210A Week 4 Notes

The Gordian Knot

Recall that last week we talked about standard error. Conceptually, standard error is the standard deviation of repeated estimates of a parameter. So if you're interested in mean and you collected data many times, *se* of mean would be the σ of your sample means. Under certain conditions we think we know what standard error looks like, $\sigma/\sqrt{n}$. Since this formula is simple, *se* is easy to calculate.

Other times it's prohibitively difficult to formally define *se* of a parameter because the parameter is exotic. Alternately, we have an idea of what the formula is but we don't know if we can trust it. "Exotic" can actually be something that seems like it should be simple. For instance, a few years ago the major radio companies signed a consent decree with the FCC to devote more airtime to independent music. Myself and my friends at the Future of Music Coalition were interested in whether it was really happening. We collected data on all the songs played every quarter for the last few years and figured out how many times each song was played (spins) and who owned the song. I then counted the total number of spins on indy labels vs major labels.

OK, this gives me a trend line, and in fact the share of airplay for the majors does decrease after the consent decree. But is this distinguishable from random noise? What I really want to have is not just a trendline but some measure of error around the trendline. I'd like a 95% confidence interval, which is basically the estimate $\pm 2se$. The end result is here, but it took an unusual route to get there.

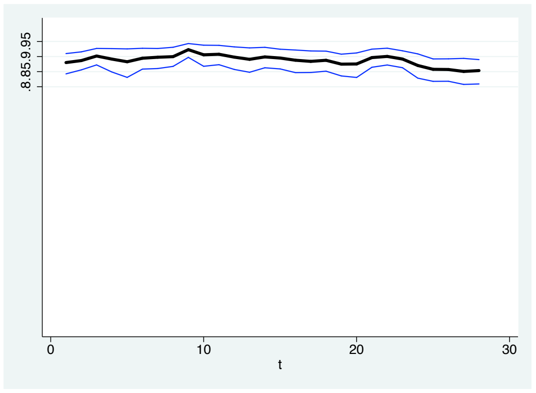

Figure 1: Major Label Share of Radio Airplay by Quarter

This seems really simple since all I want is the *se* of a proportion (or a mean of a binary variable). What could be simpler? Just take $\sigma/\sqrt{n}$, which for a binary we already know is $\leq .5/\sqrt{n}$. But the problem is what is "n"? Is it n_{spins} or is it n_{songs}? Actually it's neither of these and it's really hard to say what it is, but $n_{hitsongs}$ is a reasonable approximation. It seems like n should be of spins but spins are of particular songs, with n_{spins} per song following a count. Since spins are within songs, this violates the assumption of statistical independence from one spin to the next.

It's a little easier to understand with a concrete example. The #1 song of 2007 was "Hey There Delilah" by the Plain White T's which had about 1% of all the airplay in 2007. The Plain White Ts happen to be on Disney, which is technically an independent label.

So about a tenth of all non-major-label airplay was from this one song. Now imagine that the band just happened to have signed to Warner instead of Disney, or that the singer got hit by a bus on his way to the recording studio, or simply that the stochastic dynamics of music popularity had gone a little differently. If just this one song didn't exist or had signed to a different label, it would change the "major label share" for 2007 appreciably. In contrast, the vast majority of songs get only a few spins and you could change them without discernable effect. Technically what we're saying is that the spins aren't independent as they are clustered within songs.

How do you create standard error for that? There are lots of similar situations. Defining standard error for the percentage of America's mayors who are women is easy. Defining standard error for the percentage of Americans who live in cities where the mayor is a women is hard. New York City has almost 3% of the US population so if the mayor were a woman this would appreciably change the rate of Americans with a female mayor, whereas Cedar Rapids has about .05% of the US population so a woman as mayor there wouldn't change the big picture that much. In either case we'd be talking about just one mayor, but the number of citizens under the mayor's jurisdiction is quite different. Asking what proportion of radio airplay was from songs on indy labels or what percentage of Americans have a woman mayor *seem* like really simple questions but they're hard. Of course we can also think of questions that seem hard and are hard. So what do we do with these things?

