04:00 Wednesday 21st January, 2015

Chapter 3

Evaluating Statistical Models: Error and Inference

3.1 What Are Statistical Models For? Summaries, Forecasts, Simulators

There are (at least) three levels at which we can use statistical models in data analysis: as summaries of the data, as predictors, and as simulators.

The lowest and least demanding level is just to use the model as a summary of the data — to use it for **data reduction**, or **compression**. Just as one can use the sample mean or sample quantiles as descriptive statistics, recording some features of the data and saying nothing about a population or a generative process, we could use estimates of a model's parameters as descriptive summaries. Rather than remembering all the points on a scatter-plot, say, we'd just remember what the OLS regression surface was.

It's hard to be wrong about a summary, unless we just make a mistake. (It may or may not be *helpful* for us later, but that's different.) When we say "the slope which minimized the sum of squares was 4.02", we make no claims about anything *but* the training data. It relies on no assumptions, beyond our doing the calculations right. But it also asserts nothing about the rest of the world. As soon as we try to connect our training data to the rest of the world, we start relying on assumptions, and we run the risk of being wrong.

Probably the most common connection to want to make is to say what *other* data will look like — to make predictions. In a statistical model, with random noise terms, we do not anticipate that our predictions will ever be *exactly* right, but we also anticipate that our mistakes will show stable probabilistic patterns. We can evaluate predictions based on those patterns of error — how big is our typical mistake? are we biased in a particular direction? do we make a lot of little errors or a few huge ones?

Statistical inference about model parameters — estimation and hypothesis testing — can be seen as a kind of prediction, extrapolating from what we saw in a small

piece of data to what we would see in the whole population, or whole process. When we *estimate* the regression coefficient $\hat{b} = 4.02$, that involves predicting new values of the dependent variable, but also predicting that if we repeated the experiment and re-estimated $\hat{b}$, we'd get a value close to 4.02.

Using a model to summarize old data, or to predict new data, doesn't commit us to assuming that the model describes the process which generates the data. But we often want to do that, because we want to interpret parts of the model as aspects of the real world. We think that in neighborhoods where people have more money, they spend more on houses — perhaps each extra $1000 in income translates into an extra $4020 in house prices. Used this way, statistical models become stories about how the data were generated. If they are accurate, we should be able to use them to *simulate* that process, to step through it and produce something that looks, probabilistically, just like the actual data. This is often what people have in mind when they talk about *scientific* models, rather than just statistical ones.

An example: if you want to predict where in the night sky the planets will be, you can actually do very well with a model where the Earth is at the center of the universe, and the Sun and everything else revolve around it. You can even estimate, from data, how fast Mars (for example) goes around the Earth, or where, in this model, it should be tonight. But, since the Earth is *not* at the center of the solar system, those parameters don't actually refer to anything in reality. They are just mathematical fictions. On the other hand, we can also predict where the planets will appear in the sky using models where all the planets orbit the Sun, and the parameters of the orbit of Mars in that model *do* refer to reality.[1]

This chapter focuses on evaluating predictions, for three reasons. First, often we just want prediction. Second, if a model can't even predict well, it's hard to see how it could be right scientifically. Third, often the best way of checking a scientific model is to turn some of its implications into statistical predictions.

3.2 Errors, In and Out of Sample

With any predictive model, we can gauge how well it works by looking at its errors. We want these to be small; if they can't be small all the time we'd like them to be small on average. We may also want them to be patternless or unsystematic (because if there was a pattern to them, why not adjust for that, and make smaller mistakes). We'll come back to patterns in errors later, when we look at specification testing (Chapter 10). For now, we'll concentrate on the size of the errors.

To be a little more mathematical, we have a data set with points $\mathbf{z}_n = z_1, z_2, \ldots z_n$. (For regression problems, think of each data point as the pair of input and output values, so $z_i = (x_i, y_i)$, with x_i possibly a vector.) We also have various possible models, each with different parameter settings, conventionally written θ. For regression, θ tells us which regression function to use, so $m_\theta(x)$ or $m(x;\theta)$ is the prediction we make at point x with parameters set to θ. Finally, we have a **loss function** L which

[1] We can be pretty confident of this, because we use our parameter estimates to send our robots to Mars, and they get there.

tells us how big the error is when we use a certain θ on a certain data point, $L(z, \theta)$. For mean-squared error, this would just be

$$L(z, \theta) = (y - m_\theta(x))^2 \tag{3.1}$$

But we could also use the mean absolute error

$$L(z, \theta) = |y - m_\theta(x)| \tag{3.2}$$

or many other loss functions. Sometimes we will actually be able to measure how costly our mistakes are, in dollars or harm to patients. If we had a model which gave us a distribution for the data, then $p_\theta(z)$ would a probability density at z, and a typical loss function would be the negative log-likelihood, $-\log m_\theta(z)$. No matter what the loss function is, I'll abbreviate the sample average of the loss over the whole data set by $L(\mathbf{z}_n, \theta)$.

What we would like, ideally, is a predictive model which has zero error on future data. We basically never achieve this:

- The world just really is a noisy and stochastic place, and this means even the true, ideal model has non-zero error.[2] This corresponds to the first, σ_x^2, term in the bias-variance decomposition, Eq. 1.26 from Chapter 1.
- Our models are usually more or less **mis-specified**, or, in plain words, wrong. We hardly ever get the functional form of the regression, the distribution of the noise, the form of the causal dependence between two factors, etc., *exactly* right.[3] This is the origin of the bias term in the bias-variance decomposition. Of course we can get any of the details in the model specification *more or less* wrong, and we'd prefer to be less wrong.
- Our models are never perfectly estimated. Even if our data come from a perfect IID source, we only ever have a finite sample, and so our parameter estimates are (almost!) never quite the true, infinite-limit values. This is the origin of the variance term in the bias-variance decomposition. But as we get more and more data, the sample should become more and more representative of the whole process, and estimates should converge too.

