Computational Statistics 12.1: Supervised learning

Supervised learning

“Big data” again

Out-of-sample testing
- **Vs Statistics**
- **Regression example**
- **Example 2**
- **Example 2b**

Cross-validation

<

>

Supervised learning

Overview

This lesson introduces supervised learning and out-of-sample validation.

Objectives

After completing this module, students should be able to:

1. Employ out-of-sample testing to validate models.

Readings

Lander, 18.3

“Big data” again

In the previous module, we looked at “big data” problems with a large p (large number of variables), but no explicit dependent variable. Instead, we just wanted to know what underlying patterns – factors or clusters – could best describe our data, and might explain its complexity in terms of simpler origins.

In this module, we examine similarly big data – high p – but now we are interested in a problem more like what we saw in multiple regression: how do we model some dependent variable as a function of dozens or even hundreds of others? For various reasons, our previous regression methods break down, either failing entirely, or failing to find the true underlying correlations and instead “over-fitting” (a concept we will return to) the dependent variable on the noise in our independent variables rather than finding the signal underneath.

In a sense, our problem here is similar to the one we confronted in our discussion of variable selection – how out of a (perhaps large) number of variables do we find the ones that best predict y? Or is there some way to use them all? And if y is a binary, how can we best model that binary in order to best predict whether some observation’s y value will be a 1 or a 0?

As we will see, there are two families of methods we will be examining in this module, one suited to modeling a continuous y, which resembles our existing regression methods; and a second method for modeling a binary y, which although it is very different in appearance from the logit, is similarly concerned with categorization rather than continuous y.

Out-of-sample testing

The fundamental idea behind modeling data is to detect underlying relationships that are true independent of whatever random sample from the population we may have taken. If we take one same and find some relationship between y and x (eg, $y = 1 + 2x$), we would hope that that relationship would continue to hold even if we draw a second sample from the population and predict the new $\hat{y}$ based on our estimated beta coefficients and the new x values in that second sample. If we found that in our new sample our predicted y values $\hat{y} = 1 + 2x$ were a poor match for the true y values in that sample, then we would suspect we had done something wrong in estimating our coefficients in the first sample: either we had made some mistake, or had failed to notice that those coefficients had large standard errors and, being not significantly different from 0, may have been entirely mistaken in the first place, and in truth there was no real relationship between y and x at all.

This is what is known as out-of-sample validation, and it is an essential method for testing whether a model has identified true relationships underneath the data, or merely spurious, random, or erroneous relationships. The essential idea is:

1. Draw a sample from the population.
2. Estimate a model of Y as a function of the X variables, yielding some estimated parameters β.
3. Draw a second sample from the population, and from that sample, use the X values plus the estimated coefficients from (2) to predict $\hat{y}$ for those new observations.
4. Compare those predicted $\hat{y}$ to the true y values for the second sample. If those predictions are better than we might have done by chance (eg, just guessing the mean $\bar{y}$ for every y_i), then our model estimated in (2) is general to the population, and not just a chance artifact of the first sample.

In fact, 4 can be expanded as a way of comparing multiple different methods: the better one method predicts the y in sample 2 using coefficients derived from sample 1, the better that method has managed to extract the hidden patterns connecting y to X in the population at large. Thus if we look at the fit between $\hat{y}$ and y in the out-sample (sample 2), we can rank different methods for generating $\hat{y}$ based on how well they have done. This is a better method than looking at the fit between $\hat{y}$ and y using just the in-sample (sample 1), because it is often the case that you can predict y quite well on some sample using the coefficients derived from that sample, but when you try to extend it to a new sample the model fails because it was over-fitted to the first sample. We will see an example of this momentarily.

Vs Statistics

One question one might ask is, why do we need this out-of-sample testing when the whole point of our statistical tests for the signficance of our β coefficients was to determine whether they were signficant or not? That is, our t statistics and p values for each β tells us whether those coefficients are signficant for the population as a whole, not just for our sample. So we shouldn't need an out-of-sample test if we believe our statistical results.