In Turkey, Alexander the Great heard a local legend that whoever untied a particular knot would be king. The knot was impossible to untie, so Alexander took out his sword and cut it in half. For some statistical problems, standard error is a lot like this. Conventional standard error techniques are about understanding what the standard error is, but some error structures are almost impossible to understand. Bootstrapping is just a brute force technique where you don't need to understand it because you just plow right through. Things like ratios based on underlying counts are the Gordian knots of statistics – very difficult to understand the standard error, but bootstrapping still works through brute force.

Bootstrapping

Bootstrapping makes a lot of sense when you go back to basics and recall the *meaning* of standard error. It's not just some black-boxed magic number that feeds into our definition of significance, rather it's the standard deviation of repeated estimates of a parameter. Often, we can define this formally which is mathematical standard error, but other times it makes sense to take it literally and actually do it, which is bootstrapped standard error.

Or rather, almost do it. It's obviously unfeasible to actually replicate studies hundreds of times, to do hundreds of independent samples. Instead we can take hundreds of "resamples" from our actual sample. This is what bootstrapping is, you take hundreds of samples from your data, calculate the parameter, and treat these parameter estimates as a distribution. Bootstrapped standard error is just the standard deviation of the distribution of parameter estimates.

It's a fairly simple idea and if you know how to program you could do it from scratch in a few hours. Fortunately Stata gives us two commands that do it for us, "bsample" and "bootstrap." Here's how it works. Note that bootstrapping is an algorithm, not a formula. Also note that because it relies on randomness you'll get slightly different results every time, though as you increase the number of iterations it becomes very robust.

1. Take your dataset of size n and sample with replacement to create a new dataset, also of size n. Because you are sampling with replacement, each original case may appear

in the resample zero times, one time, or multiple times.

2. Estimate your parameter and write it down somewhere

3. Repeat this at least 50 times, preferably 1000

4. Take the list and treat it as a distribution. The standard deviation is your bootstrapped standard error

"bootstrap" and "bsample" Stata syntax

Stata gives us two commands for bootstrapping. "bootstrap" is easy to use and completely automates bootstrapping but it isn't very flexible. It's really good for regressions but doesn't do much anything else.

"bsample" just draws the bootstrap samples and you have to do the rest, however you can do whatever crazy thing you want. It's really only useful as part of a program. In parallel to the basic algorithm above, the program you write should look like this:

1. bsample

2. calculate the parameter and record it using "file write", "postfile", "matrix", "shell echo", or "estout"

3. repeat the first two steps 1000 times

4. treat your 1000 recorded results as a dataset and get the standard deviation

"bootstrap" does all four steps in a single command (actually, a prefix to other commands). The reason you might prefer to use "bsample" is that you can do whatever you want in step 2, and you aren't limited to things that work with "bootstrap". Fortunately, the most common operations (including regressions) work seamlessly with "bootstrap."

Here's an example with the auto data:

```
sysuse auto, clear
regress weight
*to bootstrap the same thing add bootstrap as prefix
bootstrap: regress weight
*can change number of replications
bootstrap, reps(100): regress weight
*can stratify the bsample by a variable
bootstrap, strata(foreign): regress weight
*also can save results to a file so you can open the file and do things manually
* like find the IQR or plot a histogram of the bstrap estimates
bootstrap, reps(100) saving(bsauto, replace): regress weight
use bsauto, clear
histogram _b_cons
sum _b_cons, detail
```

We can do the same thing with "bsample" but it's a bit more complicated. The advantage is that we have more flexibility in that the line that currently reads "quietly sum weight" could be just about anything.

```
tempname memhold
postfile ‘memhold’ meanweight using bsauto, replace
forvalues counter=1/100 {
 sysuse auto, clear
 bsample
 disp "iteration # ‘counter’"
 quietly sum weight
 post ‘memhold’ (‘r(mean)’)
}
postclose ‘memhold’
use bsauto, clear
sum meanweight
histogram meanweight
```

Scope conditions for bootstrapping

Bootstrapping can solve a lot of problems, but not all problems. Basically, bootstrapping can give you standard errors for *weird* data or *weird* parameters, but not *bad* data. So bootstrapping won't do anything to fix biased samples, or biased or confusing wording of survey items, or really small samples. Basically it can't deal with things that you know are just plain wrong.