So, because our models are flawed, we have limited data and the world is stochastic, we cannot expect even the best model to have zero error. Instead, we would like to minimize the **expected error**, or **risk**, or **generalization error**, on new data.

What we would like to do is to minimize the risk or expected loss

$$\mathbf{E}[L(Z, \theta)] = \int L(z, \theta) p(z) dz \tag{3.3}$$

[2] This is so even if you believe in some kind of ultimate determinism, because the variables we plug in to our predictive models are not complete descriptions of the physical state of the universe, but rather immensely coarser, and this coarseness shows up as randomness.

[3] Except maybe in fundamental physics, and even there our predictions are about our fundamental theories *in the context of experimental set-ups*, which we never model in complete detail.

To do this, however, we'd have to be able to calculate that expectation. Doing that would mean knowing the distribution of Z — the joint distribution of X and Y, for the regression problem. Since we don't know the true joint distribution, we need to approximate it somehow.

A natural approximation is to use our training data $\mathbf{z}_n$. For each possible model θ, we can could calculate the sample mean of the error on the data, $\overline{L}(\mathbf{z}_n, \theta)$, called the **in-sample loss** or the **empirical risk**. The simplest strategy for estimation is then to pick the model, the value of θ, which minimizes the in-sample loss. This strategy is imaginatively called **empirical risk minimization**. Formally,

$$\widehat{\theta_n} \equiv \operatorname*{argmin}_{\theta \in \Theta} \overline{L}(\mathbf{z}_n, \theta) \tag{3.4}$$

This means picking the regression which minimizes the sum of squared errors, or the density with the highest likelihood[4]. This what you've usually done in statistics courses so far, and it's very natural, but it does have some issues, notably optimism and over-fitting.

The problem of optimism comes from the fact that our training data isn't perfectly representative. The in-sample loss is a sample average. By the law of large numbers, then, we anticipate that, for each θ,

$$\overline{L}(\mathbf{z}_n, \theta) \to \mathbf{E}\left[L(Z, \theta)\right] \tag{3.5}$$

as $n \to \infty$. This means that, with enough data, the in-sample error is a good approximation to the generalization error of any given model θ. (Big samples are representative of the underlying population or process.) But this does *not* mean that the in-sample performance of $\hat{\theta}$ tells us how well it will generalize, because we purposely picked it to match the training data $\mathbf{z}_n$. To see this, notice that the in-sample loss equals the risk plus sampling noise:

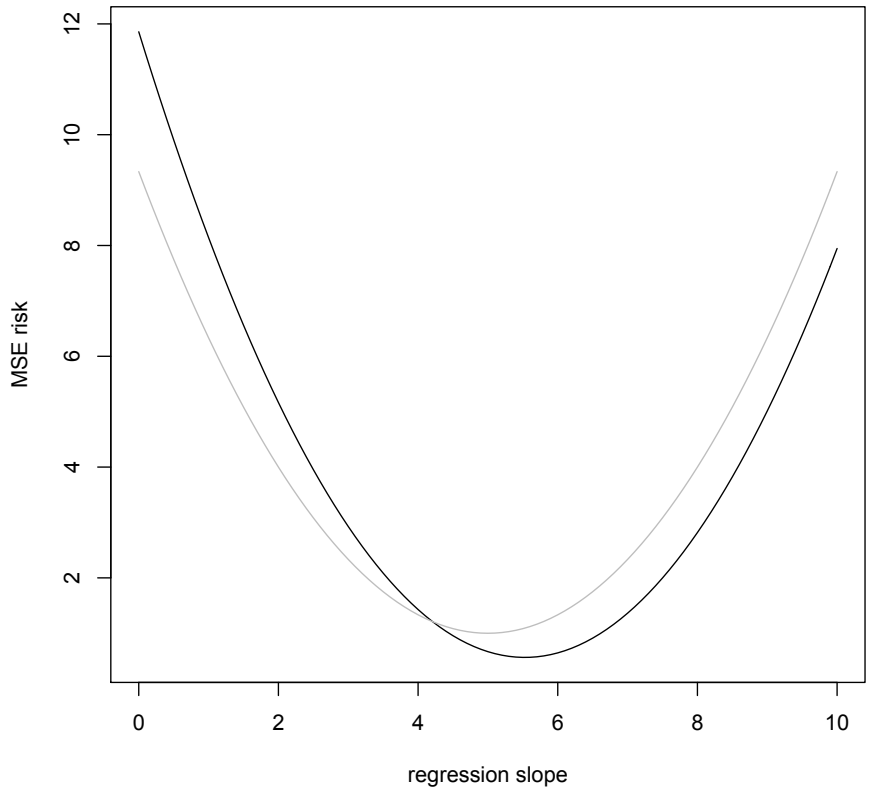
$$\overline{L}(\mathbf{z}_n, \theta) = \mathbf{E}\left[L(\mathbf{Z}, \theta)\right] + \eta_n(\theta) \tag{3.6}$$

Here $\eta(\theta)$ is a random term which has mean zero, and represents the effects of having only a finite quantity of data, of size n, rather than the complete probability distribution. (I write it $\eta_n(\theta)$ as a reminder that different values of θ are going to be affected differently by the same sampling fluctuations.) The problem, then, is that the model which minimizes the in-sample loss could be one with good generalization performance ($\mathbf{E}\left[L(\mathbf{Z}, \theta)\right]$ is small), or it could be one which got very lucky ($\eta_n(\theta)$ was large and negative):

$$\widehat{\theta_n} = \operatorname*{argmin}_{\theta \in \Theta} \left(\mathbf{E}\left[L(Z, \theta)\right] + \eta_n(\theta)\right) \tag{3.7}$$

We only want to minimize $\mathbf{E}\left[L(Z, \theta)\right]$, but we can't separate it from $\eta_n(\theta)$, so we're almost surely going to end up picking a $\widehat{\theta_n}$ which was more or less lucky ($\eta_n < 0$) as well as good ($\mathbf{E}\left[L(Z, \theta)\right]$ small). This is the reason why picking the model which best fits the data tends to exaggerate how well it will do in the future (Figure 3.1).

