligned} \tag{9}$$

where SSResid_p is the sum of squares of the residuals for some model with p predictors (including the constant predictor, if present), $\hat{\sigma}^2 = \text{SSResid}_k/(n-k)$ is the estimated error variance for the largest model under consideration, which has k predictors, and $F_{k-p,n-k}$ is the F statistic for the F test for comparison of these two models. The F statistic is about one in size if the small model is correct, in which case $C_p \approx p$. This gives us a criterion for finding reasonably

fitting models. When many models are under consideration, many of them may have $C_p \approx p$ or smaller. All such models must be considered reasonably good fits. Any might be the correct model or as close to correct as any model under consideration. The quantity estimated by C_p is the mean square error of the model with p predictors, divided by σ^2.

An idea somewhat related to Mallows' C_p, but applicable outside the regression context is the *Akaike information criterion* (AIC).

$$-2 \cdot (\log \text{ likelihood}) + 2p \tag{10}$$

where as in Mallows' C_p, the number of parameters in the model is p. Although (9) and (10) both have the term $2p$, they are otherwise different. The log likelihood for a linear regression model, with the MLEs plugged in for the parameters is

$$\log \text{ likelihood} = \text{constant} - \frac{n}{2} \log(\text{SSResid})$$

Thus for a regression model with p predictors,

$$\text{AIC} = \text{constant} + n \log(\text{SSResid}_p) + 2p$$

where "constant" here is a term that does not depend on the model and hence does not matter in model comparisons.

Finally, we add one last criterion, almost the same as AIC, called the *Bayes information criterion* (BIC)

$$-2 \cdot (\log \text{ likelihood}) + p \log(n) \tag{11}$$

In the regression context, this becomes

$$\text{BIC} = \text{constant} + n \log(\text{SSResid}_p) + p \log(n).$$

Neither AIC nor BIC have a rigorous theoretical justification applicable to a wide variety of models. Both were derived for special classes of models that were easy to analyze and both involve some approximations. Neither can be claimed to be the right thing (nor can anything else). As the "Bayes" in BIC indicates, the BIC criterion is intended to approximate using Bayes tests instead of frequentist tests (although its approximation to true Bayes tests is fairly crude). Note that BIC penalizes models with more parameters more strongly than AIC ($p \log n$ versus $2p$). So BIC always selects a larger model than AIC.

This gives us four criteria for model selection. There are arguments in favor of each. None of the arguments are completely convincing. All are widely used.

Example 3.
We use the data for Example 2 again. Now we fit polynomials of various degrees to the data, and look at our four criteria.

```
> n <- length(y)
> deg <- seq(1, 15, 2)
```

```
> cvss <- double(length(deg))
> cp <- double(length(deg))
> aic <- double(length(deg))
> bic <- double(length(deg))
> out.big <- lm(y ~ poly(x, 17))
> sigsqhat.big <- summary(out.big)$sigma^2
> for (i in seq(along = deg)) {
+     k <- deg[i]
+     out <- lm(y ~ poly(x, k))
+     aic[i] <- AIC(out)
+     bic[i] <- AIC(out, k = log(n))
+     cvss[i] <- sum((out$residuals/(1 - hatvalues(out)))^2)
+     cp[i] <- sum(out$residuals^2)/sigsqhat.big + 2 * out$rank -
+         n
+ }
> foo <- cbind(deg, cvss, cp, aic, bic)
> dimnames(foo) <- list(NULL, c("degree", "CVSS", "Cp", "AIC",
+     "BIC"))
> foo
```

```
     degree      CVSS         Cp       AIC        BIC
[1,]      1 40.439523 828.779469 194.22004 202.035551
[2,]      3 36.056686 703.387578 183.14746 196.173313
[3,]      5 10.720251 124.411825  55.11721  73.353398
[4,]      7  4.523134   7.818442 -24.75417  -1.307641
[5,]      9  4.906861  11.099780 -21.53995   7.116921
[6,]     11  5.712702  15.097179 -17.54281  16.324407
[7,]     13  5.744573  15.683763 -17.36182  21.715737
[8,]     15  4.893021  15.636742 -18.08720  26.200696
```

In this example, all four criteria select the same model, the polynomial of degree 7. This is not the model with the smallest MSE discovered by the theoretical analysis. The criteria do something sensible, but as everywhere else in statistics, there are errors (the sample is not the population).