What bootstrapping *can* help a lot with is things that you know are right, but they're just hard to completely get your head around. Our example was a ratio based on underlying count data. There's nothing wrong with this, it's just hard to understand. Likewise, with a lot of network techniques and exotic regression techniques, the standard error can be hard to understand. However it's OK to bootstrap this.

If bootstrapping is so great, why don't we do it all the time? Basically, because it's slow and most of the time we don't need to as most problems we understand well enough that mathematical standard error works great. However we could also legitimately bootstrap these simple problems if we wanted to.

Extensions of the bootstrapping logic

Bootstrapping is a specific procedure, but it's a simple idea so you can extend it by analogy to solve a lot of problems. Generally speaking, you can often exploit repeated random draws to get a handle on difficult issues that may or may not involve hypothesis testing or even the central limit theorem.

For instance, one of the issues in my book is whether what music to play is decided by individual radio stations or by the corporate office. Without getting into the details, there are multilevel techniques that let you see how much of the statistical action is clustered within identifiable groups. Because I'm interested in *when* stations start playing songs I used survival analysis with shared frailty, the overall level of which is summarized by θ. Usually, θ has a pretty straightforward interpretation but it wasn't applicable in my case because it assumes that there is just one thing spreading through the population and also that there are no small clusters – and my analysis violated both assumptions.[1] In order to interpret θ, I generated several random versions of the data that were identical to the real data except

[1] I have data on hundreds of songs, not just one. Likewise, while some radio companies are very large, there are many that have only one or two stations.

that there was only random clustering. I did this by keeping the actual data, but shuffling it so I keep the distributions of the variables but any association between them is random.[2] After doing this a hundred times I have a distribution for how likely different levels of θ are given that there is actually no clustering, and so I have a nonparametric interpretation of my results for which it doesn't matter that my analysis violates the assumptions necessary for the conventional interpretation. In my case it turns out that the actual level of clustering is entirely in line with a model of no corporate-level decision-making. My actual code is more complicated, but here's a simplified version of the important part:[3]

```
forvalues run=1/100 {
 disp "iteration # ‘run’ of 100"
 *next 2 lines shuffle the data
 gen sortorder=uniform()
 sort sortorder
 *next 4 lines displace the cluster var to the (randomly shuffled) neighbor
 gen o2=owner_n[_n-1]
 replace o2=owner[_N] in 1/1
 drop owner_n sortorder
 ren o2 owner_n
 *do the analysis and record theta in a results file
 quietly streg x1, shared(owner_n) distribution(exponential) iterate($iterations)
 shell echo "‘run’ ‘e(theta)’ ‘e(ic)’" >> _results_resample.txt
}
```

Similar approaches are a good idea when you're dealing with any kind of nonlinear complex system, including agent-based modeling or social networks. You can create many random datasets based on certain assumptions, measure some key index, then see how far off the empirical index is in terms of the distribution based on random data. This allows standard error for things that are too complex to otherwise measure.

You can also use these approaches not just for the usual application of standard error (i.e., hypothesis testing) but also for measurement. For instance in several recent papers, Ezra Zuckerman was interested in the decline of the Hollywood studio system. The usual date is the 1948 *Paramount* decision, but this wasn't really the end of the studio system so much as the beginning of the end. Rather than just calling it 1948 or some other arbitrary date, Ezra recognized that the studio system gradually faded away and so he created a metric that captured this subtlety. His measure was premised on the recognition that the studio system was at its core a system of keeping talent on salary. Therefore the studio system should be characterized by very high levels of repeat collaboration between studio and actor. Ezra measured this, but it's far too simple to simply say that X% of films involved an actor and a studio who had previously worked together. Among other things, the number of films per worker has gone from a Poisson during the studio system to a power-law at the present, which implies that more performances will be debuts, and thus ipso facto not repeat

[2]When we learn χ^2 you'll see how this approach is roughly analogous to doing a bootstrapped χ^2.