[4]Remember, maximizing the likelihood is the same as maximizing the log-likelihood, because log is an increasing function. Therefore maximizing the likelihood is the same as *minimizing* the *negative* log-likelihood.




```
n<-20; theta<-5
x<-runif(n); y<-x*theta+rnorm(n)
empirical.risk <- function(b) { mean((y.emp-b*x.emp)^2) }
true.risk <- function(b) { 1 + (theta-b)^2*(0.5^2+1/12) }
curve(Vectorize(empirical.risk)(x),from=0,to=2*theta,
  xlab="regression slope",ylab="MSE risk")
curve(true.risk,add=TRUE,col="grey")
```

Figure 3.1: Plots of empirical and generalization risk for a simple case of regression through the origin, $Y = \theta X + \epsilon$, $\epsilon \sim \mathcal{N}(0,1)$, with the true $\theta = 5$, and $X \sim \text{Unif}(0,1)$. The black curve is the mean squared error on one particular training sample (of size $n = 20$) as we vary the guessed slope; here the minimum is at $\hat{\theta} = 5.53$. The grey curve is the true or generalization risk. (See Exercise 2.) The gap between the grey and the black curves is what the text calls $\eta_n(\theta)$.

Again, by the law of large numbers $\eta_n(\theta) \to 0$ for each θ, but now we need to worry about how fast it's going to zero, and whether that rate depends on θ. Suppose we knew that $\min_\theta \eta_n(\theta) \to 0$, or $\max_\theta |\eta_n(\theta)| \to 0$. Then it would follow that $\eta_n(\widehat{\theta_n}) \to 0$, and the over-optimism in using the in-sample error to approximate the generalization error would at least be shrinking. If we knew how fast $\max_\theta |\eta_n(\theta)|$ was going to zero, we could even say something about how much bigger the true risk was likely to be. A lot of more advanced statistics and machine learning theory is thus about uniform laws of large numbers (showing $\max_\theta |\eta_n(\theta)| \to 0$) and rates of convergence.

Learning theory is a beautiful, deep, and practically important subject, but also a subtle and involved one. (See §3.6 for references.) To stick closer to analyzing real data, and to not turn this into an advanced probability class, I will only talk about some more-or-less heuristic methods, which are good enough for many purposes.

3.3 Over-Fitting and Model Selection

The big problem with using the in-sample error is related to over-optimism, but at once trickier to grasp and more important. This is the problem of **over-fitting**. To illustrate it, let's start with Figure 3.2. This has the twenty X values from a Gaussian distribution, and $Y = 7X^2 - 0.5X + \epsilon$, $\epsilon \sim \mathcal{N}(0,1)$. That is, the true regression curve is a parabola, with additive and independent Gaussian noise. Let's try fitting this — but pretend that we didn't know that the curve was a parabola. We'll try fitting polynomials of different orders in x — order 0 (a flat line), order 1 (a linear regression), order 2 (quadratic regression), up through order 9. Figure 3.3 shows the data with the polynomial curves, and Figure 3.4 shows the in-sample mean squared error as a function of the order of the polynomial.

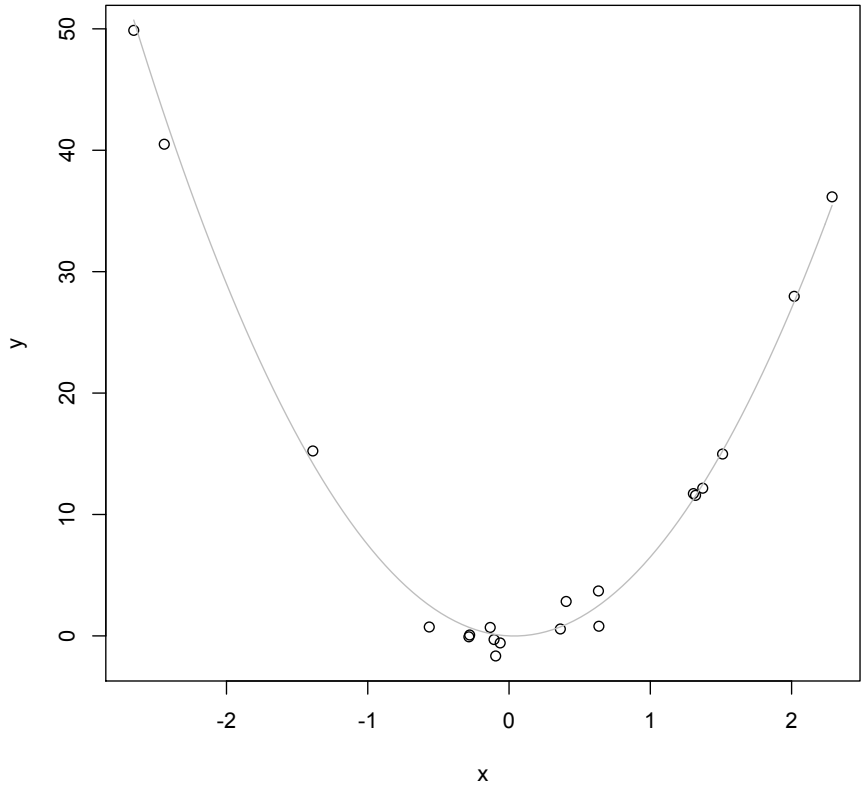
Notice that the in-sample error goes down as the order of the polynomial increases; it has to. Every polynomial of order p is also a polynomial of order $p+1$, so going to a higher-order model can only reduce the in-sample error. Quite generally, in fact, as one uses more and more complex and flexible models, the in-sample error will get smaller and smaller.[5]

Things are quite different if we turn to the generalization error. In principle, I could calculate that for any of the models, since I know the true distribution, but it would involve calculating things like $\mathbf{E}\left[X^{18}\right]$, which won't be very illuminating. Instead, I will just draw a lot more data from the same source, twenty thousand data points in fact, and use the error of the old models on the new data as their generalization error[6]. The results are in Figure 3.5.