5 All Subsets Regression

We now return to the situation in which there are k predictors including the constant predictor and 2^{k-1} models under consideration (the constant predictor is usually included in all models). If k is large and one has no non-statistical reason (e. g., a practical or scientific reason) that cuts down the number of models to be considered, then one must fit them all. Fortunately, there are fast algorithms that allow a huge number of models to be fit or at least quickly checked to see that they are much worse than other models of the same size.

There is a contributed package to R that contains a function `leaps` that does this.

Example 4 (2^k subsets).
The data set in the file `ex12.7.4.dat` in this directory, read by

```
> X <- read.table("ex12.7.4.dat", header = TRUE)
> names(X)

 [1] "y"   "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x8"  "x9"  "x10" "x11"
[13] "x12" "x13" "x14" "x15" "x16" "x17" "x18" "x19" "x20"

> attach(X)
```

consists of multivariate normal data with one response variable y and 20 predictor variables x_1, ..., x_{20}. The predictors are correlated. The distribution from which they were simulated has all correlations equal to one-half. The actual (sample) correlations, of course, are all different because of chance variation.

The true population regression function (the one used to simulate the y values) was

$$y = x_1 + x_2 + x_3 + x_4 + x_5 + e \tag{12}$$

with error variance $\sigma^2 = 1.5^2$. We fit the model by

```
> out <- lm(y ~ ., data = X, x = TRUE)
> summary(out)

Call:
lm(formula = y ~ ., data = X, x = TRUE)

Residuals:
     Min       1Q   Median       3Q      Max
-3.08973 -0.66991 -0.01928  0.81824  3.14955

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.07879    0.17449  -0.452  0.65286
x1           1.11887    0.22303   5.017 3.17e-06 ***
x2           0.79401    0.26519   2.994  0.00367 **
x3           0.45118    0.25848   1.746  0.08478 .
x4           0.59879    0.23037   2.599  0.01115 *
x5           1.09573    0.24277   4.513 2.20e-05 ***
x6           0.26067    0.24220   1.076  0.28509
x7          -0.15959    0.21841  -0.731  0.46712
x8          -0.50182    0.23352  -2.149  0.03470 *
x9           0.14047    0.22888   0.614  0.54116
x10          0.37689    0.22831   1.651  0.10275
x11          0.39805    0.21722   1.832  0.07065 .
x12          0.38825    0.22396   1.734  0.08689 .
x13         -0.07910    0.23553  -0.336  0.73788
x14          0.26716    0.20737   1.288  0.20138
```

```
x15         -0.12016    0.23073  -0.521  0.60398
x16          0.08592    0.22372   0.384  0.70195
x17          0.31296    0.22719   1.378  0.17224
x18         -0.24605    0.23355  -1.054  0.29531
x19          0.10221    0.21503   0.475  0.63586
x20         -0.45956    0.23698  -1.939  0.05604 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.494 on 79 degrees of freedom
Multiple R-Squared: 0.8724,        Adjusted R-squared: 0.8401
F-statistic: 27.01 on 20 and 79 DF,  p-value: < 2.2e-16
```

We've left in the significance codes, bogus though they may be (more on this later), so you can easily spot the regression coefficients that the least squares fit indicates may be important. If we go only with the strongest evidence (two or three stars) we get as "significant" three of the five truly important regression coefficients [recall from (12) that the true nonzero regression coefficients are β_1 through β_5]. The other two are missed.

If we use a less stringent standard, say one or more stars, we do pick up another truly nonzero regression coefficient, but we also pick up a false one. Thus we now have both false negatives (we still missed β_3) and false positives (we've picked up β_8).

With the least stringent standard, all the coefficients marked by any of the "significance codes" we now have no false negatives (all five of the truly nonzero regression coefficients are now declared "significant") but we have four false positives.