This is true: for multiple regression, the statistics are well-developed, and we do have the statstical test to answer this. In fact, if we estimate β using a sample (as we usually do), and get a 95% CI for β, then if we take another 100 complete samples each of size n and estimate β for each one, we should find that approximately 95 those β values fall within the 95% CI for our first β. That is, the statistical standard error predicts exactly what we would get if we did run multiple out-of-sample tests.

However, many other methods are more complex than multiple regression, and we cannot derive the complete statistics for errors and p values on their coefficients. As a results, we need some more theory-independent method for testing the validity of those models, and evaluating their relative accuracies. This is the value of out-of-sample testing: without knowing anything of the statistics or errors of a model or its parameters, we can still test its validity to the population as a whole by out-of-sample testing: fit the model on sample 1, and then use the fitted coefficients (or other parameters) from sample 1 to predict $\hat{y}$ in sample 2 using the new x values in sample 2. In fact, this is a black-box procedure: we don't have to know anything about the method that was used to fit the model in sample 1 or to predict $\hat{y}$ in sample 2. And indeed, many computer-science competitions work this way: they give the contestants a first sample to play with as they like, tweaking their models to fit the data as best they can, and then they give the contestants only the x values from a new sample, ask them to predict $\hat{y}$, and then at the end reveal whose $\hat{y}$ best match the true y values for that sample. (And somewhat recusively, the best contestants usually win by taking the sample they have been given, dividing *it* up into in-samples and out-samples, and running mini-contests on themselves using the data they have to best hone their own models before tackling the real contest test data.)

Regression example

Let's examine a simple example from regression. Consider the following simulation, which consists of 20 observations, a random x variable, and a y variable that we construct to be $y = 3x + \epsilon$, where as usual ϵ is random noise:

```
set.seed(2)
x <- rnorm(20)
y <- 3*x + rnorm(20)
dataf <- data.frame(y=y,x=x)
```

We divide that data into two halves, the first 10 observations and the second.

```
insample <- dataf[1:10,]
outsample <- dataf[11:20,]
```

Note that since there's no order to the observations, it doesn't matter whether we take the first 10 and second 10, or every other observation, or some random subset; nor does it matter for our purposes here whether the two samples are the same size or different sizes.

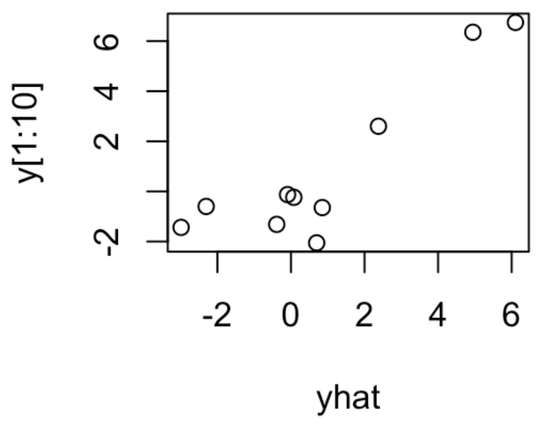
Next we regress y on x using just the in-sample observations, and show the regression results – which correctly capture the model we constructed at the beginning ($y = 3x + \epsilon$)

```
lmout <- lm(y~x,data=insample)
summary(lmout)
```

```


Call:
lm(formula = y ~ x, data = insample)

Residuals:
     Min       1Q   Median       3Q      Max 
-2.75327 -0.77507  0.09438  1.21652  1.70463 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   0.3125     0.4935   0.633 0.544296    
x             2.9178     0.5151   5.664 0.000473 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.522 on 8 degrees of freedom
Multiple R-squared:  0.8004,	Adjusted R-squared:  0.7755 
F-statistic: 32.09 on 1 and 8 DF,  p-value: 0.0004735
```

Finally we predict $\hat{y} = \hat{\beta}x$ using the built-in `predict` function, and then we plot those predicted values against the true y values:

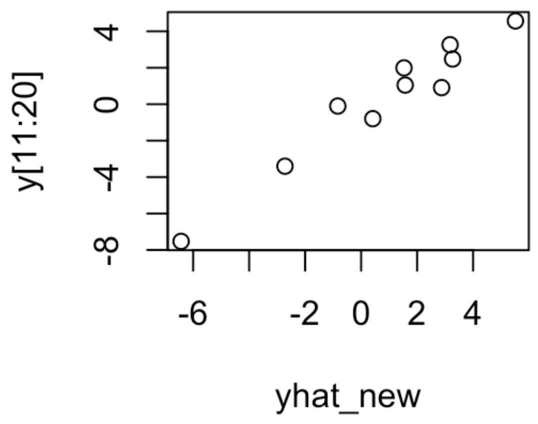
```
yhat <- predict(lmout)
plot(yhat,y[1:10])
```



As we can see, they are pretty accurate! The model, at least when fit on the in-sample and then tested on the in-sample, does pretty well.

How well do we do using the second, out-sample? Well, we can use `predict` again, which takes the $\hat{\beta}$ that we estimated in `lmout` and uses the new data we specify with `newdata=`, where here we use the second 10 observations:

```
yhat_new <- predict(lmout,newdata=outsample)
plot(yhat_new,y[11:20])
```



Here too we do fairly well: the out-sample $\hat{y}$ values predicted from the out-sample x values and the in-sample $\hat{\beta}$ coefficient nevertheless match fairly closely the true out-sample y values. That is, our original regression captured something truly in the population (that is, the model we specified for all the data, $y = 3x + \epsilon$), not just a random characteristic of our in-sample data.

Example 2

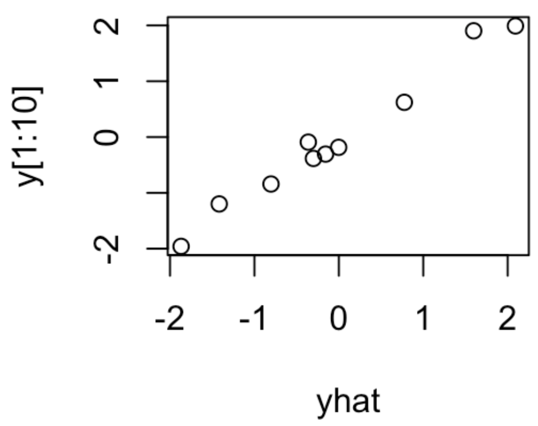
So what’s the alternative? What does it look like to appear to do well in-sample, but do poorly out-of-sample? As we said, this is something that comes up a lot when you have lots of independent variables, so we can illustrate this case with a similar data structure, but with lots of independent variables. Consider the following simulated data:

```
y <- rnorm(20)
x1 <- rnorm(20)
x2 <- rnorm(20)
x3 <- rnorm(20)
x4 <- rnorm(20)
x5 <- rnorm(20)
x6 <- rnorm(20)
x7 <- rnorm(20)
x8 <- rnorm(20)
dataf=data.frame(y=y,x1=x1,x2=x2,x3=x3,x4=x4,x5=x5,x6=x6,x7=x7,x8=x8)
insample <- dataf[1:10,]
outsample <- dataf[11:20,]
```

Note what is odd about this data: y is *not* a function of any of the x variables. All the variables, y included, are purely random noise. The values of x should offer *no* power to predict y, since they have nothing to do with each other.

But what happens if we regress y on our x’s?

```
lmout <- lm(y~x1+x2+x3+x4+x5+x6+x7+x8,data=insample)
yhat <- predict(lmout)
plot(yhat,y[1:10])
```



Surprisingly, it looks like we do a pretty good job matching our predicted $\hat{y}$ to the real y values, even though there is no connection between our x’s, our β values (which are purely random) and our y. This is because we have over-fit our data: we match the y well with our x variables just due to the fact that we have a lot of them, which allow us to pick up on chance variations in y due to chance variations in the x values.