What is happening here is that the higher-order polynomials — beyond order 2 — are not just a *little* optimistic about how well they fit, they are *wildly* over-optimistic. The models which seemed to do notably better than a quadratic actually do much,

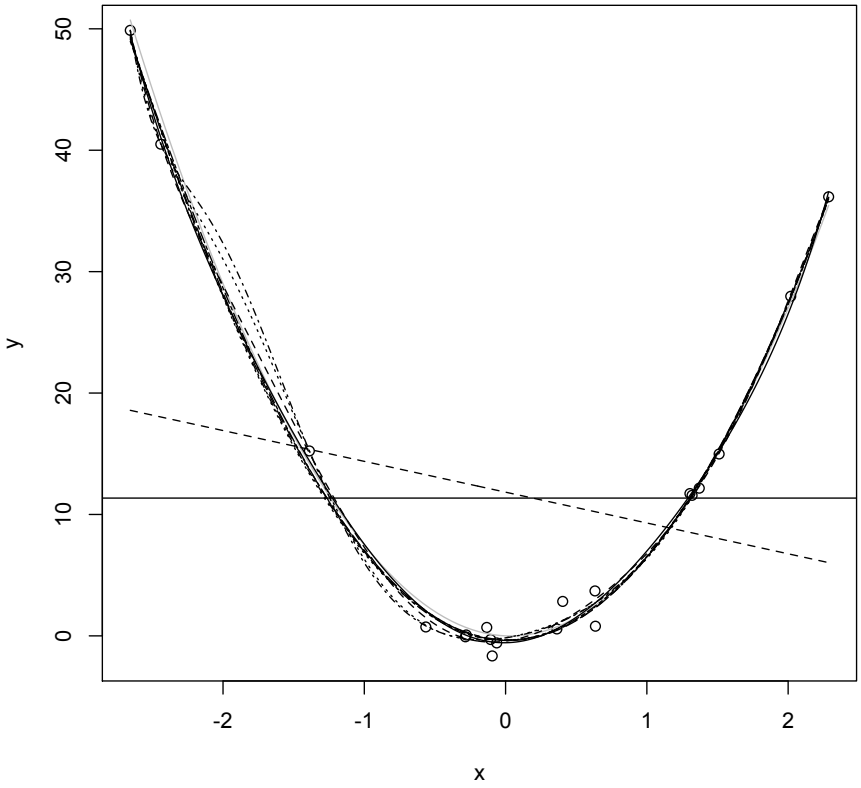
[5] In fact, since there are only 20 data points, they could all be fit exactly if the order of the polynomials went up to 19. (Remember that any two points define a line, any three points a parabola, etc. — $p+1$ points define a polynomial of order p which passes through them.)

[6] This works, yet again, because of the law of large numbers. In Chapters 5 and especially 6, we will see much more about replacing complicated probabilistic calculations with simple simulations.



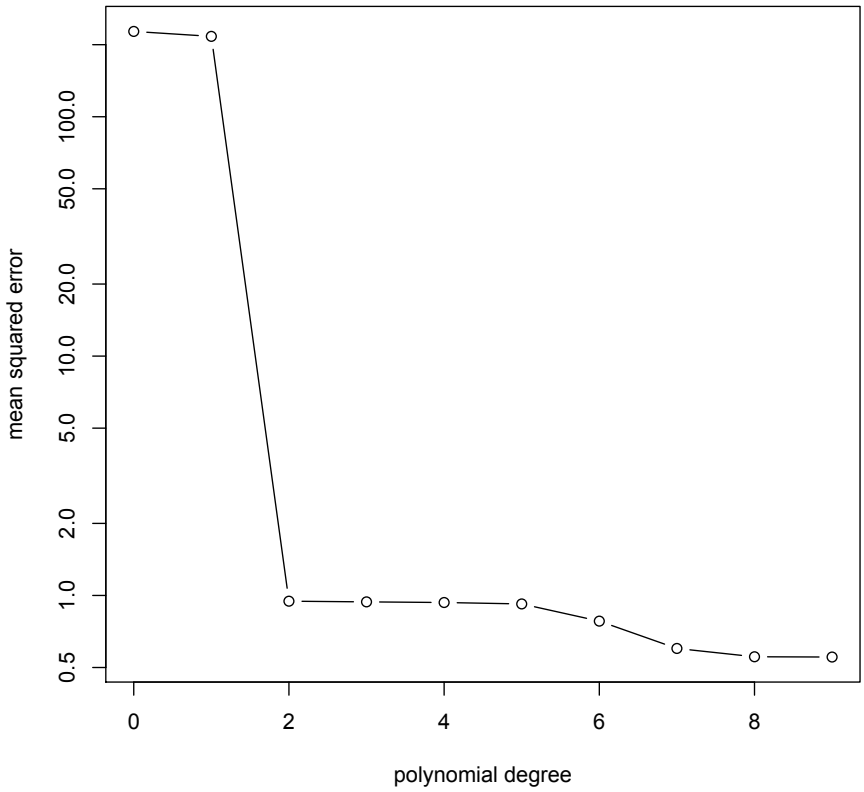

```
plot(x,y2)
curve(7*x^2-0.5*x,add=TRUE,col="grey")
```

Figure 3.2: Scatter-plot showing sample data and the true, quadratic regression curve (grey parabola).