No matter how you slice it, least squares regression doesn't pick the right model. Of course, this is no surprise. It's just "the sample is not the population." But it does show that the results of such model selection procedures must be treated with skepticism.

Actually, we haven't even started a sensible model selection procedure. Recall the slogan that if you want to know how good a model fits, you have to fit that model. So far we haven't fit any of the models we've discussed. We're fools to think we can pick out the good submodels just by looking at printout for the big model.

There is a function `leaps` in the `leaps` contributed package[1] that fits a huge number of models. By default, it finds the 10 best models of each size (number of regression coefficients) for which there are 10 or more models and finds all the models of other sizes.

It uses the inequality that a bigger model always has a smaller sum of squares to eliminate many models. Suppose we have already found 10 models of size p with SSResid less than 31.2. Suppose there was a model of size $p+1$ that we fit

[1]This has to be installed separately. It doesn't come with the default install. Like everything else about R, it can be found at `http://cran.r-project.org`.

and found its SSResid was 38.6. Finally suppose $\hat{\sigma}^2$ for the big model is 2.05. Now the C_p for the 10 best models of size p already found is

$$C_p = \frac{\text{SSResid}_p}{\hat{\sigma}^2} + 2p - n < \frac{31.2}{2.05} + 2p - n$$

and the C_p for any submodel of size p of the model with SSResid = 38.6 (i. e., models obtained by dropping one predictor from that model) has

$$C_p \geq \frac{38.6}{2.05} + 2p - n$$

This means that no such model can be better than the 10 already found, so they can be rejected even though we haven't bothered to fit them. Considerations of this sort make it possible for `leaps` to pick the 10 best of each size without fitting all or even a sizable fraction of the models of each size. Thus it manages to do in minutes what it couldn't do in a week if it actually had to fit all 2^k models. The reason why `leaps` uses C_p as its criterion rather than one of the others is that it is a simple function of SSResid and hence to these inequalities that permit its efficient operation.

We run the leaps function as follows, with the design matrix `x` defined as above,[2]

```
> par(mar = c(5, 4, 1, 1) + 0.1)
> x <- out$x
> dim(x)

[1] 100  21

> library(leaps)
> outs <- leaps(x, y, int = FALSE, strictly.compatible = FALSE)
> plot(outs$size, outs$Cp, log = "y", xlab = "p", ylab = expression(C[p]))
> lines(outs$size, outs$size)
```

Figure 7 shows the plot made by the two plot commands.

Every model with $C_p < p$, corresponding to the dots below the line is "good." There are a *huge* number of perfectly acceptable models, because for the larger p there are many more than 10 good models, which are not shown.

The best model according to the C_p criterion is one with $p = 12$, so 11 non-constant predictors, which happen to be x_1 through x_5 (the truly significant predictors) plus x_8, x_{10}, x_{11}, x_{12}, x_{14}, and x_{20}. We can get its regression output as follows.

```
> ifoo <- outs$Cp == min(outs$Cp)
> ifoo <- outs$which[ifoo, ]
```

[2]For some reason, `leaps` doesn't take formula expressions like `lm` does. The reason is probably historical. The equivalent S-plus function doesn't either, because it was written before S had model formulas and hasn't changed. The `strictly.compatible=FALSE` tells R not to be bug-for-bug compatible with S-plus.

```
[1] 100  21
```