Example 2b

Note that the regression itself is not fooled:

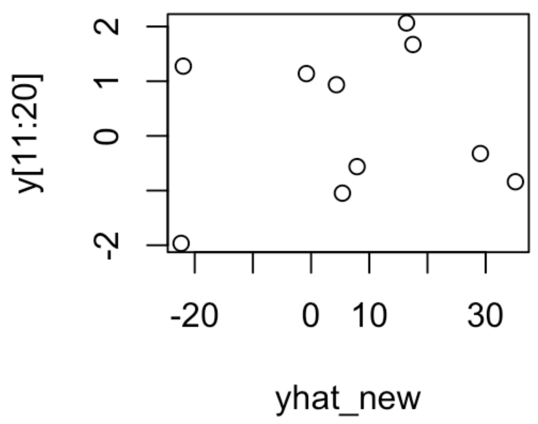
```
summary(lmout)
```

```


Call:
lm(formula = y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8, data = insample)

Residuals:
       1        2        3        4        5        6        7        8 
-0.08241 -0.09057 -0.03785  0.30636 -0.15326 -0.10111 -0.14845  0.27097 
       9       10 
-0.18182  0.21814 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   5.9056     1.0439   5.657    0.111
x1            1.5466     0.4457   3.470    0.179
x2            7.6723     1.2614   6.082    0.104
x3           -4.5446     0.7954  -5.714    0.110
x4           -0.7469     0.3113  -2.399    0.251
x5           11.4030     1.8873   6.042    0.104
x6           -0.6420     0.2324  -2.763    0.221
x7           -0.8118     0.3276  -2.478    0.244
x8            2.6710     0.5997   4.454    0.141

Residual standard error: 0.5658 on 1 degrees of freedom
Multiple R-squared:  0.9775,	Adjusted R-squared:  0.7974 
F-statistic: 5.428 on 8 and 1 DF,  p-value: 0.3209
```

Look first at the R^2 or even the adjusted R^2. Pretty high! Like our plot of $\hat{y}$ vs y, it looks like we have explained most of the variation in our dependent variable. But now note the f test's p-value at the bottom (the test of whether all the variables are jointly significant), or even the individual p-values for all the coefficients. They are all insignificant. All of our fit and R^2 is due just to over-fitting and chance.

However, as we said, often we don't have nice statistics for complicated models to fall back on. So we can instead test our fit by out-of-sample testing: let's now apply our in-sample-derived $\hat{\beta}$ coefficients to the new, out-sample x variables and try to match the $\hat{y}$ to the out-sample y values:

```
yhat_new <- predict(lmout,newdata=outsample)
plot(yhat_new,y[11:20])
```



Now we can see the truth: our out-of-sample predictions are junk. All the apparent fit between $\hat{y}$ and y in-sample was due to over-fitting, and the out-of-sample test reveals it. It was not any universal property to our population we were finding; we were just matching the random noise in y using a bunch of random x variables.

Cross-validation

Because it is so easy to over-fit data using lots of variables, and because many complex models like the ones we will be examining shortly don't have well-established statistical properties, we often use out-of-sample accuracy as the benchmark. The nice thing about out-of-sample accuracy is that it doesn't need to know anything about the model, and indeed competitions like the Netflix Challenge (which tasked people with doing a better job predicting what new movies customers would like based on their past ratings) just treated models as black boxes: you are given some in-sample data, you fit your model, and then you are given some new out-sample data to predict on.

Apart from competitions where other people set the structure, there are a number of common practices for validating one's own model using out-of-sample testing with one's own data. In general this is referred to as *cross-validation*, and what one ususally does is split the data into repeated in-sample and out-sample groups, and for each run you measure, for instance, the *mean absolute error* (MAE) between y and $\hat{y}$ in the outsample: $1/n \sum_{i=1}^{n} |y_i - \hat{y_i}|$, or perhaps the *mean squared error* (MSE): $1/n \sum_{i=1}^{n} (y_i - \hat{y_i})^2$. So that your results are not too dependent on getting lucky with a single in-sample/out-sample division, you do this whole process a few times, and report the overall average accuracy.

In *k-fold cross-validation*, you divide your data into k equal chunks, and take all but one of those chunks as your in-sample, and test your model on the remaining chunk; you then repeat this k times, leaving out one chunk each time as your out-sample. A common value of k is 5 or 10, for some reason. Another variant is to make k equal to n: that is, each out-sample is just a single observation, and you fit the model each time on the $n-1$ other observations.

We will return to these validation procedures in the next two lessons, where we examine more complex models that require out-of-sample validation rather than simply trusting the in-sample results.