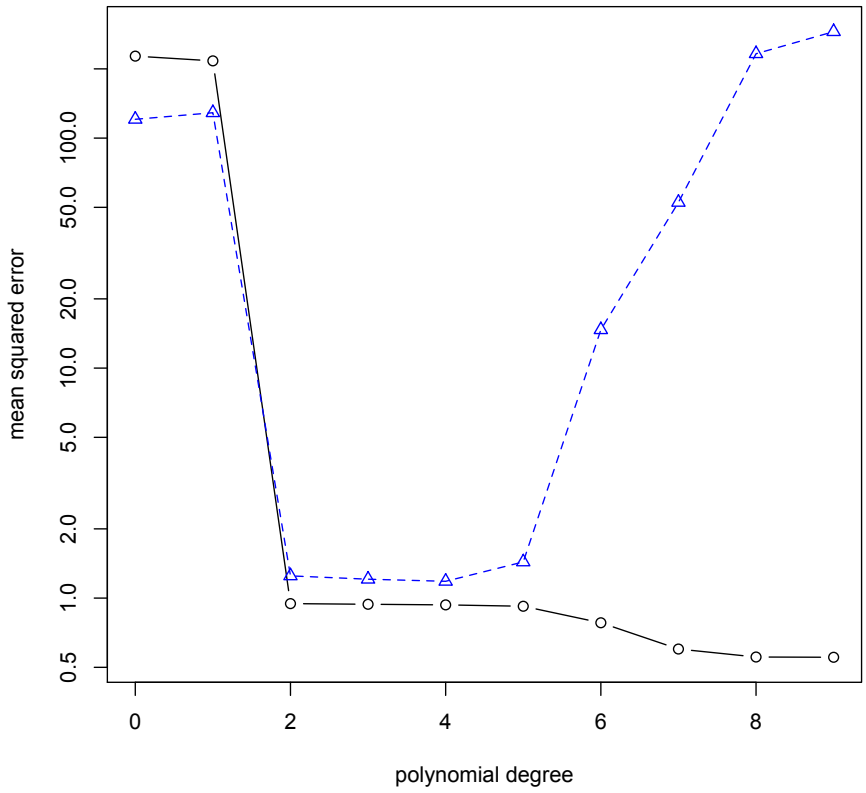

```
poly.formulae <- c("y~1", paste("y ~ poly(x,", 1:9, ")", sep=""))
poly.formulae <- sapply(poly.formulae, as.formula)
df.plot <- data.frame(x=seq(min(x),max(x),length.out=200))
fitted.models <- list(length=length(poly.formulae))
for (model_index in 1:length(poly.formulae)) {
  fm <- lm(formula=poly.formulae[[model_index]])
  lines(df.plot$x, predict(fm,newdata=df.plot),lty=model_index)
  fitted.models[[model_index]] <- fm
}
```

Figure 3.3: Twenty training data points (dots), and ten different fitted regression lines (polynomials of order 0 to 9, indicated by different line types). R NOTES: The `poly` command constructs orthogonal (uncorrelated) polynomials of the specified degree from its first argument; regressing on them is conceptually equivalent to regressing on $1, x, x^2, \ldots x^{\text{degree}}$, but more numerically stable. (See `?poly`.) This builds a vector of model formulae and then fits each one in turn, storing the fitted models in a new list.




```
mse.q <- sapply(fitted.models, function(mdl) { mean(residuals(mdl)^2) })
plot(0:9,mse.q,type="b",xlab="polynomial degree",ylab="mean squared error",
     log="y")
```

Figure 3.4: Empirical MSE vs. degree of polynomial for the data from the previous figure. Note the logarithmic scale for the vertical axis.




```
x.new = rnorm(2e4); y.new = 7*x.new^2 - 0.5*x.new + rnorm(2e4)
gmse <- function(mdl) { mean((y.new - predict(mdl, data.frame(x=x.new))^2) }
gmse.q <- sapply(fitted.models, gmse)
plot(0:9,mse.q,type="b",xlab="polynomial degree",
     ylab="mean squared error",log="y",ylim=c(min(mse.q),max(gmse.q)))
lines(0:9,gmse.q,lty=2,col="blue")
points(0:9,gmse.q,pch=24,col="blue")
```

Figure 3.5: In-sample error (black dots) compared to generalization error (blue triangles). Note the logarithmic scale for the vertical axis.

much worse. If we picked a polynomial regression model based on in-sample fit, we'd chose the highest-order polynomial available, and suffer for it.

In this example, the more complicated models — the higher-order polynomials, with more terms and parameters — were not actually fitting the *generalizable* features of the data. Instead, they were fitting the sampling noise, the accidents which don't repeat. That is, the more complicated models **over-fit** the data. In terms of our earlier notation, η is bigger for the more flexible models. The model which does best here is the quadratic, because the true regression function happens to be of that form. The more powerful, more flexible, higher-order polynomials were able to get closer to the training data, but that just meant matching the noise better. In terms of the bias-variance decomposition, the bias shrinks with the model order, but the variance of estimation grows.

Notice that the models of order 0 and order 1 also do worse than the quadratic model — their problem is not over-fitting but *under*-fitting; they would do better if they were more flexible. Plots of generalization error like this usually have a minimum. If we have a choice of models — if we need to do **model selection** — we would like to find the minimum. Even if we do not have a *choice* of models, we might like to know how big the gap between our in-sample error and our generalization error is likely to be.

There is nothing special about polynomials here. All of the same lessons apply to variable selection in linear regression, to k-nearest neighbors (where we need to choose k), to kernel regression (where we need to choose the bandwidth), and to other methods we'll see later. In every case, there is going to be a minimum for the generalization error curve, which we'd like to find.

(A minimum with respect to what, though? In Figure 3.5, the horizontal axis is the model order, which here is the number of parameters (minus one). More generally, however, what we care about is some measure of how complex the model space is, which is not necessarily the same thing as the number of parameters. What's more relevant is how flexible the class of models is, how many different functions it can approximate. Linear polynomials can approximate a smaller set of functions than quadratics can, so the latter are more complex, or have higher **capacity**. More advanced learning theory has a number of ways of quantifying this, but the details get pretty arcane, and we will just use the concept of complexity or capacity informally.)