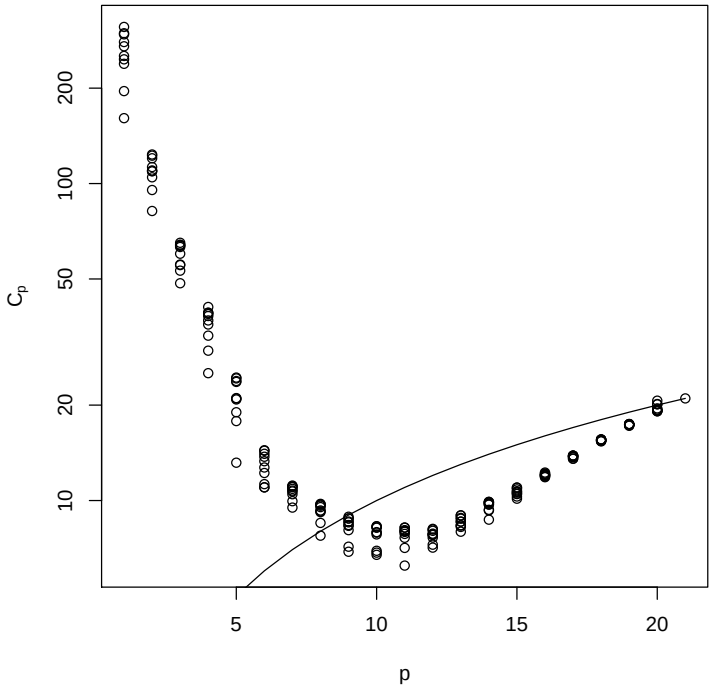

Figure 7: C_p plot. Plot of C_p versus p. The dots are the C_p for the 10 best models for each p. The curve is the line $C_p = p$ (curved because of the log scale for C_p).

```
> foo <- x[, ifoo]
> out.best <- lm(y ~ foo + 0)
> summary(out.best)

Call:
lm(formula = y ~ foo + 0)

Residuals:
     Min       1Q   Median       3Q      Max
-3.50162 -0.77683 -0.05537  0.95203  3.26516

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
foox1    1.0694     0.1982   5.395 5.61e-07 ***
foox2    0.8732     0.2419   3.610 0.000505 ***
foox3    0.4426     0.2274   1.946 0.054805 .
foox4    0.7271     0.1979   3.675 0.000406 ***
foox5    1.1517     0.2221   5.186 1.34e-06 ***
foox8   -0.3475     0.2090  -1.663 0.099923 .
foox10   0.4182     0.2111   1.981 0.050632 .
foox11   0.3402     0.2011   1.691 0.094266 .
foox12   0.3883     0.1968   1.973 0.051549 .
foox14   0.2999     0.1843   1.627 0.107181
foox20  -0.4955     0.2160  -2.294 0.024148 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.453 on 89 degrees of freedom
Multiple R-Squared: 0.8668,        Adjusted R-squared: 0.8503
F-statistic: 52.64 on 11 and 89 DF,  p-value: < 2.2e-16
```

Note that the "stargazing" doesn't correspond with the notion of the best model by the C_p criterion. One of these coefficients doesn't even have a dot (so for it $P > 0.10$), and four others only have dots ($0.05 < P < 0.10$). Considering them separately, this would lead us to drop them. But that would be the Wrong Thing (multiple testing without correction). The `leaps` function does as close to the Right Thing as can be done. The only defensible improvement would be to change the criterion, to BIC perhaps, which would choose a smaller "best" model because it penalizes larger models more. However BIC wouldn't have the nice inequalities that make `leaps` so efficient, which accounts for the use of C_p.

I hope you can see from this analysis that model selection when there are a huge number of models under consideration and no extra-statistical information (scientific, practical, etc.) that can be used to cut down the number is a mug's game. The best you can do is not very good. The only honest conclusion is that a huge number of models are about equally good, as good as one would expect the correct model to be ($C_p \approx p$).

Thus it is silly to get excited about exactly which model is chosen as the "best" by some model selection procedure (any procedure)! When many models are equally good, the specific features of any one of them can't be very important.

All of this is related to our slogan about "regression is for prediction, not explanation." All of the models with $C_p < p$ predict about equally well. So if regression is used for *prediction*, the model selection problem is not serious. Just pick any one of the many good models and use it. For *prediction* it doesn't matter which good prediction is used. But if regression is used for *explanation*, the model selection problem is insoluble. If you can't decide which model is "best" and are honest enough to admit that lots of other models are equally good, then how can you claim to have found the predictors which "explain" the response? Of course, if you really understand "correlation is not causation, and regression isn't either," then you know that such "explanations" are bogus anyway, even in the "simple" case (one non-constant predictor) where the model selection problem does not arise.