[3]Note that in this code I use "shell echo" instead of "postfile" as in the bsample example above. You're supposed to use "postfile" for this sort of thing but I'm lazy/ignorant and sometimes find it easier to use a standard Unix command than to learn a new piece of Stata syntax. The "shell" command passes commands to the operating system. This can be very easy and flexible if you're comfortable with your OS command line, but it has the downside that using it makes Stata for Windows do-files incompatible with Stata for Mac/Linux, and vice versa.

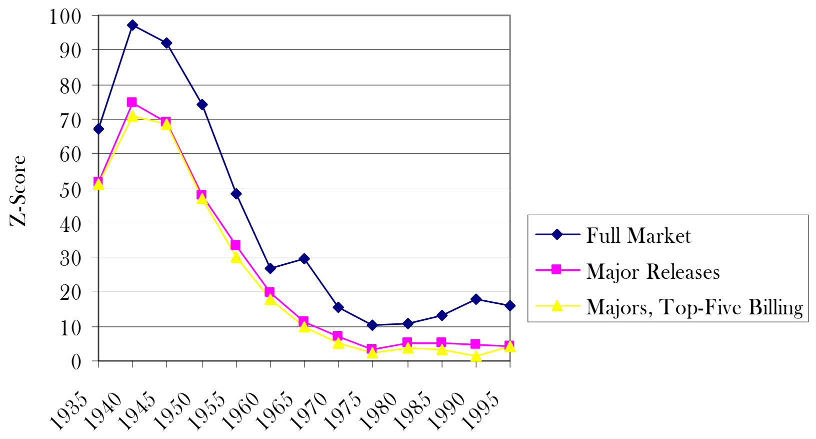

Figure 2: Z-Scores for Concentration of Actors with Particular Studio
Source: Zuckerman, Ezra W. 2005. “Typecasting and Generalism in Firm and Market: Career-Based Career Concentration in the Feature Film Industry, 1935-1995.” *Research in the Sociology of Organizations* 23:173-216.

collaborations. However it’s hard to write a mathematical function that precisely describes and adjusts for this bias.

Ezra’s solution was very similar to my own (or rather, my approach is similar to his).[4] He kept the observed distribution of individual film careers and studio prolifness but created random association between them. He did this hundreds of times and created a distribution for how likely were different levels of repeat collaboration. By comparing the empirical repeat collaboration to the distributions of repeat collaboration under randomness, he had a good measure of the strength of the tendency to repeatedly collaborate. Note that it’s not that he was testing *if* there was a special tendency for repeat collaboration – this isn’t really in doubt, even for post-studio Hollywood. Rather, Ezra was using this as a metric of the strength of association in Hollywood over time, which is always strong but is stronger at some times than at others.

The approach used by myself, Zuckerman, and Fernandez is in some ways a more general version of the Quadratic Assignment Procedure (QAP) which is useful for some social network problems (specifically the kind that might otherwise be approached as a cross-classified multilevel problem). It’s especially useful for relatively unipartite networks with non-sparse data (preferably with continuous values). That is, it’s great for import/export flows between countries, but don’t even think about applying it to the Netflix challenge. The way it works is you start with a square network matrix (i.e., each person or country is both a row and a column). You scramble the values of the matrix such that columns swap values with each other and rows swap values with each other. You use the scrambled matrix values for your dependent variable, the observed matrix values for your independent variable, and run a regression. You record the regression model, then repeat the process a thousand times. The distribution of your regression models gives you a distribution of parameter estimates under

[4]Ezra in turn got the idea from a study of homophily in workplace social capital.
Fernandez, Roberto M., Emilio J. Castilla, and Paul Moore. 2000. “Social Capital at Work: Networks and Employment at a Phone Center.” *American Journal of Sociology* 105:1288-1356.

the null hypothesis and let's you see how far outside of it the model based entirely on the observed data falls.