3.4 Cross-Validation

The most straightforward way to find the generalization error would be to do what I did above, and to use fresh, independent data from the same source — a **testing** or **validation** data-set. Call this $\mathbf{z}'_m$, as opposed to our training data $\mathbf{z}_n$. We fit our model to $\mathbf{z}_n$, and get $\widehat{\theta_n}$. The loss of this on the validation data is

$$\mathbf{E}\left[L(Z, \widehat{\theta_n})\right] + \eta'_m(\widehat{\theta_n}) \tag{3.8}$$

where now the sampling noise on the *validation* set, η'_m, is independent of $\widehat{\theta_m}$. So this gives us an unbiased estimate of the generalization error, and, if m is large, a precise

```
A_vs_B <- sample(rep(c("A","B"),length.out=nrow(housing))
half_A <- which(A_vs_B=="A"); half_B <- which(A_vs_B=="B")
small_formula = "Median_house_value ~ Median_household_income"
large_formula = "Median_house_value ~ Median_household_income + Median_rooms"
small_formula <- as.formula(small_formula)
large_formula <- as.formula(large_formula)
(mAsmall <- lm(small_formula,data=housing,subset=half_A))
(mBsmall <- lm(small_formula,data=housing,subset=half_B))
(mAlarge <- lm(large_formula,data=housing,subset=half_A))
(mBlarge <- lm(large_formula,data=housing,subset=half_B))
in.sample.mse <- function(model) { mean(residuals(model)^2) }
in.sample.mse(mAsmall); in.sample.mse(mAlarge)
in.sample.mse(mBsmall); in.sample.mse(mBlarge)
new.sample.mse <- function(model,rows) {
   test <- housing[rows,]
   predictions <- predict(model,newdata=test)
   return(mean((test$Median_house_value - predictions)^2))
}
new.sample.mse(mAsmall,half_B); new.sample.mse(mBsmall,half_A)
new.sample.mse(mBlarge,half_A); new.sample.mse(mAlarge,half_B)
```

Code Example 1: Code used to generate the numbers in Figure 3.6. (Code used to display values from the data frames omitted.)

one. If we need to select one model from among many, we can pick the one which does best on the validation data, with confidence that we are not just over-fitting.

The problem with this approach is that we absolutely, positively, cannot use any of the validation data in estimating the model. Since collecting data is expensive — it takes time, effort, and usually money, organization, effort and skill — this means getting a validation data set is expensive, and we often won't have that luxury.

3.4.1 Data-set Splitting

The next logical step, however, is to realize that we don't strictly need a separate validation set. We can just take our data and *split* it ourselves into training and testing sets. If we divide the data into two parts at random, we ensure that they have (as much as possible) the same distribution, and that they are independent of each other. Then we can act just as though we had a real validation set. Fitting to one part of the data, and evaluating on the other, gives us an unbiased estimate of generalization error. Of course it doesn't matter which half of the data is used to train and which half is used to test, so we can do it both ways and average.

Figure 3.6 illustrates the idea with a bit of the data and linear models from §31, and Code Example 1 shows the code used to make Figure 3.6.

[[TODO: Turn figure to portrait mode, make everything bigger, space out, add arrows to guide eye through data flow]]

	Median_house_value	Median_household_income	Median_rooms
1	909600	111667	6.0
2	748700	66094	4.6
3	773600	87306	5.0
4	579200	62386	4.5
...	...	...	...
10605	253400	71638	6.6

(A)

	Median_house_value	Median_household_income	Median_rooms
2	748700	66094	4.6
3	773600	87306	5.0
5	480800	55658	4.8
6	460800	38646	4.3
...	...	...	...
10605	253400	71638	6.6

(B)

	Median_house_value	Median_household_income	Median_rooms
1	909600	111667	6.0
4	579200	62386	4.5
7	473500	52837	4.3
8	439300	59091	4.4
...	...	...	...
10604	209500	56667	6.0

(A)

	$\widehat{\beta}_{\text{intercept}}$	$\widehat{\beta}_{\text{income}}$	$\widehat{\beta}_{\text{rooms}}$	MSE
Income only	$(2.74 \pm 0.56) \times 10^4$	5.252 ± 0.085	NA	2.62×10^{10}
Income + Rooms	$(4.772 \pm 0.093) \times 10^5$	7.748 ± 0.081	$(-1.125 \pm 0.020) \times 10^5$	1.66×10^{10}

(B)

	$\widehat{\beta}_{\text{intercept}}$	$\widehat{\beta}_{\text{income}}$	$\widehat{\beta}_{\text{rooms}}$	MSE
Income only	$(3.99 \pm .55) \times 10^4$	4.99 ± 0.8	NA	2.59×10^{10}
Income + Rooms	$(5.040 \pm 0.089) \times 10^5$	7.609 ± 0.079	$(-1.162 \pm 0.020) \times 10^5$	1.56×10^{10}

	MSE($A \to B$)	MSE($B \to A$)	average
Income only	2.60×10^{10}	2.62×10^{10}	2.61×10^{10}
Income + Rooms	1.56×10^{10}	1.67×10^{10}	1.61×10^{10}

Figure 3.6: Example of data-set splitting. The top table shows three columns and seven rows of the housing-price data used in §31. This is then randomly split into two equally sized parts (tables in the next row). I estimate a linear model which predicts house value from income alone, and another model which predicts from income and the median number of rooms, on each half (parameter estimates and in-sample MSEs in the third row). The fourth row shows the performance of each estimate on the other half of the data, and the average for each of the two models. Note that the larger model *always* has a lower in-sample error, whether or not it is really better, so the in-sample MSEs provide no *evidence* that we should use the larger model. Having a lower score under data-set splitting, however, *is* evidence that the larger model generalizes better. (For R commands used to get these numbers, see Code Example 1.) — Can you explain why the coefficient on the number of rooms is negative?

3.4.2 k-Fold Cross-Validation (CV)

The problem with data-set splitting is that, while it's an unbiased estimate of the risk, it is often a very noisy one. If we split the data evenly, then the test set has $n/2$ data points — we've cut in half the number of sample points we're averaging over. It would be nice if we could reduce that noise somewhat, especially if we are going to use this for model selection.

One solution to this, which is pretty much the industry standard, is what's called **k-fold cross-validation**. Pick a small integer k, usually 5 or 10, and divide the data at random into k equally-sized subsets. (The subsets are often called "folds".) Take the first subset and make it the test set; fit the models to the rest of the data, and evaluate their predictions on the test set. Now make the second subset the test set and the rest of the training sets. Repeat until each subset has been the test set. At the end, average the performance across test sets. (This is the same as data-set splitting if $k = 2$.) This is the cross-validated estimate of generalization error for each model. Model selection then picks the model with the smallest estimated risk.[7] Code Example 2 performs k-fold cross-validation for linear models specified by formulae.

The reason cross-validation works is that it uses the existing data to simulate the process of generalizing to new data. If the full sample is large, then even the smaller portion of it in the testing data is, with high probability, fairly representative of the data-generating process. *Randomly* dividing the data into training and test sets makes it very unlikely that the division is rigged to favor any one model class, over and above what it would do on real new data. Of course the original data set is never *perfectly* representative of the full data, and a smaller testing set is even less representative, so this isn't ideal, but the approximation is often quite good. It is especially good at getting the *relative* order of different models right, that is, at controlling over-fitting.[8]

Cross-validation is probably the most widely-used method for model selection, and for picking control settings, in modern statistics. There are circumstances where it can fail — especially if you give it *too many* models to pick among — but it's the first thought of seasoned practitioners, and it should be your first thought, too. The assignments to come will make you *very* familiar with it.

3.4.3 Leave-one-out Cross-Validation

Suppose we did k-fold cross-validation, but with $k = n$. Our testing sets would then consist of single points, and each point would be used in testing once. This is called **leave-one-out cross-validation**. It actually came before k-fold cross-validation, and has two advantages. First, it doesn't require any random number generation, or keeping track of which data point is in which subset. Second, and more importantly, because we are only testing on *one* data point, it's often possible to find what the

[7] A closely related procedure, sometimes also called "k-fold CV", is to pick $1/k$ of the data points at random to be the test set (using the rest as a training set), and then pick an *independent* $1/k$ of the data points as the test set, etc., repeating k times and averaging. The differences are subtle, but what's described in the main text makes sure that each point is used in the test set just once.

[8] The cross-validation score for the selected model still tends to be somewhat over-optimistic, because it's still picking the luckiest model — though the influence of luck is much attenuated. Tibshirani and Tibshirani (2009) provides a simple correction.

```
cv.lm <- function(data, formulae, nfolds=5) {
  data <- na.omit(data)
  formulae <- sapply(formulae, as.formula)
  response.name <- function(formula) { all.vars(formula)[1] }
  responses <- sapply(formulae, response.name)
  names(responses) <- as.character(formulae)
  n <- nrow(data)
  fold.labels <- sample(rep(1:nfolds, length.out=n))
  mses <- matrix(NA, nrow=nfolds, ncol=length(formulae))
  colnames <- as.character(formulae)
  for (fold in 1:nfolds) {
    test.rows <- which(fold.labels == fold)
    train <- data[-test.rows,]
    test <- data[test.rows,]
    for (form in 1:length(formulae)) {
       current.model <- lm(formula=formulae[[form]], data=train)
       predictions <- predict(current.model, newdata=test)
       test.responses <- test[,responses[form]]
       mses[fold, form] <- mean((test.responses - predictions)^2)
    }
  }
  return(colMeans(mses))
}
```

Code Example 2: Function to do k-fold cross-validation on linear models, given as a vector (or list) of model formulae. Note that this only returns the CV MSE, not the parameter estimates on each fold. See online for comments.

prediction on the left-out point would be by doing calculations on a model fit to the *whole* data. This means that we only have to fit each model once, rather than k times, which can be a big savings of computing time.

The drawback to leave-one-out CV is subtle but often decisive. Since each training set has $n-1$ points, any two training sets must share $n-2$ points. The models fit to those training sets tend to be strongly correlated with each other. Even though we are averaging n out-of-sample forecasts, those are correlated forecasts, so we are not really averaging away all that much noise. With k-fold CV, on the other hand, the fraction of data shared between any two training sets is just $\frac{k-2}{k-1}$, not $\frac{n-2}{n-1}$, so even though the number of terms being averaged is smaller, they are less correlated.

There are situations where this issue doesn't really matter, or where it's overwhelmed by leave-one-out's advantages in speed and simplicity, so there is certainly still a place for it, but one subordinate to k-fold CV.[9]

[[TODO: Appendix on AIC?]]

3.5 Warnings

Some caveats are in order.

1. All of these model selection methods aim at getting models which will generalize well to new data, *if it follows the same distribution* as old data. Generalizing well even when distributions change is a much harder and much less well-understood problem (Quiñonero-Candela *et al.*, 2009). It is particularly troublesome for a lot of applications involving large numbers of human beings, because society keeps changing all the time — it's natural for the variables to vary, but the *relationships* between variables also change. (That's history.)

2. All the model selection methods we have discussed aim at getting models which *predict well*. This is not necessarily the same as getting the *true theory of the world*. Presumably the true theory will also predict well, but the converse does not necessarily follow. We will see examples later where false but low-capacity models, because they have such low variance of estimation, actually out-predict correctly specified models.

3.5.1 Parameter Interpretation

The last item is worth elaborating on. In many situations, it is very natural to want to attach some substantive, real-world meaning to the parameters of our statistical model, or at least to some of them. I have mentioned examples above like astronomy, and it is easy to come up with many others from the natural sciences. This is also extremely common in the social sciences. It is fair to say that this is much less carefully attended to than it should be.

[9] At this point, it may be appropriate to say a few words about the Akaike information criterion, or AIC. AIC also tries to estimate how well a model will generalize to new data. One can show that, under standard assumptions, as the sample size gets large, leave-one-out CV actually gives the same estimate as AIC (Claeskens and Hjort, 2008, §2.9). However, there do not seem to be any situations where AIC works where leave-one-out CV does not work at least as well. So AIC should really be understood as a very fast, but often very crude, approximation to the more accurate cross-validation.

To take just one example, consider the paper "Luther and Suleyman" by Prof. Murat Iyigun (Iyigun, 2008). The major idea of the paper is to try to help explain why the Protestant Reformation was not wiped out during the European wars of religion (or alternately, why the Protestants did not crush all the Catholic powers), leading western Europe to have a mixture of religions, with profound consequences. Iyigun's contention is that all of the Christians were so busy fighting the Ottoman Turks, or perhaps so afraid of what might happen if they did not, that conflicts among the European Christians were suppressed. To quote his abstract:

> at the turn of the sixteenth century, Ottoman conquests lowered the number of all newly initiated conflicts among the Europeans roughly by 25 percent, while they dampened all longer-running feuds by more than 15 percent. The Ottomans' military activities influenced the length of intra-European feuds too, with each Ottoman-European military engagement shortening the duration of intra-European conflicts by more than 50 percent.

To back this up, and provide those quantitative figures, Prof. Iyigun estimates linear regression models, of the form[10]

$$Y_t = \beta_0 + \beta_1 X_t + \beta_2 Z_t + \beta_3 U_t + \epsilon_t \tag{3.9}$$

where Y_t is "the number of violent conflicts initiated among or within continental European countries at time t"[11], X_t is "the number of conflicts in which the Ottoman Empire confronted European powers at time t", Z_t is "the count at time t of the newly initiated number of Ottoman conflicts with others and its own domestic civil discords", U_t is control variables reflecting things like the availability of harvests to feed armies, and ϵ_t is Gaussian noise.

The qualitative idea here, about the influence of the Ottoman Empire on the European wars of religion, has been suggested by quite a few historians before[12]. The point of this paper is to support this rigorously, and make it precise. That support and precision requires Eq. 3.9 to be an accurate depiction of at least part of the process which lead European powers to fight wars of religion. Prof. Iyigun, after all, wants to be able to interpret a negative estimate of β_1 as saying that fighting off the Ottomans *kept* Christians from fighting each other. If Eq. 3.9 is inaccurate, if the model is badly mis-specified, however, β_1 becomes the best approximation to the truth within a systematically wrong model, and the support for claims like "Ottoman conquests lowered the number of all newly initiated conflicts among the Europeans roughly by 25 percent" drains away.

To back up the use of Eq. 3.9, Prof. Iyigun looks at a range of slightly different linear-model specifications (e.g., regress the number of intra-Christian conflicts in year t on the number of Ottoman attacks in year $t-1$), and slightly different methods of estimating the parameters. What he does *not* do is look at the other implications of the model: that residuals should be (at least approximately) Gaussian, that they

[10] His Eq. 1 on pp. 1473; I have modified the notation to match mine.

[11] In one part of the paper; he uses other dependent variables elsewhere.

[12] See §1–2 of Iyigun (2008), and MacCulloch (2004, *passim*).

should be unpredictable from the regressor variables. He does not look at whether the relationships he thinks are linear really are linear (see Chapters 4, 9, and 10). He does not try to simulate his model and look at whether the patterns of European wars it produces resemble actual history (see Chapter 5). He does not try to check whether he has a model which really supports causal inference, though he has a causal question (see Part III).

I do not say any of this to denigrate Prof. Iyigun. His paper is actually *much better* than most quantitative work in the social sciences. This is reflected by the fact that it was published in the *Quarterly Journal of Economics*, one of the most prestigious, and rigorously-reviewed, journals in the field. The point is that by the end of this course, you will have the tools to do *better*.

3.6 Further Reading

Some comparatively easy starting points on statistical learning theory are Kearns and Vazirani (1994), Cristianini and Shawe-Taylor (2000) and Mohri *et al.* (2012). At a more advanced level, look at the tutorial papers by Bousquet *et al.* (2004); von Luxburg and Schölkopf (2008), or the textbooks by Vidyasagar (2003) and by Anthony and Bartlett (1999) (the latter is much more general than its title suggests), or read the book by Vapnik (2000) (one of the founders). Hastie *et al.* (2009), while invaluable, is much more oriented towards models and practical methods than towards learning *theory*.

Cross-validation goes back in statistical practice for many decades, though often as a very informal tool. One of the first important papers on the subject was Stone (1974), goes over the earlier history. Arlot and Celisse (2010) is a good recent review.

On model selection in general, the best recent summary is the book by Claeskens and Hjort (2008); it is more theoretically demanding than this book, but includes many real-data examples.

White (1994) is a thorough treatment of parameter estimation in models which may be mis-specified, and some general tests for mis-specification. It also briefly discusses the interpretation of parameters in mis-specified models. That very important topic deserves a more in-depth treatment, but I don't know of one.

3.7 Exercises

1. Suppose that one of our model classes contains the true and correct model, but we also consider more complicated and flexible model classes. Does the bias-variance trade-off mean that we will over-shoot the true model, and always go for something more flexible, when we have enough data? (This would mean there was such a thing as *too much* data to be reliable.)

2. Derive the formula for the generalization risk in the situation depicted in Figure 3.1, as given by the `true.risk` function in the code for that figure. In particular, explain to yourself where the constants 0.5^2 and $1/12$ come from.